A Dynamic Service Composition Model for Adaptive Systems in Mobile Computing Environments

Nanxi Chen and Siobhán Clarke

Distributed Systems Group, SCSS
Trinity College, Dublin, Ireland
nchen@tcd.ie, Siobhan.Clarke@scss.tcd.ie

Abstract. Service-based applications must be adaptable to cope with the dynamic environments in which they reside. Dynamic service composition is a common solution to achieving adaptation, but it is challenging in mobile ad hoc network (MANET) environments where devices are resource-constrained and mobile. Existing solutions to dynamic service composition predefine the multiple configurations that may be possible, but this requires knowledge of the configurations a-priori. Alternatively, some solutions provide on-demand composition configurations, but they depend on central entities which are inappropriate in MANET environments. We propose a decentralized service composition model, in which a system dynamically adapts its business process by composing its fragments on-demand, as appropriate to the constraints of the service consumer and service providers. Results show a high composition success rate for the service compositions in high mobility environments.

Keywords: Service composition·Distributed·MANET·Overlay networks.

1 Introduction

Extensive use of mobile devices, coupled with advances in wireless technology like Wi-Fi direct, increase the potential for shared ownership applications for mobile ad hoc networks (MANETs) [6]. Devices can employ computational resources in a network to accomplish not only data routing tasks but also a complex user task with value-added services. A widely accepted mechanism to carry out such user tasks is service-based applications (SBAs), in which complex tasks are modelled as loosely-coupled networks of services. A SBA provides appropriate functionalities to consumers by composing cooperating services.

Typical MANET environments are dynamic; mobile SBAs must be adaptable to cope with potential changes in their dynamic operating environments (e.g. topology changes, network disconnections or service failures). Centralized service management for traditional adaptive systems is not applicable to MANETs, as device mobility is likely to be unpredictable, with devices joining and leaving the network at any time. There is, therefore, no guarantee that a suitably resource-rich central node will be available for the duration of a complex service provision.

X. Franch et al. (Eds.): ICSOC 2014, LNCS 8831, pp. 93–107, 2014.

A number of approaches to *distributed service composition* [2], [15], [24], [25] have been proposed. They support the specification of a full composition request as an abstract workflow and distribute it through the network. They can also partition a workflow into independent parts and then assign these parts to corresponding nodes. However, most workflow-based distributed service composition approaches have not addressed adaptation in abstract workflows. Thus, existing systems may fail or have to rediscover service providers, if a service query in such a request is not matched. *Dynamic service compositions* [8], [13], [16], cope with the adaptation problem using task planning algorithms, or graph-theory based techniques. However, graph-theory based techniques have limited support to analyze multiple input/output (I/O) dependencies among services [13]. Moreover, existing task planning algorithms either are constrained to sequential composition or require central knowledge bases.

In this paper, we introduce a service composition algorithm to form and adapt a full business process for a complex user task on-the-fly, without the requirement for a central node. Forming a business process relies on a novel overlay network named Semantic Service Overlay Network (SSON), which clusters semantically similar nodes and semantically dependent nodes for dynamic composition. On SSON networks, a service provider can adapt composition requests and generate potential execution fragments that can be selected to build global execution paths. New service providers may be discovered to replace any that result in composition failures or service outages, composing them into the current execution.

The advantage of this work is that programmers can develop mobile applications structured based on complex user tasks, instead of concretely specifying the possible workflows and configurations in advance. In addition, the approach allows new services not known a-priori to add to the environment at composition as well as execution time, increasing the composition success rate. The approach can be used in application scenarios where service providers are intermittent, or where there are dynamic complex applications with system configurations that cannot be generated a-priori, such as automotive and wearable mobile devices. Taking examples from the automotive domain, systems can benefit from cooperation between vehicles to produce information such as forward collision warning.

The main contribution of this work is a decentralized mechanism to enable service compositions. Our evaluation compares our service composition approach with a dynamic service composition model. The results show that service compositions can be adapted in a decentralized manner and provide a high composition success ratio.

The reminder of this paper is organized as follows. Section 2 introduces the system model. Section 3 illustrates a semantic-based mechanism that supports service composition. Section 4 shows the service composition algorithm and the strategy of service execution management. Section 5 presents the evaluation and result. Section 6 discusses related work. Section 7 summaries this work.

2 System Model Overview

In the service-based environment considered in this paper, a complex user task can be supported with the collaboration of two or more basic services that are composed into a composite service. To enable such collaboration, a service composition can specify complex user tasks as abstract workflows. In much of the existing work that relies on abstract workflows, systems execute ordered service queries individually in a predefined succession [6]. Each of the queries can match with either one basic service, or trigger a process to generate a composite service when there is no service that suffices independently. Such a service matching scheme can be categorized as one-to-N matching (N≥1). This paper investigates a novel, decentralized dynamic L-to-M (L≥1, M≥1) matching scheme for service composition. Instead of matching one service query to N basic service(s), our service composition model matches L service queries to M services, by adapting abstract workflows and their concrete implementation details.

Consider a navigation task as an example. This navigation has a service query list (abstract workflow) including two requirements: a GetLocation service and, a Navigator service that needs the results of a GetLocation service as input. In service composition, if a system can only find a Navigator service that uses both location data and map data as inputs, the system should be able to adapt the original service query list, by adding a requirement for getting map data.

The theory behind this L-to-M scheme is similar to that of semantic-based service compositions [12], but we realize it without needing a centralized reasoning mechanism, and systems can adapt a completed composition during execution. This is a non-trivial challenge as it is based on the assumption that no single node can maintain full knowledge of the network for execution planning. Our approach relies on local nodes receiving a global composition request, and generating the potential fragments of the global execution plan. Network knowledge can be gradually learned through interactions between participating nodes, and each node can adjust a composition request when new knowledge becomes available. Our main hypothesis is that decentralized service compositions can reduce composition failures from mismatched composition requests.

This work is supported by a three-layer composition structure, as illustrated in Fig.1. Specifically, we have defined an overlay network called a Semantic Service Overlay Network (SSON). A SSON can be constructed from service descriptions which are semantically annotated using ontology concepts. We define these concepts using standard taxonomies such as SIC[1] to outline the goal of services. Section 3 elaborates more on how to structure and maintain a SSON.

The lowest layer in the service composition structure is the service layer, which includes services with annotated service descriptions. In particular, we annotate the service with its functionality, inputs and outputs. Naturally, these annotated elements can have different ontology concepts taken from a common standard ontology. For example, an on-line meal order service can capture concepts such as: restaurant meals delivery (functionality), address (input) and price (output),

[1] `http://www.epa.gov/envirofw/html/sic_lkup.html`

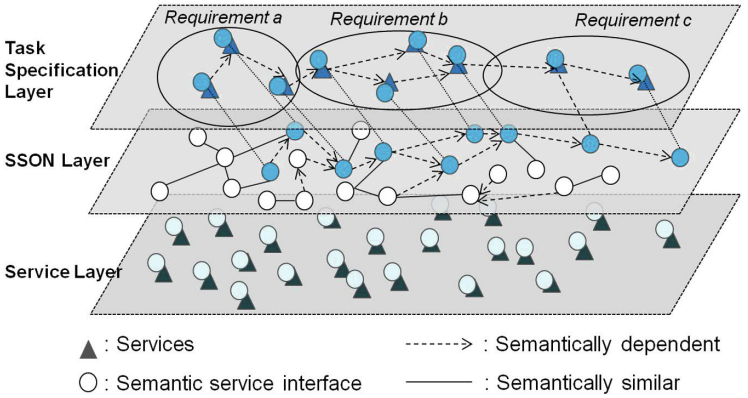

Fig. 1. Overview of Service Composition Model

which are drawn from an online NAICS [2] and Restaurant ontology[3]. The SSON layer is built over the service layer, linking different services based on their semantic similarity and dependency. Services are semantically similar if they provide similar functionalities or they require similar input data and produce outputs with similar types. Two services are semantically dependent if one's output has the same semantic type as the other's input. In other words, it is possible for one to use the others result to execute its own operations. The highest layer is a user task specification layer where a set of composition requirements are defined as a task specification, with the SSON layer aggregating appropriate services for the task specified, as described in Section 4.

For the service model, we describe a service $S =< S_{id}, IN, OUT >$, consisting of a service identification S_{id} and two sets capturing I/O parameters. Semantic annotations for a service are defined in a *Service Annotation Profile* $AP =< S_{id}, AF, AIN, AOUT >$, where AF, AIN, and $AOUT$ are ontology concepts mapped to corresponding functionalities and I/O parameters in a service description. APs inform service compositions about services' logical semantic interfaces, and can be described by semantically rich data models, such as OWL-S and SAWSDL.

3 Semantic Service Overlay Network (SSON)

Research on service composition has explored overlay networks underpinning the service discovery process. There are existing different types of overlay networks to this end, such as service-specific overlay networks [2], service overlay networks [13], and semantic overlay networks (SONs) [5]. SSON is an extension of SONs which were originally explored to improve service discovery performance for Peer-to-Peer (P2P) networks. A SON is a logical network based on similarities between peers shared content, which the network uses to organize peers and improve

[2] `http://www.census.gov/eos/www/naics/index.html`
[3] `http://wise.vub.ac.be/ontologies/restaurant.owl`

content-based search. The core notion of a SON is to bunch similar peers, making query processes discover bunches instead of individual peers for faster locating. Current research on facilitating distributed service discovery with SONs, like DHT-SON [18] and SDSD [4], shows a SON can be structured at comparable cost to that of producing a normal network by using probe messages.

This proposed SSON supports service composition by introducing the idea of I/O dependencies from service overlay networks [13], [23] which create links between service providers by matching I/O parameters. A SSON combines service overlay networks with normal SON activities. I/O parameters-dependent links can be built as a byproduct of general SONs. A SSON is structured by linking semantically-related services, and the relationship between services can be classified as similarity or dependency. The former represents two service sharing similar functionalities like that defined in traditional SONs; the latter indicates two services with potential I/O data dependencies. Semantic links rely on matchmaking techniques to be established.

3.1 Matchmaking Models

In this work, we introduce a *deductive matchmaking model* (MatchAP), combined with a distributed similarity-based model called ERGOT [19]. ERGOT uses a similarity function $Csim(c_1, c_2) \in [0, 1]$ that returns whether two ontology concepts (c_1 and c_2) are similar. The MatchAP model explores semantic connections (similarity and dependency) between two semantic service interfaces AP_1 and AP_2. It matches the inputs of AP_1 to that of AP_2, as well as matching their outputs and functionalities. It also matches the inputs (respectively, the outputs) of AP_1 to the outputs (respectively, the inputs) of AP_2 to determine any dependency between the two interfaces. We use a matching function [19] between the parameters in two sets X_1 and X_2:

$$ParamSim(X_1, X_2) = \sum_{b \in X_2} \max_{a \in X_1} Csim(a, b) \tag{1}$$

where $AP_1 =< S_1, AF_1, AIN_1, AOUT_1 >$ and $AP_2 =< S_2, AF_2, AIN_2, AOUT_2 >$. The matching function between two APs can then be defined by using of Equation1:

$$\begin{aligned} &MatchAP(AP_1, AP_2) = \alpha ParamSim(AF_1, AF_2) \\ &+\beta(ParamSim(AIN_1, AIN_2) + ParamSim(AOUT_1, AOUT_2)) \\ &+\gamma ParamSim(AIN_1, AOUT_2) \end{aligned} \tag{2}$$

where the parameters α, β, and γ are used to weight similarity. Depending on values returned by function $MatchAP(AP_1, AP_2)$, we state that if $MatchAP(AP_1, AP_2)$ is larger than a threshold value Θ, a semantic link can be built between the services, and they become semantic neighbours to each other. A semantic link $A \rightarrow B$ can be ranked. We define five different ranks for semantic links that are listed below, taking into account their match types, which extend the definition from a conventional SON [4]:

- *R0_same*: A and B provide the **same functionalities** and ask for the same sets of input data. (e.g. A: Get Location, B: Get Address)
- *R1_reqI*: A provides the same functionalities with B , but asks for **additional input data** with respect to that of B. (e.g. A: Get Location, B: Get Address by Name and Phone Number)
- *R2_share*: A and B have **shared functionalities**. (e.g. A: Navigator, B: Route Planner)
- *R3_dep*: A depends on B's execution result. (e.g. A: Navigator, B: Get Location)
- *R4_in*: A can provide input data to B. (e.g. A: Navigator, B: Route Render)

3.2 SSON Construction

Nodes can join or leave the network over time, so they should be in a position to discover their semantic neighbours. When a peer joins a new network, it advertises its services by initializing a probe service request using the information in the APs of the services and finding its semantic neighbours in the network. The request also includes a threshold value for semantic matchmaking. Such a service request query only operates once when the peer joins the network. As soon as a peer has established semantic links with other peers, other newcomers are able to take advantage of this peers knowledge through their own probe service requests. For example, suppose a Navigator service relies on locations and map data as inputs to generate navigation information between locations. The semantic links Navigator $\rightarrow$ Get Location service can be created with a rank: *R3_dep* if the latter can provide any of the inputs. Semantic neighbours are neighbouring logically but physically it is also possible that multiple services with semantic relations may be published in the same service provider (peer).

Considering peers' mobility and the dynamic environment in which they reside, overlay management protocols like CYCLON [22] and some stabilization protocols can be used to monitor the neighbours presence and to update the list of semantic neighbours on peers. Updating the list of semantic neighbours can trigger system adaptations that will be illustrated in Section 4.3.

4 Decentralized Service Composition

MANET environments cannot guarantee to provide a single, continuously accessible node to serve a service composition as a central entity because nodes may leave the network, or otherwise fail. This section introduces an alternative based on distributing the processing of service compositions, utilizing semantic links in SSON network.

4.1 Task Model

Complex user tasks are handled by a service composition system that receives the task's specification as an input request and composes value-added services. Our work specifies a task as $SC\_T =< T_{id}, T_{input}, T_{output}, \delta >$, where T_{id} is the

request's identification. The end-to-end inputs and outputs are represented as two sets: T_{input} and T_{output}. A set of operations δ is defined that summarise the task's goal, and an operation $Opt_i =< Opt_{id}, IN, OUT > \in \delta$ consists of a specification of operation name (Opt_{id}) and I/O parameters (IN and OUT). A composition request message containing a SC_T can be defined as $SC\_Req =< SC\_T, Log, Comp\_Cache >$, where Log accumulates message-passing history to prevent providers repeatedly processing a request for the same workflow. The $Comp\_Cache$ is used to resolve service composition for a single operation when a service for a requested operation is not found. It caches existing services to discover if a combination from them can match the single operation. In this work, we define a *Composer* role in a service composition process. Task requests can be passed from a composer to its neighbours and adapted by the composer.

Definition 1: A *Composer* is a node in a network who decides to participate in a composition process. It can cooperate with other composers in the network to dynamically generate and maintain the fragments of global execution paths. One node becomes a composer as soon as it successfully matches a received composition request from other nodes. The node resigns this role when the provided services have been executed, or no fragment remains in it.

4.2 Distributed Planning

Logical reasoning algorithms like forward- or backward chaining have previously been used to facilitate semantic-based service composition [12]. General implementation of reasoning forwards and backwards uses decision trees, for which systems require a global view of facts (available services) to build. Our approach leverages a decentralized *backward-chaining* mechanism to create composition plans using local knowledge for user tasks. The core function of this planning process is twofold: a) it aggregates potential providers for a composition request and dynamically adjusts composition requests hop-by-hop based on new local knowledge; b) it allows a potential provider to generate a list of fragments for system configurations. A fragment defines how the potential provider can compose with its semantic neighbours to provide functionalities for the consumer.

Potential Provider Aggregation. When a consumer launches a service composition request, a distributed strategy to devise composition plans for this request, piece by piece, is initiated. Specifically, service consumers initiate a composition request from task specifications. As can be seen in Algorithm 1 (a), an initiator sends the request over the network to discover potential providers who can produce (or partially produce) the set of requested end-to- end outputs. Afterwards, it waits for composers sending tokens to it. Each token represents a discovered plan of a completed execution path, or a completed branch in a parallel execution path. The initiator receives tokens, and starts service execution phases when a complete execution plan emerges (Line 6 in Algorithm 1 (a)).

When a node receives a composition request for the first time, it calls Algorithm 1(b) to decide how its published services could combine to match one or more operations specified in the request's task specification. If an operation is

List 1 : Algorithm 1 (a) - Initiator

```
1  Send SC_R
2  Set timer T
3  /* Waiting */
4  if T expires : the composition fails;
5  Receive a token and store it
6  if a complete composition exists : send input data and execute;
7  else : back to step 3;
```

List 2 : Algorithm 1 (b) - Service provider S

Input: Service Composition Request: SC_R
SC_R: $< R_{id}, T_{input}, T_{output}, \delta, Log, Comp\_Cache >, Opt_i \in \delta$
Output: (i) template information and (ii) an updated request

```
1  /* Listening */
2  S Receives SC_R from Composer Q and log(Log);
3  if ComposerMode_S = ON: produceTemplateInfo (SC_R);    //Output (i)
4  else : for each operation Opt[i]{
5     if S supports no operation: back to step 1;
6     else{ ComposerMode_S := ON
7        if S supports Opt[i]'s output type but Opt[i]'s functionality:
8           Strategy 1- AddtoComp_Cache (S);
9        if S (or S+ Comp_Cache) can provide full functionality for Opt[i]: {
10          eliminate Opt[i]
11          if Opt[i] is the last operation in a branch : sendToken(Initiator);
12          if S (or S+ Comp_Cache) requires extra input data which
13          is not specified in Opt[i]: Strategy 2- create a new operation;
14          produceTemplateInfo (SC_R) }                    //Output (i)
15       replyToSender(Q)
16       updateTask(SC_R, S)                                //Output (ii)
17       SendOutList :=getSemanticNeighbours(R3_dep)
18       sendRequest (SC_R, SendOutList) };
19 };
```

successfully matched, the node becomes a composer. This new composer then stores the received request, eliminates the matched operation of the request and sends out a new request. It also draws up template information for creating configuration fragments. Therefore, the number of operations in a request reduces hop by hop. If the eliminated operation is the last operation of a workflow branch in the request, the composer sends a token to the initiator.

A node has a chance to provide a full service for an operation by combining to other peers in the network when it can only provide a partial service for that operation. To this end, we take into account two situations and apply corresponding two strategies (Line 8 and Line 13 in Algorithm 1 (b)). These strategies allow composers to adapt the abstract workflow on-the-fly. To illustrate such adaptation in our backward planning process, we present a brief example scenario: finding a restaurant and routing to it. Table 1 shows the operations defined in the composition request and the original abstract workflow. It also illustrates available service providers in the example scenario. *We assume there is no provider in the network that can work alone to serve operation* o_C.

Table 1. An example scenario: finding a restaurant and routing to it

Abstract workflow: $o_A \rightarrow o_B \rightarrow o_C$ (deduced from the I&O dependencies)			
Operations	Functionality	Input	Output
o_A	Restaurant Recommender	Personal profiles (food preferences)	A list of restaurant
o_B	Get Location	Names	Addresses
o_C	Navigator	Addresses	Audio routing results
Available Providers	Functionality	Input	Output
Provider_A	Restaurant Recommender	Food preferences	A list of restaurant
Provider_B	Get Address	Names	Addresses
Provider_C	Navigator	Addresses+Map blocks	Text routing results
Provider_Y	Text to Audio	Plain text	Audio stream
Provider_X	Map cache	Name of place	Map blocks of the place

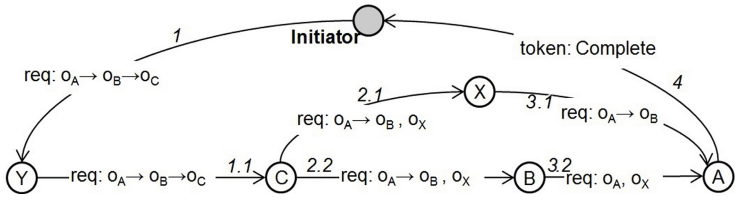

Fig. 2. An example of distributed backward task planning (req: a service composition request. An abstract workflow is implied by the request.)

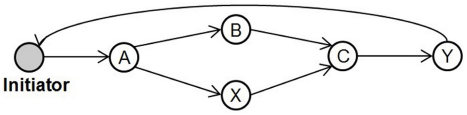

Fig. 3. Concrete workflow from a global perspective

As can be seen in Fig.2, the abstract workflow ($o_A \rightarrow o_B \rightarrow o_C$) is processed backward with the forwarding of composition requests. This process starts from an original composition request sent from the initiator (Arrow 1) to providers that can produce audio streams. Provider_Y receives the request and finds itself cannot support the full functionality of o_C. It applies *Strategy 1*, caching information to indicate its capabilities and forwarding the request to its semantic neighbours. Provider_C gets the request from Provider_Y. As shown in Table 1, Provider_C can fully serve o_C, but requires extra support on input data. Therefore, Provider_C uses *Strategy 2*, eliminating o_C from the request and adding a new operation for discovering the required input data. Afterwards, the updated request will be sent out to match the remainder of the operations. A complete execution path exists after Provider_A applies the requests sending from Provider_X and Provider_B. *During the distributed planning process, the original abstract workflow ($o_A \rightarrow o_B \rightarrow o_C$) is adapted as shown in Fig.3.*

Configuration Fragments Creation. A *configuration fragment* (CF) indicates the possible position of a service in execution paths by defining the pre-conditions and the post-conditions of the service's execution. It is created from template information that is one of the outputs of Algorithm 1(b). Template information includes a CF template that defines how many different pre-conditions and post-conditions the execution of the service has. It also contains a set of CF update instructions (CFUIs) for maintaining the list of CFs. To create CFs we propose two notions: *Pre-Conditional Neighbours (PreCNs)* and *Post-Conditional Neighbours (PostCNs)*. They are both extracted from a potential provider's semantic neighbours. A PreCN or a PostCN has input or output dependence on the potential provider, respectively.

Definition 2: A CF can be described as $CF =< CF_{id}, S\_Pre, S\_Post >$, where S_Pre represents a set of selected PreCNs as the entries of the CF; S_Post represents a set of selected PostCNs as the exits of the CF; and CF_{id} is an identification of the CF. An example of a CF for the scenario illustrated in Table 1 is shown in Fig.4(a).

A CF is created by interactions between composers. A node becomes a composer and gains template information (Output (i) in Algorithm 1(b)). This composer then starts the creation of CFs with this information. The composer regards the composition request sender as one PostCN in the S_Post of a CF and waits for replies from other nodes to select PreCNs for S_Pre. If a group of replies are received and the reply senders can satisfy the CF template in the template information, a CF is created. If more than one group of reply senders are available, a list of CF can be made by repeating the creation step.

4.3 Service Execution and Dynamic Adaptation of CFs

Traditional service composition techniques start service executions only after service binding has completed. The composition mechanism in this paper combines the service binding phase and the execution phase. Our previous work on opportunistic service composition illustrates a distributed execution model [10], [9] that allows systems to bind one service provider, directly executing its provided services, and then forwards the remainder of composition request on to the next node. The bound provider then waits for other providers to reply with messages that include their service functionality information. We apply the distributed execution model in this work with some extensions to discovery mechanisms. Instead of forwarding the rest of the request, the bound provider (composer) selects the best matched CF, sending its execution results to the nodes in the S_Post of the CF.

This paper provides dynamic adaptation mechanisms for systems. Global adaptation is realized by selecting adaptable local CFs hop-by-hop during service execution. The CFs of a composer can be adapted during composition planning phases and service execution phases. Such adaptation is modelled by *a MAPE (Monitor-Analyze-Plan-Execute)* loop, as shown in Fig 4(b). Composers can *monitor* adaptation trigger events with SSON, *analyze* these events when they appear and assign

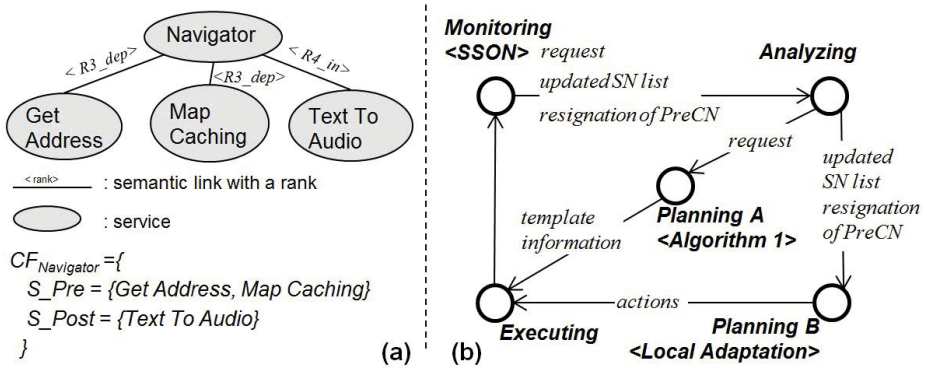

Fig. 4. (a) An example of the CF for Provider_C (Table 1), (b) CFs Adaptation (SN: Semantic Neighbours)

Table 2. Decentralized adaptation rules (Planning B)

Links	Composer's (A) adaptation actions to the new node (B)
R0_same	*Action 1*: A clones *the CFs and the composition request kept in A* to B. B becomes a composer.
R1_reqI	*Action 2*: A clones *the composition request kept in A* to B. B becomes a composer and calls Algorithm 1(b) to create CFs
R2_share	*Action 3*: A decides if the functionality it shares with B can support the composition. If so, do *Action 1 or Action 2* depending on the required input of B
R3_dep	*Action 4*: A send *the composition request updated by A* to B. B calls algorithm 1(b), deciding to take part in the composition or not.

the event information to corresponding *planning* algorithms. These algorithms generate local adaptation plans which are *executed* to adapt CFs.

Adaptation can be triggered by newly arrived composition requests. When an existing composer receives a new request for the same composition, it then analyses the request using its historical data (stored requests and existing template information), and directly generates template information (Planning A in Fig 4(b)). The CFUIs in template information are used to guide the execution of the adaptation. According to different kinds of new requests, we define four CFUIs for adaptation: clone, alter, wait and remove. For example, the Provider_A in our example scenario (as shown in Table 1) receives the request$< req : o_A, o_X >$, and creates a CF $(CF_1 : S\_Pre = \{Initiator\},\ S\_Post = \{B\})$. When a new request $< req : o_A \rightarrow o_B >$ is pushed to Provider_A, this new request will trigger adaptation to alter the CF_1 to $\{S\_Pre = \{Initiator\},\ S\_Post = \{B, X\}\}$.

This approach introduces a decentralized adaptation mechanism (Planning B in Fig 4(b)) that deals with *potential node failures* and *recently joined nodes* by adapting the CFs stored in composers. Node failures can be leaded by the selection of invalid CFs during service execution. A CF becomes invalid when the nodes that are recorded in it cannot be reached. If a semantic neighbour of a

composer leaves the physical network, this absence of the node can be detected through the management of SSON. The composer removes all the CFs that contain the absent node. If a composer resigns its role, it sends a message to all the PreCN and PostCN nodes, asking them to adapt their CFs by removing invalid CFs. A node can also join the network and engage in the composition at runtime. If a composer finds a new node in its semantic neighbour list, this new node can participate in the composition in different ways depending on the rank of the semantic link established between it and the composer (See Table 2).

5 Evaluation

Our approach maps a composition request to an L-to-M matching scheme to address the problem where a queried service cannot be matched during service composition. Solving one-to-one mis-matching has been investigated by several approaches, for example, discovery algorithms to find more potential providers and dynamic one-to- more composition [2], [13]. Our direction is similar in that it tries to compose a basic service to match a single service query. Thus, we conducted simulations to compare our solution against a Graph-Based (GB) approach [13]. The GB approach assumes a service provider entering a network by broadcasting its service information. The network can rely on a central directory that collects service information and maintains a global service network graph to receive composition requests. The directory finds a consecutive execution path out of the graph based on received requests. We measured the composition success ratio, which is the number of composition requests that are successfully processed with execution paths divided by the total number of requests.

We applied a simulation scenario [2] with services of alphabet converters and joiners. In this scenario, a converter service receives input A and produce output B ($A \rightarrow B$), and a joiner service receives inputs A and B generating C as an output ($A + B \rightarrow C$). We employed 7 alphabets to represent I/O parameters in the scenario; therefore, there are 21 different converter services and 35 different joiner services. We used the NS-3 simulator to study the efficiency of our self-adaptive approach and the GB approach. We simulated both approaches with the same limitation of the maximum hops of broadcasting during service publishing. We applied the random walk 2D mobility model, by which we control service density and the proportion of mobile nodes. The service density is the radio scope of a node divided by the whole field where all the nodes are located. This simulation ran 10 rounds with varying numbers for mobile nodes and 9 rounds with different service densities. Each of them is repeated 100 times with random providers' composition requests, and we report the average (see Fig.5). The number of operations in a request is set to 4. We used Self-Adapt to represent our approach in the following simulation study.

Fig.5 shows (a) the success ratio results with different service densities, and (b) the success ratios at 70% service density with the ratio of mobile nodes varying from 20% to 100%. These two studies only employed the converter services since the GB approach cannot model services with multiple I/O parameters.

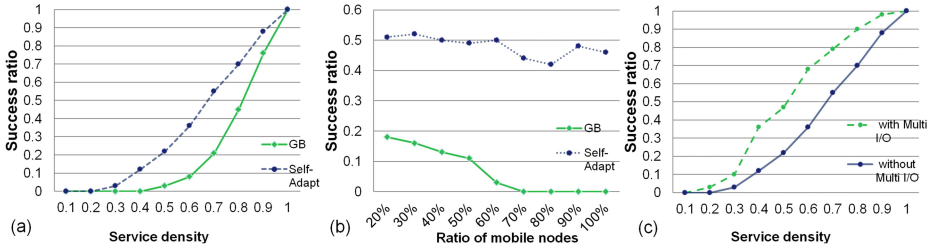

Fig. 5. The result of the study on composition success ratio

The result (a) shows that Self-Adapt has a higher success ratio, especially when the density of services is low, because the SSON network, benefited from its decentralized management, collects and mantains more service information than the service overlay network used in the GB approach. The result (b) suggests that Self-Adapt is more successful than the GB approach in high mobility scenarios. Fig.5(c) illustrates the success ratio with varying service density of services, comparing Self-Adapt using the multiple I/O strategy (see Section 4) with when it does not use this strategy. It shows that the use of the strategy results in a higher success ratio.

6 Related Work

Workflow-based adaptive systems [21] choose, or implement services from a predefined abstract workflow that determines the structure of services. The abstract workflow is implemented as a concrete (executable) workflow by selecting and composing requested services. Adaptation of concrete workflows has been explored in the literature [1], [3]. However, these require central entities for composition and an explicit static abstract workflow, which is usually created manually. Decentralized process management approaches [24], [25] explore distributed mechanisms, like process migration, to manage and update the execution of concrete workflows, which is close to our work in terms of service execution. However, they still need a well- defined system business process at deployment time. In our approach, the partial workflows that composers generate locally, distribute over participating service providers during service discovery phases to gradually devise a global one.

Dynamic service composition can also be reduced to an AI planning problem. Similar to our solution, decentralized planning approaches [7], [11], [20] form a global plan through merging fragments of plans that are created by individual service agents. However, with these approaches, programmers need to provide an explicitly defined goal for planning. The initial plan can become unreliable when the environment changes. Automatic re-planning schemes [12], [14], [17] allow plans to be adapted when matching services are unavailable, but existing approaches depend on central knowledge bases.

Considerable research effort has targeted dynamic service compositions supporting one-to-M ($M > 1$) matching while a matching basic service is not located. They usually define a composition result as a directed acyclic graph [8],

[13]. The nodes in a DAG represent services, and the edges show compositions between the collaborating services. Service composition is modelled as a problem of finding a shortest path (or the one with the lowest cost) from two services in the DAG. However, existing work has limited support for services with multiple I/O parameters. In addition, creating such DAG requires the aggregation of service specifications from a central registry. Al- Oqily [2] proposed a decentralized reasoning system for one-to-M ($M > 1$) matching, which is the closest work to us. It composes services using a Service-Specific Overlay Network built over P2P networks and enables self-organizing through management of the network. However, this approach is based on an assumption that every node in the network knows its geographic location, as service discovery is realized by broadcasting a request over its physical neighbours. Geographic locations usually can be obtained from location services like GIS, but these are not readily accessible for every node in the network. Our approach uses a semantic-based overlay network to discover logical neighbours instead of geographic ones.

7 Conclusion and Future Work

Distributed service composition approaches allow systems to perform complex user tasks without central entities, but do not fit well in dynamic environments. The distributed service composition algorithm proposed in this paper addresses this dynamic problem by adapting workflows during composition planning processes as well as service execution phases. This paper also proposed a SSON network to underpin such adaptation. The presented evaluation result shows an improvement of composition success rate comparing to an influential abstract workflow adaptation scheme for dynamic service composition.

In future work, first, we will investigate the performance of the proposed approach for different application scenarios and varying composition paths like liner, parallel and hybrid composition paths. Although this approach provides a higher success ratio than the GB approach, the cost of maintaining SSON networks and adaptation processes may outweigh the high success ratio benefits in some application scenarios. Second, our future research will include an interesting topic for Quality of Service (QoS)-aware adaptation to not only increase the composition ratio but also provide services with good quality.

References

1. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A service-oriented implementation of dynamic flexibility in workflows. In: Proc. Int. Conf. on On the Move to Meaningful Internet Systems (2006)
2. Al-Oqily, I., Karmouch, A.: A Decentralized Self-Organizing Service Composition for Autonomic Entities. ACM Trans. Auton. and Adapt. Syst. (2011)
3. Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M.: Context-aware workflow management. In: Proc. 7th Int. Conf. ICWE (2007)

4. Bianchini, D., Antonellis, V.D.: On-the-fly collaboration in distributed systems through service semantic overlay. In: Proc. 10th Int. Conf. Inf. Integr. Web-based Appl. Serv. (2008)
5. Crespo, A., Garcia-Molina, H.: Semantic overlay networks for p2p systems. Agents and Peer-to-Peer Computing (2005)
6. Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K.: A journey to highly dynamic, self-adaptive service-based applications. Autom. Softw. Eng. 15(3-4), 313–341 (2008)
7. El Falou, M., Bouzid, M., Mouaddib, A.I., Vidal, T.: A Distributed Planning Approach for Web Services Composition. In: 2010 IEEE Int. Conf. Web Serv. (2010)
8. Fujii, K., Suda, T.: Semantics-based dynamic service composition. IEEE Journal on Selected Areas in Communications, 2361–2372 (December 2005)
9. Groba, C., Clarke, S.: Opportunistic service composition in dynamic ad hoc environments. IEEE Trans. Services Computing PP(99), 1 (2014)
10. Groba, C., Clarke, S.: Opportunistic composition of sequentially-connected services in mobile computing environments. In: 2011 IEEE Web Services, ICWS (2011)
11. Helin, H., Klusch, M., Lopes, A., Fernández, A., Schumacher, M., Schuldt, H., Bergenti, F., Kinnunen, A.: CASCOM: Context-aware service co-ordination in mobile P2P environments. In: Eymann, T., Klügl, F., Lamersdorf, W., Klusch, M., Huhns, M.N. (eds.) MATES 2005. LNCS (LNAI), vol. 3550, pp. 242–243. Springer, Heidelberg (2005)
12. Hibner, A., Zielinski, K.: Semantic-based dynamic service composition and adaptation. In: 2007 IEEE Congress on Services, pp. 213–220 (2007)
13. Kalasapur, S., Kumar, M., Shirazi, B.A.: Dynamic Service Composition in Pervasive Computing. In: IEEE Trans. Parallel and Distributed Systems (2007)
14. Klusch, M., Gerber, A.: Semantic web service composition planning with owls-xplan. In: Proc. 1st Int. AAAI Fall Symp. Agents and the Semantic Web (2005)
15. Martin, D., Wutke, D., Leymann, F.: A Novel Approach to Decentralized Workflow Enactment. In: 12th Int.IEEE Enterp. Distrib. Object Comput. Conf. (2008)
16. Mokhtar, S., Liu, J.: QoS-aware dynamic service composition in ambient intelligence environments. In: Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng. (2005)
17. Peer, J.: A pop-based replanning agent for automatic web service composition. In: Proc. 2nd EU Conf. The Semantic Web: Research and Applications (2005)
18. Pirrò, G., Talia, D., Trunfio, P.: A DHT-based semantic overlay network for service discovery. Future Generation Computer Systems 28(4), 689–707 (2012)
19. Pirrò, G., Trunfio, P., Talia, D., Missier, P., Goble, C.: ERGOT: A Semantic-Based System for Service Discovery in Distributed Infrastructures. In: 10th IEEE/ACM Int. Conf. Clust. Cloud Grid Comput. (2010)
20. Poizat, P., Yan, Y.: Adaptive composition of conversational services through graph planning encoding. In: Proc. 4th Int. Conf. Leveraging Apps. of Formal Methods, Verification, and Validation - Volume Part II (2010)
21. Smanchat, S., Ling, S., Indrawan, M.: A survey on context-aware workflow adaptations. In: Proc. 6th Int. Conf. Adv. Mob. Comput. and Multimed. (2008)
22. Voulgaris, S., Gavidia, D., Steen, M.: CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. J. Netw. Syst. Manag. 13 (2005)
23. Wang, M., Li, B., Li, Z.: sFlow: Towards resource-efficient and agile service federation in service overlay networks. Distributed Computing Systems (2004)
24. Yu, W.: Scalable Services Orchestration with Continuation-Passing Messaging. In: 2009 First Int. Conf. Intensive Applications and Services, pp. 59–64 (April 2009)
25. Zaplata, S., Hamann, K.: Flexible execution of distributed business processes based on process instance migration. J. Syst. Integr. 1(3) (2010)