This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



# Goal-Driven Service Composition in Mobile and Pervasive Computing

Nanxi Chen, Nicolás Cardozo and Siobhán Clarke

**Abstract**—Mobile, pervasive computing environments respond to users' requirements by providing access to and composition of various services over networked devices. In such an environment, service composition needs to satisfy a request's goal, and be mobile-aware even throughout service discovery and service execution. A composite service also needs to be adaptable to cope with the environment's dynamic network topology. Existing composition solutions employ goal-oriented planning to provide flexible composition, and assign service providers at runtime, to avoid composition failure. However, these solutions have limited support for complex service flows and composite service adaptation.

This paper proposes a self-organizing, goal-driven service model for task resolution and execution in mobile pervasive environments. In particular, it proposes a decentralized heuristic planning algorithm based on backward-chaining to support flexible service discovery. Further, we introduce an adaptation architecture that allows execution paths to dynamically adapt, which reduces failures, and lessens re-execution effort for failure recovery. Simulation results show the suitability of the proposed mechanism in pervasive computing environments where providers are mobile, and it is uncertain what services are available. Our evaluation additionally reveals the model's limits with regard to network dynamism and resource constraints.

**Index Terms**—Services Composition, Requirements Driven Service Discovery, Pervasive computing, Mobile Computing.

## 1 Introduction

Pervasive computing environments respond to human users' requirements by providing access to various resources over networked systems. Such environments have evolved from closed (special purpose) and static to *open and dynamic*, embracing a large number of third-party mobile entities (e.g., wearable technologies, smart phones, etc.). Such open, mobile pervasive computing environments are likely to incorporate abundant functionalities and heterogeneous smart devices that have the potential to collaborate.

Service-oriented computing's (SOC) packaging of heterogeneous resources as services that are discoverable, accessible, and reusable has emerged as an important paradigm in pervasive computing environments [1] [2]. SOC provides unifying interfaces for services to ease users' access via communication networks. To address a particular requirement, a combination of multiple services may be required, and so a fully-functional service composition process will tackle potentially complex and mutable user requests [3] with flexible composition of value-added services, rather than simply providing information query or file transmission services.

Given the environment's openness and dynamism, such a composition process faces significant challenges (see the dash rectangle in Fig.1 ):

- **Flexible service composition**- services of interest are *independently* deployed and maintained by different service hosts that form a wireless network in ad hoc ways. An individual service host has only a local system view because of its limited communication ranges and resources. It is a challenge for service hosts to collaborate to find possibly more potential service providers to reduce composition failure.
- **Adaptable composites**- service hosts' *mobility* leads to changes to the network topology as well as the established communication channel between composite participants (service links). Such changes may result in service link loss during service execution, which further causes service execution path loss. Efficient interactions between composite participants are required to reduce the dependency on such an error-prone communication channel. In addition, composite services must be adaptable to increase the chance that results can be delivered even when a communication channel between their composite participants drops.

This paper focuses on flexible service composition and adaptable composites, assuming trustworthy service providers. The remaining issues in Fig.1, service heterogeneity and third-party providers, are out of this paper's scope and are discussed further in Section 5.

As a motivating scenario (Fig.2), Anne is in a shopping mall's car park with her 1-year-old son. She would like to get a step-free route to a shop that sells nappies and from there to the nearest baby-changing facility in the mall, using her smart watch. The mall offers an official website and a mobile application for information browsing. But Anne's smart watch is incompatible with the application and its screen is too small to display the route properly. However, there is a pervasive computing environment in the mall including various embedded devices owned by customers, shop clerks, taxi drivers, information desks, or house keep-

- *N. Chen, N. Cardozo and S. Clarke are with the Department of Computer Science, Trinity College, Dublin, Dublin 2, Dublin, Ireland. E-mail: nchen@tcd.ie, cardozon@scss.tcd.ie, Siobhan.Clarke@scss.tcd.ie*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



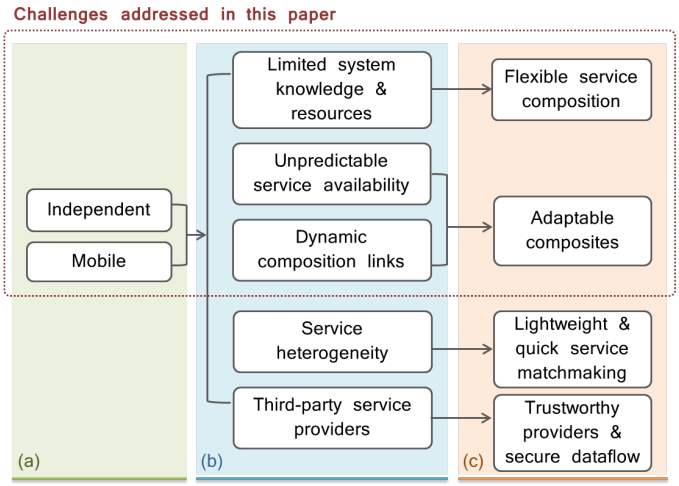


Fig. 1. Open issues in service composition in pervasive computing environment (a) service deployment device, (b) service composition problems, (c) service composition requirements; the red dash line represents the challenges addressed in this paper.

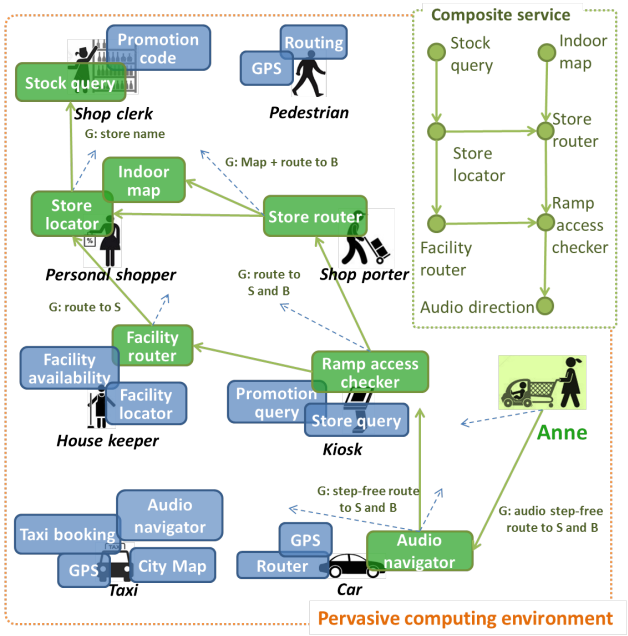


Fig. 2. Services and a service composite in a shopping mall's pervasive computing environment

ers. These devices can package their capabilities, like GPS, navigation, facility routing, taxi booking or indoor map to be accessible via network connections. Anne's smart watch has been configured to incorporate surrounding communication networks to make use of available resources [4]. Thus, her smart watch may be able to directly get an audio route stream via an ad hoc network that forms from different devices in the pervasive computing environment, such as a personal-shopper's phone, a nearby car's satellite navigator, and the mall's information kiosk. The pervasive computing environment helps Anne avoid browsing the website, increasing the likelihood of matching all her hardware and software capabilities to get the routing result she needs.

Open issues with current research on service composition (e.g., goal-oriented planning [5], open workflows [6] [7] [8], adaptable composition [9] [10] [11] and AI planning [12] [13] ) are that *i)* existing service composition models are inflexible; and *ii)* the composition models have limited support for handling composite service adaptation. Specifically, workflow-driven solutions [6] [7] [8] model user requirements as a complex task request in the form of workflows (i.e., abstract composite services). However, dynamic service availability is unpredictable. Such an exactly-defined task request removes the possibility of using services that may contribute to the user's request, but are not outlined in the predefined workflow. For example, with a workflow "get a shop location→ get ramp access points → get a baby-changing facility's location → get an audio route", Anne's request cannot be solved by the environment shown in Fig.2. Goal-driven solutions [13] [14] [15] [16] are more flexible, and model service composition problems by dynamically composing multiple services when an individual one for a service request is unavailable. However, existing goal-oriented approaches either handle only sequential service flows for such a composite service [15] [16], or employ AI planning algorithms that support various service flows but requires planning infrastructures [14]. In addition, existing approaches towards composite service adaptation and re-planning [12] [10] [11] [17] [18] rely on structured networks, central controllers or service repositories. Keeping such network topology or central controllers up to date in highly dynamic environments requires periodic interactions over error-prone channels to monitor whether a composite participant is absent.

This paper proposes a self-organizing, **Go**al-driven services **Co**mposition model in **Mo**bile and pervasive computing environments, called GoCoMo. In particular, it leverages a decentralized planning algorithm based on backward-chaining [19] to support flexible service querying. This algorithm supports complex service flows such as parallel service flows and hybrid service flows. Further, GoCoMo introduces an on-demand adaptation overlay that allows execution paths to dynamically adapt, which reduces failures. This model uses, with some extensions, an opportunistic resource allocation scheme [9] to lock a participant provider's resource for a composite only when this provider's service is about to execute.

This work extends our existing algorithm [20] with three contributions: First, the number of potential services available to the system may expand as providers enter the system's scope, automatically merging with the existing service-flows to resolve user requirements. Second, dynamic candidate execution path adaptation and selection, based on path reliability, are introduced to accomplish service execution with less failure. Third, a novel heuristic service request transition model is employed that prevents service requests flooding the network, which trades off traffic overhead with service discovery scope in service planning.

The evaluation compares this work with a distributed goal-driven service composition approach [13] [21]. Simulation results show the suitability of the proposed mechanism in a pervasive computing environment where providers are mobile and it is uncertain what services are available.

The reminder of this paper is organized as follows. Section 2 defines the problem and introduces the service

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



model that supports service composition. Section 3 describes the service composition algorithm. Section 4 presents the evaluation and result. Section 5 discusses the validity of this approach and remaining challenges in detail. Section 6 outlines related work. Section 7 summarizes this work and discusses the future work.

## 2 Service Model

The target network for this work is open and so may be unstructured or semi-structured with some autonomic service hosts. In other words, service hosts do not follow any authority in the network requiring them to schedule services for service requesters. Service hosts of interests will be cooperative and be prepared to process composite requests.

A service composite for complex user tasks can be modelled as a restrictive data transition in which the system data changes from initial data (i.e., a user's input parameters) to goal data (the requested output data), while satisfying all the requested functionalities or constraints. A participating service for the composite, packaging its resources (e.g., data, functionality), can support all or a part of (based on its resource provision's granularity) the data transition. This paper assumes services' functions and I/O parameters are semantically annotated using globally understood semantics and language, and able to match a service request with semantic matchmakers [20]. We assume such semantic service annotations are kept in local service hosts (devices) and can be advertised using probe messages. A service's invocation must be based on all the specified input data, and local devices can form an ad hoc network, cooperating with each other to resolve a complex user task.

A service is described as $S = \langle S^f, IN, OUT, QoS^{time} \rangle$, where $S^f$ represents the semantic description of service $S$'s functionality. $IN = \{\langle IN^S, IN^D \rangle\}$ and $OUT = \{\langle OUT^S, OUT^D \rangle\}$ describe the service's input and output parameters as well as their data types respectively. For this work, execution time $QoS^{time}$ is the most important quality of service (QoS) criterion as delay in composition and execution can cause failures [9]. A service composition model should select services with short execution time to reduce delay in execution.

Complex user tasks can be modeled as a service request $R = \langle R_{id}, \mathcal{I}, \mathcal{O}, \mathcal{F}, \mathcal{C} \rangle$, where $R_{id}$ is a unique id for a request. The set $\mathcal{F}$ represents all the functional requirements, which consists of a set of essential while unordered functions. The composition constraints set $\mathcal{C}$ are execution time constraints. A composition process fails if $\mathcal{C}$ expires and the client receives no result during service execution. A service composite request also includes a set of initial parameters (input) $\mathcal{I} = \{\langle \mathcal{I}^S, \mathcal{I}^D \rangle\}$ and a set of goal parameters (output) $\mathcal{O} = \{\langle \mathcal{O}^S, \mathcal{O}^D \rangle\}$.

Given the mobile nature of devices of interest, this paper discusses service provision in networks with high dynamism where the network topology is likely to change faster than it takes to entirely complete service composition or execution. A client is assumed to have limited resource and communication range, which are insufficient to aggregate enough service providers for a task. The solution proposed in this paper addresses mobility by providing a fully decentralized self-organizing/adapting composition model.

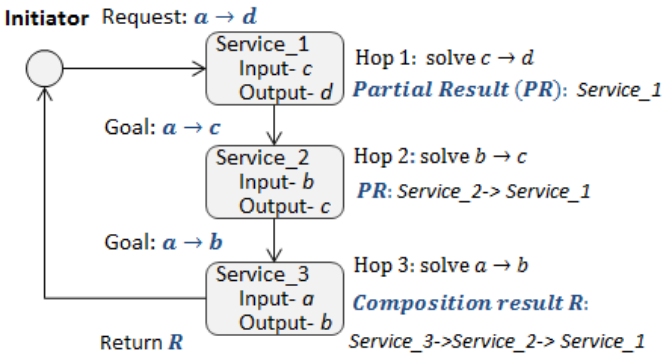


Fig. 3. General distributed backward-chaining model for service composition

## 3 Service Composition

A service composition process can be modelled as classic AI backward-chaining [12]. A backward-chaining process, also known as goal-driven reasoning [13], starts by searching for the knowledge that can infer a request's goals (consequences), and then the request is resolved backward from the goal to the request's antecedents, by converting the goal into subgoals, resolving back through these subgoals (e.g., Goal $a \rightarrow c$ in Fig.3). The process finds a solution when all the antecedents are reached. A general goal-driven reasoning process for service composition is shown in Fig.3. The initiator issues a service request $a \rightarrow d$ to start the process, which relies on distributed knowledge bases stored in local service hosts (planners). In each step of the service discovery, a part of the request's goal can be solved (e.g., $Goal : a \rightarrow d$ can be partially solved on $Service\_1$ that provides data transition of $c \rightarrow d$), and the remaining request ($Goal : a \rightarrow c$) is forwarded to the next hop service providers (i.e., Service_2). In such a process, it is the request's goal that determines which services will be selected and used. This process produces more flexible planning results than that of workflow-driven approaches, as it considers service discovery as an open-ended problem and dynamically generates composite services according to runtime service availability. For example, in Anne's scenario, she can specify her requirements as a goal: *an audio step-free route to a store and then to a baby-changing facility* which will be resolved hop-by-hop and eventually supported by a composite service. As illustrated in Fig.2, the planning resolves first to AudioNavigator, back to RampAccessChecker, then to StoreRouter as well as FacilityRouter, to StoreLocator, to IndoorMap, and finally to StoreQuery.

Modelling service composition as such a process has been explored in infrastructure-rich networks where composition planning is based on infrastructures like repositories [22] or pre-existing overlay networks [7] . However, this kind of infrastructure, as mentioned in Section 1, is not suitable for our target environment, and neither are the existing goal-driven reasoning processes. To allow mobile pervasive computing environments to benefit from the flexibility that such a goal-driven service composition brings, this paper

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



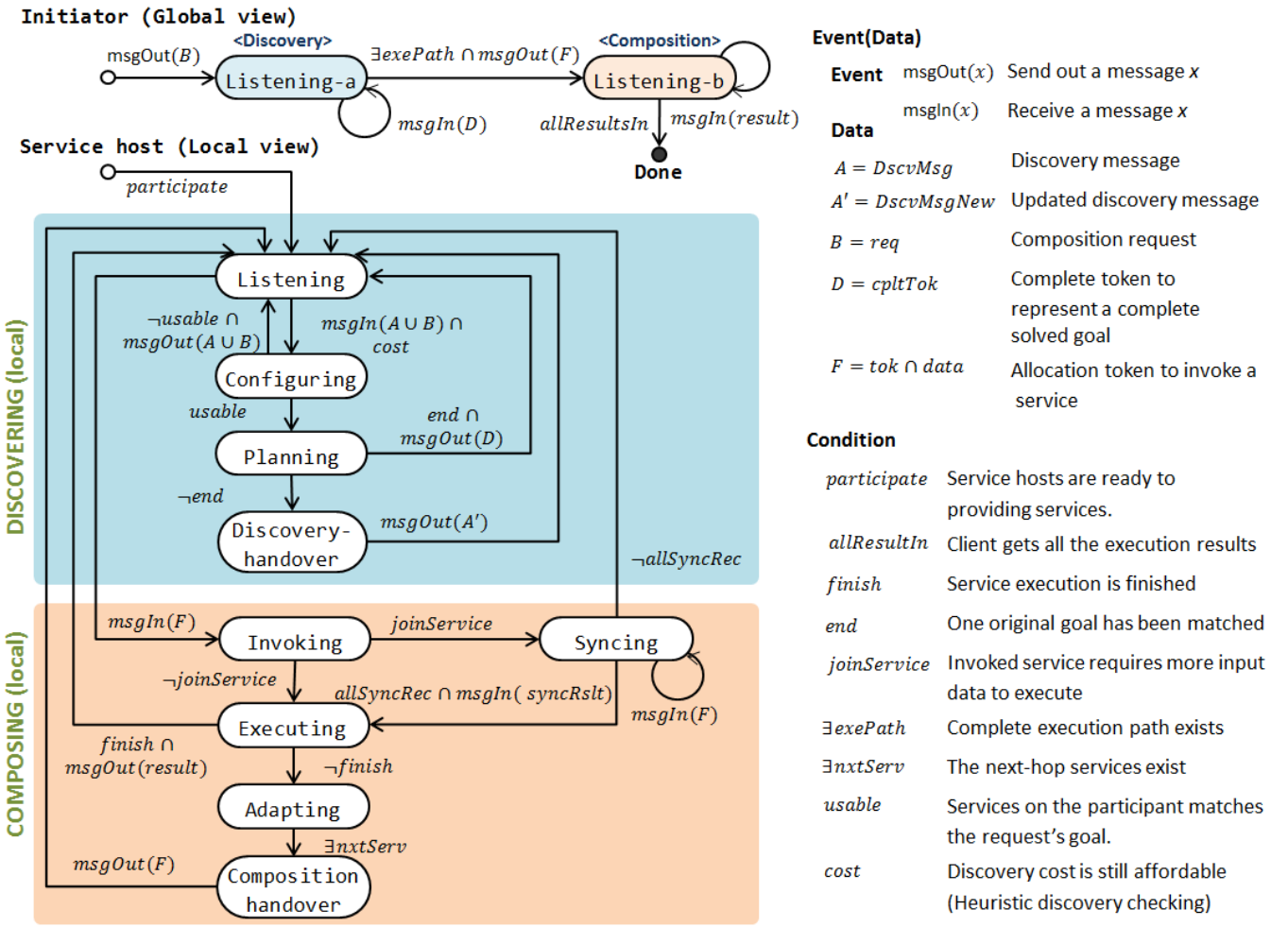


Fig. 4. Decentralized service composition model

proposes a service composition model that handles the following issues:

- **Goal-Driven Service Discovery**- is handled via a composite participant cooperation mechanism to coordinate distributed knowledge bases and independent planners to support the generation and the maintenance of various service flows. (Section 3.1)
- **Opportunistic Service Execution**- is handled via online adaptable reasoning to create awareness of and compose potentially better services that may appear during service execution. (Section 3.2)
- **Heuristic Discovery Checking**- is handled via a distributed heuristic discovery mechanism based on QoS attributes to increase the likelihood of time-efficient services being selected during execution and prevent composite requests flooding the network. (Section 3.3)

The model captures a goal-driven reasoning algorithm to support a fully decentralized service composition process, which is modelled as a state-transition diagram in Fig.4. A transition between these illustrated states is triggered by message communication events (e.g., msgIn, msgOut) or local conditions (e.g., participate, usable). The model includes i) a goal-driven service discovery protocol to discover and link all the reachable and usable services, and ii) an opportunistic service composition protocol that is based on the discovery result, to select, compose and execute services hop-by-hop on demand.

In particular, the *global* service discovery ($listening - a$ state) starts when a client sends out a composite request to look for composite participants. Composite participants in this model are service hosts who are capable of reacting to and reasoning about a composite request. They are also responsible for invoking their subsequent services during execution. From a composite participant's perspective, the *local* discovering loop starts in the *listening* state and ends when the composite participant is invoked (the invoking state), while a *local* composing process, starts in the *invoking* state and ends in a *composition-handover* state.

### 3.1 Goal-Driven Service Discovery

Global service discovery is a process to group composite participants. Each composite request can be resolved partially (or completely), and the remaining request is forwarded to the composite participant's neighbours to continue the discovering process. In this composition protocol, any remaining request is enclosed in a discovery message that is forwarded between composite participants.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



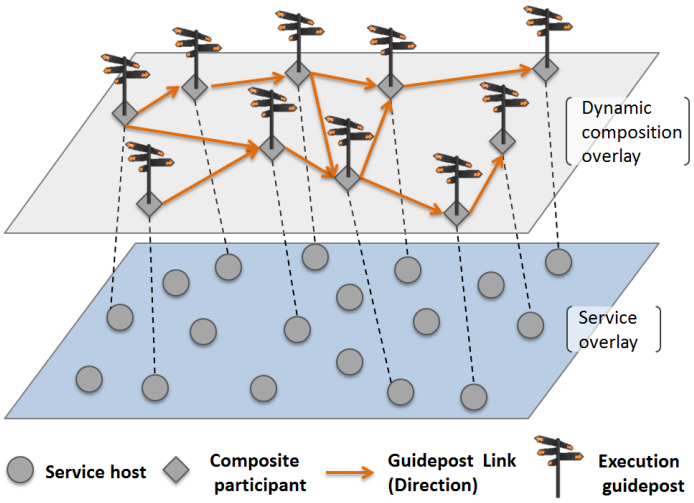


Fig. 5. Dynamic composition overlay

**Definition 1.** *A **discovery message** including a request's remaining part $R'$, is represented as $DscvMsg = \langle R', cache, h\rangle$, where cache stores the progress of resolving split-join controls for parallel service flows (see Section 3.1.2), and $h$ is a criterion value for request forwarding and service allocation (see Section 3.2 and 3.3).*

The global service discovery process establishes a temporary overlay network called a dynamic composition overlay (Fig.5), which contains all the reached usable composite participants. Such a network only lasts for the duration of a composition. It is managed by a set of *execution guideposts*, which are control elements for the discovered service flow. As shown in Fig.5, each guidepost is maintained by a composite participant, linking the corresponding service to who sent the discovery message. An execution guidepost is adaptable and a byproduct of the composite participant's local discovery process.

**Definition 2.** *An **execution guidepost** $G = \langle R_{id}, \mathcal{D}\rangle$ includes a set of execution directions $\mathcal{D}$ and the id of its corresponding composite request. For each $d_j \in \mathcal{D}$, $d_j = \langle d_j^{id}, \mathcal{S}^{post}, \omega, \mathcal{Q}\rangle$, where $d_j^{id}$ is a unique id for $d_j$, and the set $\mathcal{S}^{post}$ stores the participant's post-condition services that can be chosen for next-hop execution. The set $\omega$ represents possible waypoints on the direction to indicate execution branches' join-nodes when the participant is engaged in parallel data flows. The set $\mathcal{Q}$ reflects the execution path's reliability of this direction, e.g., the estimated execution path strength and the execution time (Section 3.2).*

The local discovering process for a composite participant is described in Algorithm 1. Note that the state transiting conditions like $\neg end$, *cost* and *usable* are not shown in this algorithm since they have been illustrated in Fig.4. This process reacts when a composite request or a discovery message is received, and generates an execution guidepost as well as new discovery messages that enclose the part of the composite request which cannot be solved on this composite participant.

**Data** : Sender $Y$ sends $DscvMsg$. Receiver $X$ hosts $S = \langle S^f, IN, OUT, QoS^{time}\rangle$.
**Result**: an execution guidepost, A $DscvMsg$ containing the remaining request R

```
1  /* Listening */ ;
2  while receive message DscvMsg do
3      /* Configuring */;
4      GoalMatch(S, R);
5      /* Planning (when usable)*/;
6      if ∄DscvMsg^log then
7          New D;
8          DscvMsg^log ← DscvMsg;
9          if partUsable then
10             Event← addJoin
11         else
12             Event← add
13         end
14     else
15         if DscvMsg^log and DscvMsg have matched
           cache value then
16             Update cache; Event← addSplit
17         end
18         if Progress(DscvMsg^log) < Progress(DscvMsg)
           then
19             Event← adapt;
20         end
21         if Progress(DscvMsg^log) ==
           Progress(DscvMsg) then
22             Event← add ;
23         end
24     end
25     switch Event do
26         case addSplit:
27             foreach d_i ∈ D do S^post ← S^post + Y
28         endsw
29         case adapt:
30             Clean D; d_y ← ⟨R_id, Y⟩;
31         endsw
32         case add||addJoin: d_y ← ⟨R_id, Y⟩;
33     endsw
34     //When a branch's resolving is finished
35     Initiate cpltMsg' = ⟨R', cache, h⟩;
36     Send cpltcvMsg';
37     /* Handover */;
38     if (Event!=add)&&(RemainReq) then
39         Initiate DscvMsg' = ⟨R', cache', h'⟩,
           R' = ⟨I', O', F', C'⟩;
40         O' ← IN;
41         F' ← F − S^f_matched;
42         C' ← C − QoS^time;
43         Calculate h;
44         if GoalMatch(S, R) = partial then
45             Initiate cache_i = ⟨S_id, m, c⟩;
46             S_id ← P^rec, m ← OUT ∩ O;
47             c ← ⟨ num(m)/ num(O)⟩;
48             cache' ← cache + cache_i;
49         end
50         Send DscvMsg';
51     end
52 end
```

**Algorithm 1:** Local Service Discovering Algorithm

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



### 3.1.1 Local Discovering and Discovery Messages

The initial goal of a composite request is to produce the final, required output, and if a service matches the goal, the service is usable for this composite request. A service $S = \langle S^f, IN, OUT, QoS^{time} \rangle$ matching a composite request $R$'s goals ($R = \langle R_{id}, \mathcal{I}, \mathcal{O}, \mathcal{F}, \mathcal{C} \rangle$) is ranked from different matching levels:

$$GoalMatch(S, R) = \begin{cases} usable & \text{if } OUT \supseteq \mathcal{O} \\ partUsable & \text{if } OUT \subsetneq \mathcal{O}, OUT \neq \emptyset \\ unusable & \text{otherwise} \end{cases} \quad (1)$$

In the configuring state, a composite participant checks the goal matching level for a composite request. To prevent repeatedly checking for the same composite request, composite participants maintain a log for composite requests. During service planning (Line 5-33), a matched participant can create a direction to link to the request sender, which will be added into an execution guidepost.

In the discovery handover state, a discovery message can be created that depends on its matching level for the in progress composite request (Line 39-49). A composite participant updates the composition request, by removing the goal, adding its required input parameters as a new goal, removing the matched function and changing the execution time ($QoS^{time}$) requirement in the request. Then, the updated request is enclosed in a discovery message.

A $cache$ is used in discovery messages that stores information about a node $Y$ which sends a request $R$ to the composite participant $S$, in which the goal is partially matched (Line 44), i.e., when $GoalMatch(S, R) = partUsable$. A composite participant caches a request sender when the participant only partially match the request's goal. Such a request sender's information is represented as $C_i \in cache$ ($C_i = \langle S_{id}, G^{matched}, \rho \rangle$), where $S_{id}$ is the unique id of the requester node (e.g., the node $Y$), and the set $G^{matched}$ stores matched outputs. The parameter $\rho$ ($\rho \in (0, 1)$) captures the progress of addressing the partially matched goal. For example, in Anne's scenario, an IndoorMap provider (see Fig.1) receives a discovery message from a StoreRouter, which includes a subgoal for Anne to ask for two input parameters: *local map* and *store address*. As the IndoorMap service host can only provide the *local map*, the StoreRouter's goal is half matched, so the IndoorMap caches the request sender by adding $C = \langle IndoorMap,\ localmap,\ 0.5 \rangle$ into the $cache$ set. A discovery message will be forwarded to the composite participant's neighbours. As a discovery message is sent out, the model may continue with the discovery of the remaining services before the current local discovering cycle has collected all the matched directions.

### 3.1.2 Data-Parallel Tasks and Execution Guideposts

A user requirement may include data-parallel tasks, such as multi-source data aggregation, which are modelled as parallel workflows in the service composition model [6] [9]. Goal-driven service discovery should resolve a data-parallel task without a-priori knowledge of its inner data transaction. In the planning state, a composite participant recognises split-join control logic for such a task and generates a corresponding direction for it:

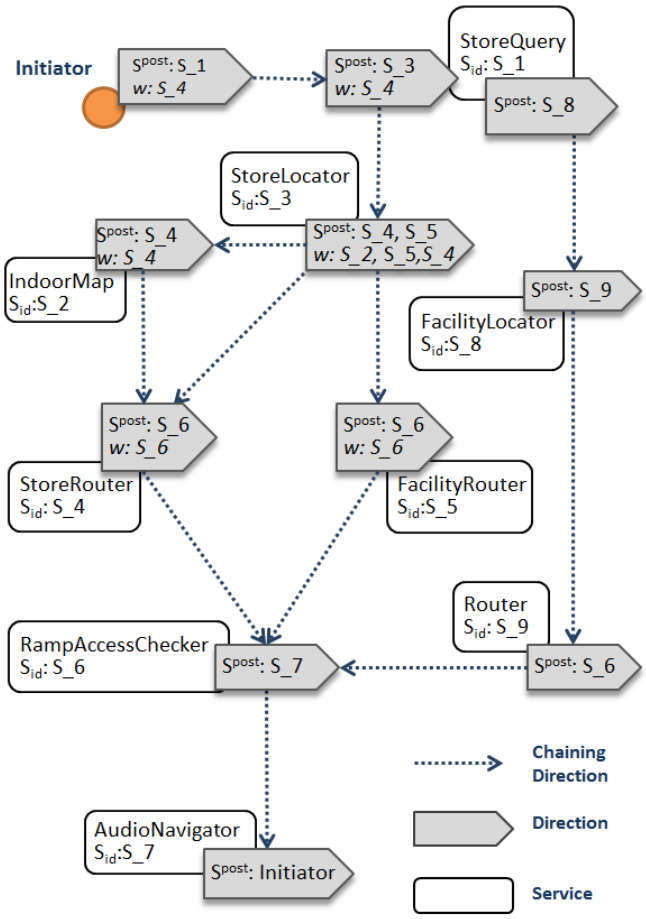


Fig. 6. An example for dynamic composition overlay networks

**Definition 3.** *An **and-splitting direction** directly links to multiple services, which requires the composite participant to simultaneously invoke these services for execution. An **and-joining direction** links to a waypoint-service (join-node) that collects data from the composite participant and other services on different branches.*

A composite participant detects various events (Line 6-23 in Algorithm 1) as triggers for building these directions. Establishing a set of splitting directions is triggered by the *addSplit* event, and a joining direction can be created when an *addJoion* event occurs. An example for a dynamic composition overlay is illustrated in Fig.6. A splitting direction on the service StoreLocator links to the FacilityRouter as well as the StoreRouter, while a joining direction to this service indicates that a data syncing process will be needed on its subsequent execution.

An execution guidepost has a life cycle with four phases: preparing, verifying, directing and waiting. It spans the duration of a participant's local discovering and local composing processes. When a composite participant $i$ establishes an execution guidepost, a lifetime $T$ is assigned for this execution guidepost. The $T$ is determined according to the remaining time constraints ($\mathcal{C}$) and a heuristic value (Section 3.3). Before the participant gets invoked for execution, the guidepost prepares for composition by collecting the directions. When the participant is about to compose services,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



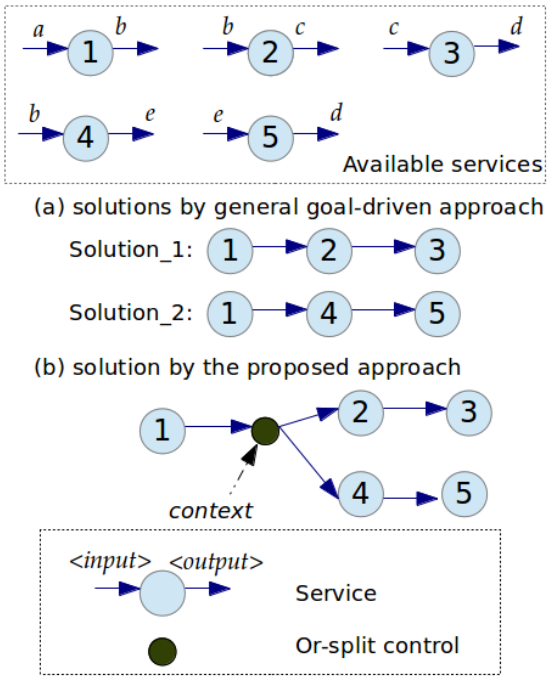


Fig. 7. An example for composition solutions

the directions in the guidepost are verified by sending a probe message to check the service availability associated with the directions. Afterwards, a valid direction will be selected. Subsequently, if its lifetime $T$ is still not expired, the guidepost will live for a while in case the remaining execution needs a rollback for failure recovery (see Section 3.2). If $T$ has expired, the participant drops the guidepost for the composition request, removing itself from consideration in the composition. The overall composition process continue to work with the dynamic composition overlay (see Fig.5).

### 3.2 Opportunistic Service Composition

Execution guideposts support local discovery results. In a global view, as shown in Fig.7(b) each execution guidepost that maintains multiple directions acts as a split-choice[1] control element. This split-choice guidepost allows the system to select the best path according to the current system context, like node availability or the reliability of the remaining path. A re-selection mechanism is also proposed to recover the system from path failures. Such a solution is partially found beforehand (in the global discovery stage), and can be expanded during service execution if newly matched services join the network.

The service composition process extends our opportunistic service execution mechanism [9] [24] that binds services on demand and releases them after execution. This allows a composite participant's computing resources to be locked only for the duration of its local composing process. This may reduce the time a composite participant is occupied, which in turn increases its overall service availability. A local service composition process is a transition from the invoking state to the composition handover state, as shown in Fig.4. Directions with the best quality value $\mathcal{Q}$ (Definition 3) will be chosen for the next-hop invocation. The $\mathcal{Q}$ ($\mathcal{Q} > 0$) on participant $n$ is calculated based on service execution time $\sum QoS_n^{time}$ on each service and the remaining execution path's reliability.

$$\mathcal{Q} = \alpha * \sum QoS_n^{time} + \beta h_n \tag{2}$$

where $\alpha$ is a weight value determined by the local network's dynamism, and $h_n$ is a heuristic value that reflects the remaining path's length. The $\beta$ derives from a path duration estimation scheme [25] to estimate execution paths' reliability in mobile ad hoc networks.

$$\beta = \lambda_0 v / \mathcal{R} \tag{3}$$

where $\lambda_0$ is a proportion constant defined by network factors like node density, $v$ is the nodes' average speed, and $\mathcal{R}$ represents the transmission range [25]. A direction that can route to a reliable execution path with a quick execution time can have a low $\mathcal{Q}$ value. As a direction for services executed in parallel may have waypoints, to synchronize a parallel service flow, the join-node will be selected when a parallel flow starts to execute.

A composite participant checks the availability of the first service on a direction before it hands over the composing process to the service. When such checks result in "unavailable", the composition model applies a *backjumping* mechanism to prevent failures. During composition, a composite participant will get the id of the closest service that has multiple available directions from a service allocation token (see data $F$ in Fig.4.). If a composite participant cannot find a service available for composition handover, the composition will back-jump to the one that has multiple available directions, as long as it still locks resources for the composition (i.e., its guidepost is in the waiting phase), so that another potential remaining execution path can be picked out for execution. For example (Fig.6), if the $Router$ detects its link to the $RampAccessChecker$ is unavailable when it is handling service execution, the $Router$ will allow the service execution process backjumps to the $StoreQuery$ who has a backup direction that is able to invoke another execution path. An execution can ultimately fail if no execution path is available.

### 3.3 Heuristic Discovery Checking

The service discovery model finds services hop-by-hop based on service data dependency. During service discovery, service hosts relay a composite request when they cannot match the request. Given the large number of possible data dependency relations and possible relay nodes in a network, a discovery mechanism is required. This is to find enough usable services in a reasonable period of time[2] by employing a practical number of composite participants.

We use a function $r(n)$ to calculate the relaying cost (relaying hops) from the last composite participant to current composite participant $n$ or a relay node $i$ (for $r(i)$). These

1. A split-choice control element can have more than one outbound paths, and only one of them is selected for invocation [23].

2. A tolerable waiting time for a simple information query is about 2 second [26]. For operating tasks, the waiting time should be within 15 seconds [27].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



calculations are based on the heuristic value $h$ in a discovery message.

$$h_n = \sum r(n) + n \quad (4)$$

The discovery cost is defined as $d(n) = \mu(h_{n-1} + r(i))$, where $\mu$ is a local communication channel parameter that is defined by local composite participants and relay nodes. When a client issues a composite request, it sets up a time $T_{discovery}$ for global service discovery. When the estimated remaining discovery time $T'_{discovery}$ is above a threshold $\tau$: $T'_{discovery} = T_{discovery} - d(n) > \tau$, the node/participant will stop relaying/processing the composition request. The threshold is determined by the remaining time constraint value on participant $n$, represented by $\tau = l\mathcal{C}_n$, where $l$ is a weight value that determines a level for heuristic discovery checking, and $\mathcal{C}_n$ is the execution time constraints. This heuristic discovery check trades off the service discovery scale and the service user's QoS requirements (in particular, response time). It prevents the system from over-expanding the discovery scope and unnecessary communications, but supports discovering sufficient composition results to support the user requirement. The selection of the $l$ value will be discussed in Section 4.2.

TABLE 1
Simulation Configuration

| | |
|---|---|
| **Controlled** -(a) Section 4.1 and 4.2 | |
| Number of providers | 20 (sparse), 30 (medium-dense), 40 (dense), 50 |
| Types of services | 10 |
| Service instance per request | 5, 10 |
| Mobility (node speed) | slow: 0-2 ($m/s$), fast: 8-13 ($m/s$), medium-fast: 2-8 ($m/s$) |
| **Controlled** -(b) Section 4.3 | |
| Number of providers | 20, 30, 40, 50 |
| Types of service flows | sequential, parallel, hybrid |
| Types of services | 5-15 |
| Service instance per request | 5 |
| Mobility (node speed) | medium-fast: 2-8 ($m/s$) |
| **General** | |
| Simulator | NS-3 |
| Clients | 1 |
| Communication range | 250 ($m$) |
| Field | 1000*1000 ($m^2$) |
| Service deployment | 1 service per node |
| Semantic matchmaking delay | 0.2 ($s$) [28] |
| Service routing | dynamic AODV 10-hops |
| Sample | 300 runs |
| **Random** | |
| Node placement | |
| Service execution time | 0.01-0.1 ($s$) |
| Node movement | random walking mobility model |

## 4 Evaluation

The dynamic composition overlay, the related discovery algorithms and the composition logic were implemented on the NS-3 simulator, and evaluated focusing on the following three metrics:

- Planning failure rate (Section 4.1): the algorithm's success at finding sequential solutions to a goal request. The service composition/planning failure rate is calculated as the ratio ($\in [0, 1]$) of the number of failed planning processes to the number of all the issued requests during the simulation cycles. The duration for a client to receive the first pre-execution plan and the sent messages (system traffic) during this process were also measured for performance analysis.
- Execution failure rate (Section 4.2): the algorithm's success at handling potential failures that may appear during service execution. In this experiment, the execution failure rate is computed as the ratio ($\in [0, 1]$) of the number of failed execution to the number of all the successful planning, considering different system configurations (i.e., mobility, and network size). This experiment also included measurements for execution performance, such as response time for a client to receive the execution result and the system traffic during this process.
- The failure rate for composing parallel service flows (Section 4.3): the algorithm's success at finding and invoking parallel solutions. Composition failure rates were calculated under different configuration of service availability.

For the purposes of establishing a good baseline against which to compare GoCoMo, we have combined state-of-the-art functionality on a decentralised cooperative discovery model [13] with a continuing message passing model [21] that enables decentralized service invocation. This baseline approach is referred to as CoopC in the following sections. CoopC's cooperative discovery employs a backward goal-driven service query and forward service construction mechanism. It generates a sequential service flow as the discovery result. However, CoopC's offline approach to service planning means that it does not support runtime composite service adaptation. Unlike the cooperative discovery model [13] used as an input to CoopC [29], we have implemented CoopC to start service execution when the first pre-execution plan is found. The plan is passed through the service execution path to indicate which service will be invoked for subsequent execution. This makes CoopC and GoCoMo more comparable when measuring response time.

### 4.1 Flexibility of Service Planning

The evaluation scenario for assessing the flexibility of the proposed discovery strategy contains one client node and a specific number of service provider nodes. The context configurations of the scenario is shown in Table 1.(a). There are 10 different atomic services and their duplicates in the scenario, one per service provider. These services have the potential to form a 5-service sequential flow or a 10-service sequential flow to support service queries with varying complexity. GoCoMo and CoopC were simulated with configurations that differ in their service density, node mobility and the complexity of service composition.

The failure rate and the performance are illustrated in Fig.8 - Fig.10. On finding pre-execution plans, GoCoMo shows a higher possibility of returning a pre-execution plan than CoopC (Fig.8(a) and (b)). In particular, with the increasing complexity of service composition (Fig.8(c)), GoCoMo raised about $0 - 6\%$ failures while CoopC raised approximately $0 - 8\%$ failures in most of the scenarios. The

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



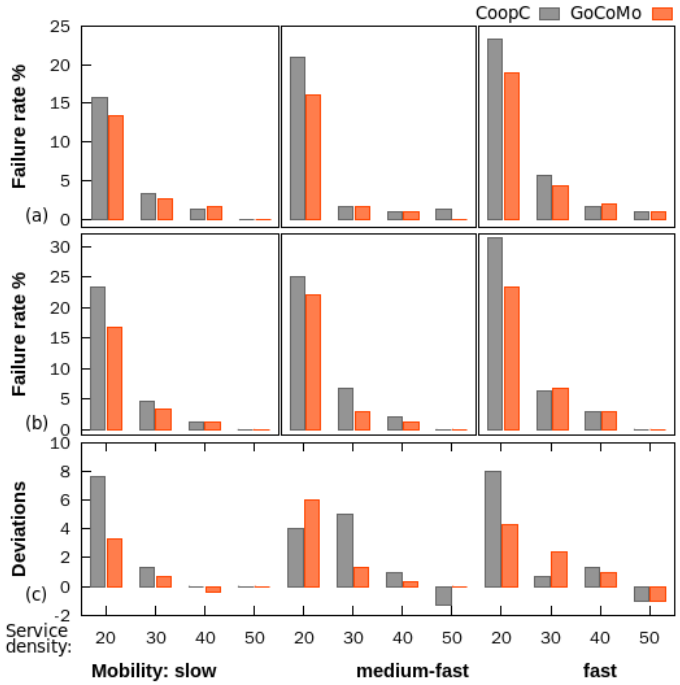


Fig. 8. Planning failure rate in mobile networks: (a) 5 service instance per request, (b) 10 service instance per request, and (c) the failure rate deviation between (a) and (b)

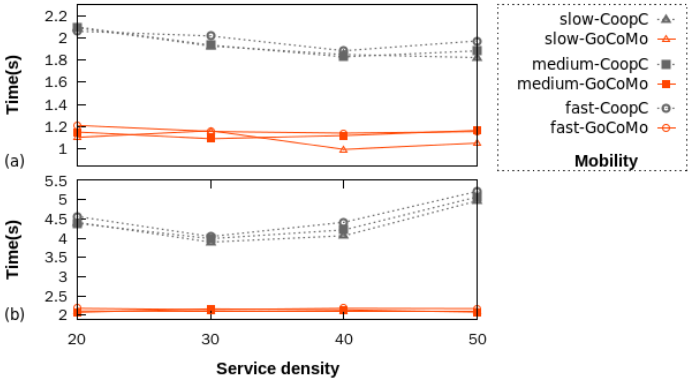


Fig. 9. Discovery time in mobile networks: (a) 5 service instance per request, (b) 10 service instance per request

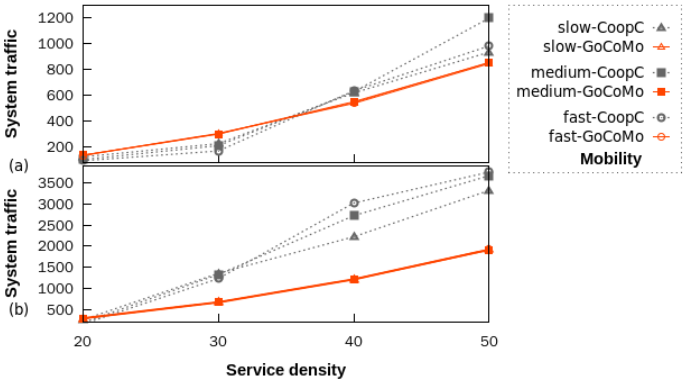


Fig. 10. Discovery traffic in mobile networks: (a) 5 service instance per request, (b) 10 service instance per request

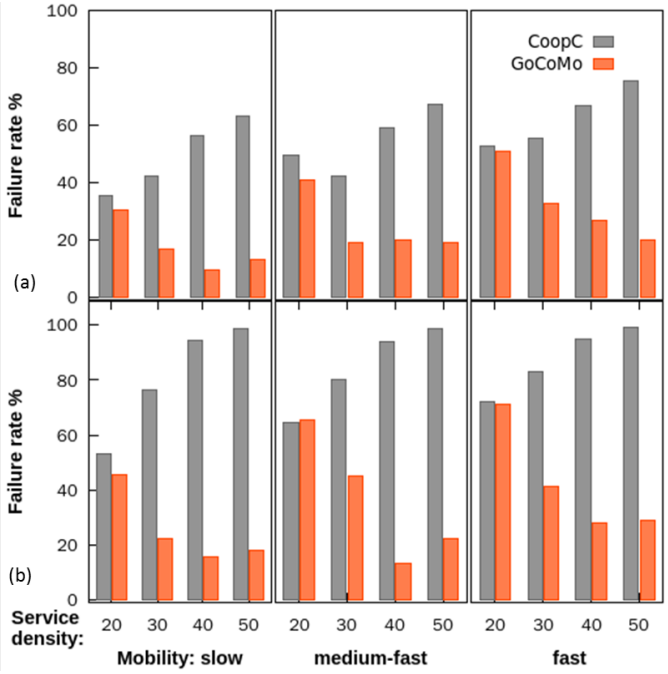


Fig. 11. Execution failure rate in mobile networks: (a) 5 service instance per request, (b) 10 service instance per request

results also show that GoCoMo discovery spent less time than CoopC to return the first pre-execution plan (Fig.9), but it sent slightly more messages than CoopC's discovery model (Fig.10(b)) to resolve a simple request in the sparse scenario (20 nodes) and the medium-dense scenario (30 nodes). GoCoMo discovers more quickly, since it is not like its counterpart that requires one more step to finish the service discovery process, which constructs a pre-execution composite by forwarding a construction message to all the participant service providers after the backward service query. GoCoMo allows the fragments of execution plans to be selected and cooperate at execution time. GoCoMo produces slightly more traffic for simple requests (5 service instance per request) in sparse and medium-dense scenarios (Fig.10(a)) as it discovers more service links to find various possible execution paths for a single pre-execution plan (see Fig.7(b)).

### 4.2 Adaptability of Composite Services

A composite solution is adaptable if the system is able to compose solutions and complete service execution even in a mobile environment. The execution failure rate was calculated to show such adaptability for GoCoMo and CoopC. In this simulation, both of the approaches are implemented such that service execution starts immediately when the first pre-execution plan is returned to the client.

For execution failure rate (Fig.11) GoCoMo is more successful compared to CoopC in sparse networks for most scenarios (20 nodes) and also in dense networks (30-50 nodes). CoopC produced heavy system traffic during service execution (see Fig.13) when service density increases. This is because when the first returned plan is applied for execution, CoopC may still have participants that are performing service discovery (mainly in the forwarding

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



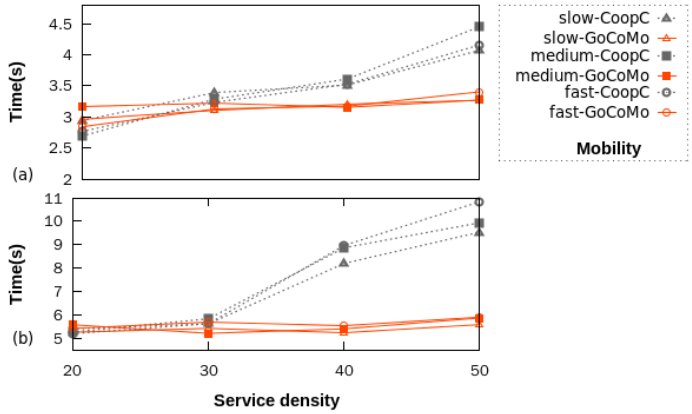


Fig. 12. Response time: (a) 5 service instance per request, (b) 10 service instance per request

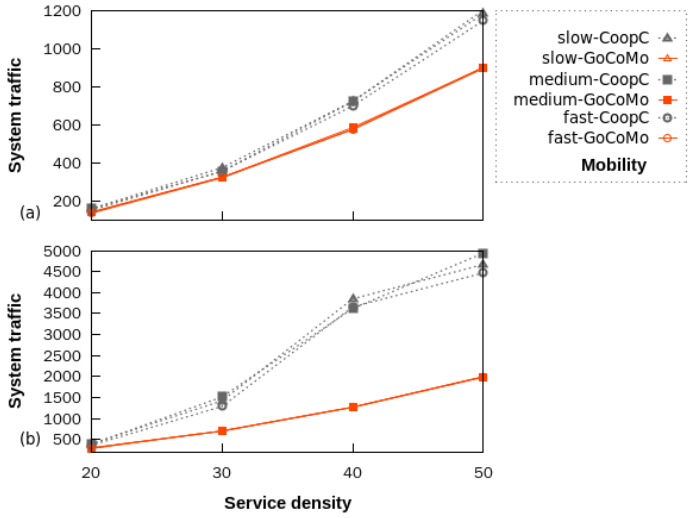


Fig. 13. Overall traffic: (a) 5 service instance per request, (b) 10 service instance per request

process for service construction). Such system traffic occurs at anytime less than $t$, for $t \in [0.55, 5.3]s$, which indicates a frequent interaction between composition participants[3]. Frequent interactions in a network increases the possibility of high packet collision failures [30] [31]. Therefore, CoopC had more failures even though service density is increased. Although GoCoMo had the same tendency when service density increases from 40 to 50, in general, GoCoMo produced less failures than CoopC in dense networks. Starting service execution after the service discovery process leads to all participants reducing some interactions in the service execution process, which may prevent such frequent interactions, but it delays the service composition process, which may cause even more fails because of service path loss in a mobile environment [9] [32]. With a high service density (e.g., above 40 services), GoCoMo in a fast mobility network returns about $0 - 17.33\%$ more fails than in a medium-fast mobility network or a slow mobility network. In a low service density network (e.g., a network with 20 services),

3. Service execution time interval $[0.55, 5.3]s$ is calculated by removing the time spent on discovery (in Fig.9) from the response time (in Fig.12).

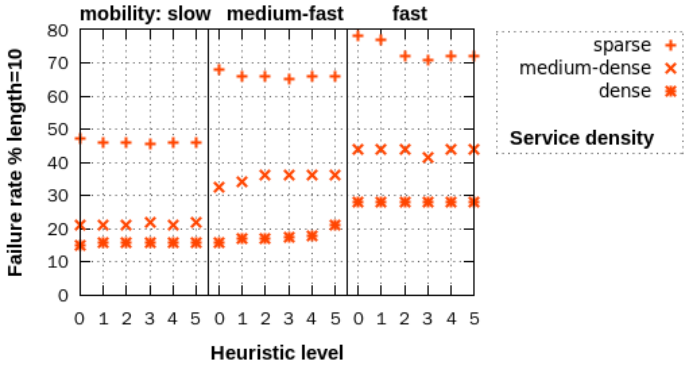


Fig. 14. Heuristic levels affect failure rates

the failures increase to $4.67 - 25.33\%$. This is because low service density networks only have a limited number of services in a node's communication range, which makes it hard for GoCoMo to find alternative service execution paths to replace failed paths.

For service execution performance, the response time was measured and the system traffic for a composition process was counted. The results (Fig.12 and Fig.13) show that GoCoMo processes more quickly than CoopC approach in high density scenarios and is less affected by service density. In highly dynamic (fast) networks, the time spent on service composition for GoCoMo increases slightly faster than that in slow networks. This is because execution failure recovery requires jumping back to an executed node. Fig.13 shows that GoCoMo generates less traffic compared to CoopC, because CoopC merges all the partial plans during service discovery, which increases interactions among participants.

GoCoMo applied a heuristic service discovery mechanism. The selection of the threshold level $l$ may affect the results for returning a composition solution. Because a high threshold level can lead to more discovery messages that may increase the possibility of packet collisions and packet loss [30], and in turn composition failures. A test was run to measure this influence, which assumed all the nodes in the network use the same threshold level for simplicity. The test applied 6 levels of heuristic discovery from 0 to 5. Level 5 means there is no heuristic check, and the level 0 process used the smallest threshold value, which represents a comparatively small search scope. Fig.14 shows that, in a medium-dense network (30 services) with medium-fast mobility, a low level (i.e., Level 0) of heuristic discovery will reduce composition failures. Because, in such a dense network, even a small search scope can support enough backup execution paths for failure recovery, while it sends less query messages than its high level counterparts, reducing the potential for packet collision failures.

### 4.3 Planning Complex Service Flows

The evaluation scenario for assessing the support for complex service flows contains one client node and 5-15 different atomic services and their duplicates, one per service provider. These atomic services vary in the type and the number of their input and output parameters. This reflects that when service instances engage in a workflow, each may

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



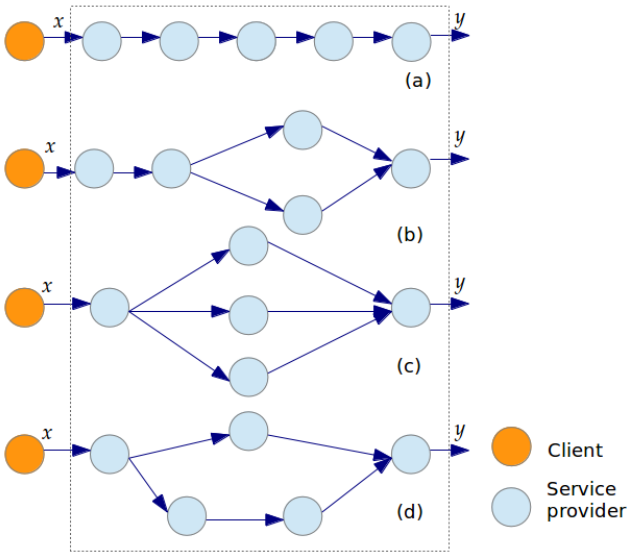


Fig. 15. Potential service flow for a composite service that supports data transition: $x \rightarrow y$

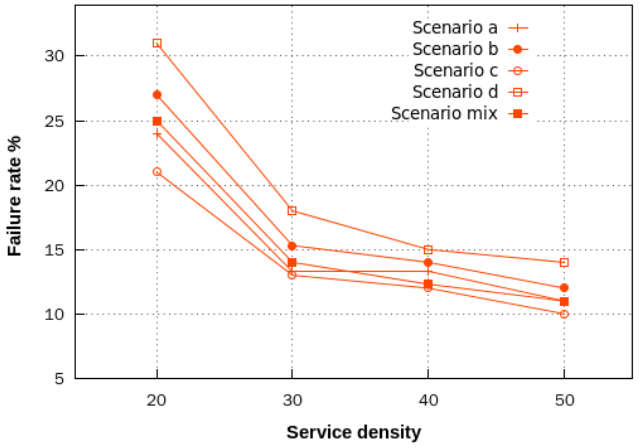


Fig. 16. Failure rate for the data transition ($x \rightarrow y$) request

have multiple, differing in-degrees and out-degrees[4]. The potential service flows constructed by these atomic services for a composite service is illustrated in Fig.15. These service flows connect participating services relying on their data dependency, which include sequential workflow models (model a), parallel workflow models (model c and d), and hybrid workflows (model b) [33] [23]. Each of them includes 5 service instances. All the service flows start and end with the client node that issues a service query $x \rightarrow y$.

The failure rate of GoCoMo on scenarios that contain the different kinds of flows are shown in Fig.16. It was simulated under medium-fast networks. On returning solutions, GoCoMo supports model c services (Fig.15(c)) the best, and has more failures in the scenario with model d services (Fig.15(d)). CoopC supports only model a (sequential Fig.15(a)) services (see Section 4.1), and so only GoCoMo results are illustrated.

4. In-degree and out-degree indicate the number of connections entering and leaving a workflow node, respectively [33].

## 5 Discussion

GoCoMo is designed for mobile and pervasive computing applications that require multiple participants, and run in dynamic ad hoc environments. GoCoMo generally reduces the failure rate of service composition in dynamic, open pervasive computing environments. Specifically, flexible compositions of services are possible by using the proposed goal-driven composition planning model, and the impact of changes in the operating environments can be reduced by GoCoMo's adaptation and execution mechanism with a reasonable cost. The context of this work refers to user requirements and the runtime environment (e.g., service availability). The model allows processes for pervasive computing, like information querying, data aggregation, and in-network data processing to be planned and executed in dynamic environments. It also raises the potential for applications like the messaging-based concierge service [34] [35] to be applied in pervasive computing environments. On the other hand, as service flows that include iterative logics have not been addressed, this model does not suit applications that imply complicated transactions, such as e-commerce.

Service composition systems are likely to be provided and maintained by third-parties, and in general, hop-by-hop processing may expose the overall control and dataflow to assigned providers. Service composition should be processed in reliable entities and able to protect a composition requester's privacy (e.g., data flow). Although this work assumes a trustworthy environment, the backward composition model described here goes some way towards addressing the privacy issue by partitioning the dataflow and the composition requester's goal. In particular, a service provider cannot see the whole data flow and the requester's data is not visible to the full service-flow's provider (i.e., the ending services' provider).

Service discovery process matches services to a composition request. This work assumed no matchmaking infrastructure for service discovery, which implies the use of dynamic matchmaking. To cope with heterogeneous services, a dynamic semantic matchmaker is needed. However, dynamic semantic matchmaking can be resource-consuming and reduce the overall performance of service composition [36]. A semantic matchmaking model was previously outlined for dynamically maintaining a distributed semantic service dependency overlay network, which goes some way towards efficient semantic matchmaking [20], but for resource-constrained devices, the cost of maintaining a semantic overlay network, may make participation in such an open, sharing model infeasible. A mechanism that can cope with fast and lightweight semantic matchmaking is still requested.

## 6 Related Work

Service composition has emerged as a promising solution to service-rich environments such as those predominant in pervasive computing. To reduce composition and execution failures while dealing with complex user requirements, existing service composition techniques investigate flexible composition planning mechanisms, and service execution

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2533348, IEEE Transactions on Services Computing



policies. Examples of such investigations applied to pervasive computing include open service discovery approaches, and dynamic service planning approaches, both of which automatically discover a combination of multiple services to support a user goal or a task when a single matched service is not available. Efforts on dynamic binding for decentralized service execution are also included.

### 6.1 Open Service Discovery

Service discovery allows a system to aggregate information that is scattered in local service providers, to resolve composite requests. Open service discovery approaches are flexible compared to those that rely on a predefined workflow to find services that can exactly match the sub-tasks in the workflow.

A graph-based service aggregation method [8] models services and their I/O parameters in an aggregation graph based on the parameter dependence of the services. Complex user requirements in this method are resolved according to predefined abstract composites. It dynamically composes services to support a task in a workflow when a direct match between the task and a single service does not exist. A composite service is found if the aggregation graph contains a path to link the task's output parameter and its input parameter. Similarly, a dependency graph [37] was used for service aggregation, but its usage differs from that in [8] since it directly maintains service dependency relations rather than their I/O parameters relations. An open workflow [6] has been proposed to support service composition in mobile ad hoc networks. It models workflows that already exist in the environment as a supergraph, and discovers services through the data flow in the supergraph. However, the above approaches require central entities for the graph-based service directory, which implies frequent network communications to maintain such a directory when the network topology changes quickly. In addition, they require a pre-existing workflow to discover services that support complex data flows, such as one with parallel logic. Such a workflow may need to be generated offline by a domain expert or a composition planning engine, which is inconvenient when a change is required at runtime. A decentralized reasoning system [38], which does not need a pre-existing workflow, composes services using a distributed overlay network built over P2P networks and enables self-organizing service composition through management of the network. However, this approach assumed that all the participants know their geographic locations, and the service request is sent to the participants' physical neighbours. In mobile environments where devices' geographic locations can change quickly, frequently updating geographic locations wastes computation resources.

### 6.2 Dynamic Composition Planning

Dynamic composition planning resolves user requirements and generates service flows during service composition. Classic AI-planning algorithms, such as forward-chaining and backward-chaining, have been applied for dynamic composition planning. WSPR [14] proposes a novel AI planning-based algorithm for large-scale Web services. This work is based on the analysis of complex networks. The EQSQL-based planning algorithm [39] applies rank-based models to promote service composition efficiency. Ukey et.al.[15] model a Web service as a conversion from an input state to an output state and maintained published services as a dependency graph. They employed a bi-direction planning algorithm to find a path with the smallest cost from the dependency graph. Khakhkhar et.al. [5] proposed a bidirectional planning algorithm that combines a forward-chaining approach and a backward-chaining approach. It allows systems to plan a composite service from the input data and the goal output at the same time. A planning solution emerges when the searches from the two directions meet at some point in the solution's service flow. However, these approaches require central service repositories to maintain service overlays, and they have no support for dynamic composition replanning for composition failures. WSMO [12] describes single-direction planning models, which forward or backward chain service providers for user tasks. WSMO reasons over service execution plans and adapt composite services on the fly, addressing flexible service composition, but still requiring central controllers to schedule services. PM4SWS [16] is a distributed framework to discover and compose web services. However, this service exploration process simply floods the network with query messages for service discovery, which is not suitable for pervasive environments, especially where there are large numbers of services to be considered [31] [40]. A cooperative discovery model [13] provides fully distributed support for unstructured P2P networks. This work is the closest solution to our distributed goal-driven planning approach, but it assumes the network has super-peers[5] that are capable of managing network links. This assumption is not safe in dynamic environments, as super-peer-based groups may fail if any relevant peer leaves the network, or is no longer able to fulfil its role because of reduced capacity. In addition, this work assumed offline planning which limits the potential of the planning solution's dynamic adaptability. This work also included a bi-direction search scheme that allows a data-driven (finding services that match the query's input parameters) service query and a goal-driven service query which start concurrently to increase discovery efficiency. However, this bi-direction search scheme requires an initiator node to aggregate service informations, which implies resource-rich devices, and according to evaluation results [13], using the bi-direction search scheme lowers the discovery success rate. The advantage of the efficient discovery cannot out weight a decrease in successful discovery.

AI-planning algorithms like TLPlan [41], Haley[42], and a fuzzy TOPSIS method [43] have been investigated for dynamic composition planning and have features for automatic re-planning to handle failures. TLPlan [41] allows a system to plan an abstract solution from a pre-existing abstract service repository, and then to discover and bind services for execution. It supports on-the-fly re-composition and execution failure recovery by a composition re-planning mechanism. Haley[42] applied a first-order semi-Markov decision for dynamic composition planning. It also provides composition plan regeneration to recover the system from failures. A fuzzy TOPSIS method [43] is employed to

5. peers with special capabilities [14].

support user-centric service compositions. It adapts service bindings according to real-time user preferences. However, these AI-planning algorithms rely on central composition engines that have not yet been applied on mobile devices. In addition, they need to re-generate a new plan for failure recovery, which is time-consuming and not suitable for dynamic environments.

The proposed service discovery differs from the existing open service discovery and the dynamic composition planning approaches in three aspects. First, a split-choice logic is introduced in the resolution result, which makes the solution more flexible to fit in the execution context. Second, the solution found by the proposed model is adaptable as new split-choice logic can be added in the solution even when the global service discovery is finished. Third, the local discovery process may continue to discover new available services even when the global discovery process ends.

### 6.3 Decentralized Service Execution

The OSIRIS approach [44] decomposes a central service flow description to a set of execution units that can be deployed on service providers in a P2P network. These service providers are found during service discovery. The service execution process migrates from one service provider to the other. At each time such a migration occurs, the client node can select anther node that is available. It defines special observer nodes to monitor the nodes that may cause failure. If a failed node is detected the execution instance can be migrated to another available node. However, it starts execution after service discovery is finished, and it requires that a part of the execution unit is deployed to all the providers. It provides no means to allow a potential new provider that may appear in the network to participate in the composition.

A fuzzy-based service composition [45] has been introduced for MANETs, considering resource-constrained devices and error-prone wireless communication channels. Each node maintains the neighbours' service information and gets the real-time QoS information during service discovery. However, it has no support for adapting composite services at runtime.

A minimum disruption service composition model [10] investigated the types of composition failures in MANET, and provided network-level and service-level adaptation as well as recovery mechanisms. It quantitatively estimated service execution paths' reliability and availability. However, this model has no support for parallel service flows. A cache based service execution and recovery model [11] for MANETs is similar to GoCoMo in terms of using a cache to store backup services. However, it assumes that service providers constantly advertise their services and requires service participants to maintain repositories for neighbours' service information.

The work proposed in this paper extends the opportunistic service execution model [9] [24] but uses goal-driven service discovery beforehand to reduce the possibility of having a dead end execution. This is because the opportunistic service composition does not consider the remaining path's information (e.g., availability or reliability) when invoking subsequent services, which may increase the possibility of invoking a participant that cannot find a service provider for its subsequent execution.

## 7 Conclusion and Future Work

In a service-oriented computing environment where complex user requests can be represented as a collection of data requirements and functionality requirements, each of the requirements can be supported by a set of services. In a mobile, pervasive environment, there is potential for a range of different types of devices (e.g., wearables, mobiles) to participate as a service provider, but challenges include flexible service discovery, and working with mobile devices and dynamic network topologies. Related work, in particular goal-oriented service composition approaches, has presented elegant solutions to flexible service composition using graph-based composition planners, AI-planning or dynamic composition re-planning, but these do not suit mobile environments. In this paper, a novel goal-oriented service composition model is introduced. The model introduces the split-choice logic into the dynamic composition result, thereby handling real-time system environment changes, but it also particularly addresses systems with high mobility.

The proposed model is evaluated against a baseline with multiple simulation scenarios, different request complexity, varying device mobilities and a range of service densities. The results show the goal-oriented service composition model is more successful in returning a composition result than the baseline, has less wireless communication and a faster response during service composition. In future work, we plan to address other QoS attributes in a pervasive computing environment, such as memory or processing power, and optimise the model to cater for these as far as possible.

## References

[1] N. Ibrahim and F. Mouel, "A survey on service composition middleware in pervasive environments," *International Journal of Computer Science Issues*, vol. 1, pp. 1–12, 2009.

[2] V. Raychoudhury, J. Cao, M. Kumar, and D. Zhang, "Middleware for pervasive computing: A survey," *Pervasive and Mobile Computing*, vol. 9, no. 2, pp. 177–200, Apr. 2013.

[3] Y. Wei and M. Blake, "Service-oriented computing and cloud computing: Challenges and opportunities," *Internet Computing, IEEE*, 2010.

[4] M. Handte, "Peer-based automatic configuration of pervasive applications," *Pervasive Services, 2005. ICPS '05. Proceedings. International Conference on*, 2005.

[5] S. Khakhkhar, V. Kumar, and S. Chaudhary, "Dynamic Service Composition," *International Journal of Computer Science and Artificial Intelligence*, vol. 2, pp. 32–42, Sep. 2012.

[6] L. Thomas, J. Wilson, G. Roman, and C. Gill, "Achieving coordination through dynamic construction of open workflows," *Middleware 2009*, 2009.

[7] S. Kalasapur, M. Kumar, and B. A. Shirazi, "Dynamic Service Composition in Pervasive Computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 7, pp. 907–918, 2007.

[8] W. Zhenghui, X. Tianyin, Q. Zhuzhong, L. Sanglu, Z. Wang, T. Xu, Z. Qian, and S. Lu, "A Parameter-Based Scheme for Service Composition in Pervasive Computing Environment," in *Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on*, no. 3. Ieee, Mar. 2009, pp. 543–548.

[9] C. Groba and S. Clarke, "Opportunistic service composition in dynamic ad hoc environments," *IEEE Transactions on Services Computing*, vol. X, no. c, pp. 1–1, 2014.

[10] S. Jiang, Y. Xue, and D. C. Schmidt, "Minimum disruption service composition and recovery in mobile ad hoc networks," *Computer Networks*, vol. 53, no. 10, pp. 1649–1665, Jul. 2009.

[11] X. Zhou, Y. Ge, X. Chen, Y. Jing, and W. Sun, "A Distributed Cache Based Reliable Service Execution and Recovery Approach in MANETs," *2011 IEEE Asia-Pacific Services Computing Conference*, pp. 298–305, Dec. 2011.

[12] A. Hibner and K. Zielinski, "Semantic-based Dynamic Service Composition and Adaptation," in *Services, 2007 IEEE Congress on*, 2007, pp. 213–220.

[13] A. Furno and E. Zimeo, "Efficient Cooperative Discovery of Service Compositions in Unstructured P2P Networks," *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 58–67, Feb. 2013.

[14] S. Oh, D. Lee, and S. Kumara, "Effective Web Service Composition in Diverse and Large-Scale Service Networks," *IEEE Trans. Services Computing*, 2008.

[15] N. Ukey, R. Niyogi, A. Milani, and K. Singh, " A bidirectional heuristic search technique for web service composition," *Computational Science and Its Applications*, 2010.

[16] M. Gharzouli and M. Boufaida, "PM4SWS: A P2P Model for Semantic Web Services Discovery and Composition," *Journal of Advances in Information Technology*, vol. 2, pp. 15–26, 2011.

[17] J. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw, "Task-based adaptation for ubiquitous computing," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 36, no. 3, pp. 328–340, May 2006.

[18] S. Schuhmann, "Adaptive Composition of Distributed Pervasive Applications in Heterogeneous Environments," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2013.

[19] M. Chein and M.-L. Mugnier, *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[20] N. Chen and S. Clarke, "A Dynamic Service Composition Model for Adaptive Systems in Mobile Computing Environments," *IEEE International Conference on Service-Oriented Computing*, 2014.

[21] W. Yu, "Scalable Services Orchestration with Continuation-Passing Messaging," *2009 First International Conference on Intensive Applications and Services*, pp. 59–64, Apr. 2009.

[22] G. Zou, Y. Gan, Y. Chen, B. Zhang, R. Huang, Y. Xu, and Y. Xiang, "Towards automated choreography of Web services using planning in large scale service repositories," *Applied Intelligence*, vol. 41, no. 2, pp. 383–404, Mar. 2014.

[23] R. Liu and A. Kumar, "An analysis and taxonomy of unstructured workflows," *Business Process Management*, pp. 268–284, 2005.

[24] C. Groba and S. Clarke, "Synchronising Service Compositions in Dynamic Ad Hoc Environments," *2012 IEEE First International Conference on Mobile Services*, pp. 56–63, Jun. 2012.

[25] N. Sadagopan and F. Bai, "PATHS: analysis of PATH duration statistics and their impact on reactive MANET routing protocols," *Proceedings of the 4th MobiHoc*, pp. 245–256, 2003.

[26] F. Nah, "A study on tolerable waiting time: how long are Web users willing to wait?" *Behaviour & Information Technology*, pp. 1–37, 2004.

[27] R. Miller, "Response time in man-computer conversational transactions," *Proceedings of the December 9-11, 1968, AFIPS Fall Joint Computer Conference*, vol. 33, pp. 267–277, 1968.

[28] M. Klusch, "Overview of the S3 contest: Performance evaluation of semantic service matchmakers," *Semantic Web Services*, pp. 1–18, 2012.

[29] A. Furno and E. Zimeo, "Self-scaling cooperative discovery of service compositions in unstructured P2P networks," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2994–3025, Oct. 2014.

[30] J. Lipman, H. Liu, and I. Stojmenovic, "Broadcast in Ad Hoc Networks," in *Guide to Wireless Ad Hoc Networks*, ser. Computer Communications and Networks, S. Misra, I. Woungang, and S. Chandra Misra, Eds. London: Springer London, 2009, pp. 121–150.

[31] T. Jun, N. Roy, and C. Julien, "Modeling delivery delay for flooding in mobile ad hoc networks," *Communications (ICC), 2010 IEEE*, 2010.

[32] C. Groba and S. Clarke, "Opportunistic composition of sequentially-connected services in mobile computing environments," *Web Services (ICWS), 2011 IEEE*, 2011.

[33] B. Kiepuszewski, A. Harry, and C. J. Bussler, "On Structured Workflow Modelling Structured Workflows," *Advanced Information Systems Engineering LNCS*, vol. 1789, 2000, pp. 431–445, 2000.

[34] J. Shaffer and J. Keaveney, "Automated concierge system and method," *US Patent 8,160,614*, vol. 2, no. 12, 2012.

[35] J. Breau, E. Miller, S. Ng, and C. Persson, "Concierge for portable electronic device," *US Patent 8,489,080*, vol. 1, no. 12, 2013.

[36] O. Davidyuk and N. Georgantas, "MEDUSA: Middleware for end-user composition of ubiquitous applications," *Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives IGI Global (Ed.) (2011) 197-219*, no. 2011, pp. 1–22, 2011.

[37] P. Rodriguez-mier, M. Mucientes, and M. Lama, "A Dynamic QoS-Aware Semantic Web Service Composition Algorithm," *Service-Oriented Computing LNCS*, vol. 7636, pp. 623–630, 2012.

[38] I. Al-Oqily and A. Karmouch, "A Decentralized Self-Organizing Service Composition for Autonomic Entities," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 6, no. 1, pp. 1–18, Feb. 2011.

[39] K. Ren, N. Xiao, and J. Chen, "Building Quick Service Query List Using WordNet and Multiple Heterogeneous Ontologies toward More Realistic Service Composition," *IEEE Trans. Services Computing*, 2011.

[40] F. Dai and J. Wu, "Performance analysis of broadcast protocols in ad hoc networks based on self-pruning," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 15, no. 11, pp. 1–13, 2004.

[41] M. Vukovi and P. Robinson, "Application development powered by rapid , on-demand service composition," 2007.

[42] H. Zhao and P. Doshi, "A hierarchical framework for logical composition of web services," *Service Oriented Computing and Applications*, vol. 3, no. 4, pp. 285–306, Nov. 2009.

[43] D.-Y. Cheng, K.-M. Chao, C.-C. Lo, and C.-F. Tsai, "A user centric service-oriented modeling approach," *World Wide Web*, vol. 14, no. 4, pp. 431–459, May 2011.

[44] C. Schuler and R. Weber, "Scalable peer-to-peer process management-the OSIRIS approach," *Web Service*, 2004.

[45] G. Prochart and R. Weiss, "Fuzzy-based support for service composition in mobile ad hoc networks," *IEEE International Conference on Pervasive Services*, pp. 379–384, 2007.


**Nanxi Chen** received her BEng degree in Software Engineering from Wuhan University, China, and her MEng degree in Electronic Engineering from Dublin City University, Ireland. She is currently working towards the PhD degree in the Department of Computer Science, Trinity College Dublin, Ireland. Her research interests include mobile and pervasive computing, composition of (mobile) services, and semantic service discovery. Her research is funded by the Science Foundation Ireland (SFI).

**Nicolás Cardozo** received his PhD degree in computer science from the Université catholique de Louvain and Vrije Universiteit Brussel in 2013, specializing on programming language design for context-aware environments. He is a research fellow working in the Distributed Systems Group at Trinity College Dublin. The focus of his research is in the field of self-adaptive systems, studying analysis and verification techniques to assure adaptation consistency. His research is partially funded by the Science Foundation Ireland (SFI) grant to Lero and FP7 grant to the DIVERSIFY project.

**Siobhán Clarke** is a Professor and Fellow of Trinity College Dublin, where she leads the Distributed Systems Group and is Director of Future Cities, the Trinity Centre for Smart and Sustainable Cities. Her research interests are in software engineering models for the dynamic provision of smart and dynamic software services to urban stakeholders in large-scale, mobile environments. She received her BSc and PhD degrees in Computer Science from Dublin City University.