Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Doctoral Thesis

Goal-Driven Service Composition in Mobile and Pervasive Computing

---

Author:
Nanxi Chen

Supervisor:
Prof. Siobhán Clarke

A thesis submitted in fulfilment of the requirements for

the degree of Doctor of Philosophy

in the

School of Computer Science & Statistics
Trinity College Dublin, The University of Dublin

October 2015

Declaration of Authorship

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

Signed: Nanxi Chen

Date: October 2015

Abstract

Pervasive computing environments enable access to diverse resources and services over networked computing systems. Mobile systems have the potential to be very active participants in such environments as resource providers, since they are in wide-spread use, and can sense and exchange their operating environments' context data. Service-oriented computing has emerged as an important paradigm in pervasive computing, because it packages heterogeneous resources as services that are discoverable, accessible, and reusable. Services offered by potentially multiple devices can be composed to create a new value-added service. Service provision through service composites is explored in this thesis, particularly in pervasive environments where service providers are mobile and communicate with each other in an ad hoc manner.

Mobile service providers are free to join and leave a system, making the availability of the services they provide unpredictable. Service execution may fail because of a previously available service provider's absence at runtime. There is significant potential for improving overall service quality in real-time services provisioning by re-composing better services from the environment including those that may have appeared even during service execution. Mobility also changes the network topology and the links between services, which can lead to execution path loss, and in turn composition failures at runtime. Thus, service composition requires a comprehensive and dynamic discovery model to reason about an appropriate combination of services that match the given functionality, as well as an efficient mechanism that adapts composite services to dynamic environments.

Existing research on service-oriented computing has led to automatic planning, adaptive composition and composition recovery to tackle dynamic environments, but requires global service knowledge or a view of the real-time service links. Given mobile devices' limited communication ranges, the network topology changes quickly when devices are roaming, and keeping such system views up-to-date leads to additional communication, maintenance overhead, and may delay the composition process.

This thesis presents a fully decentralized services composition model that supports flexible service discovery and execution in mobile pervasive environments. The model is goal-driven, focusing on time-efficient service provisioning to reduce the interference

of topology changes. This goal-driven approach achieves flexible service discovery by dynamically planning a service workflow, which supports not only sequential service composites but also complex composites such as parallel or hybrid service flows. Service links' reliability and quality of service issues are considered when selecting services for invocation, which reduces the possibility of execution failures and the effort required for maintaining backup services for composition recovery. If necessary, failure recovery is attempted by adaptable OR-split transitions in the service workflow.

The model has been evaluated using both simulation and a prototype case study. Evaluation metrics include measurements of composition success rates under various mobility models, and the composition model's scalability and performance. Simulation results illustrate both the strengths and the limitations of the proposed mechanism in dynamic pervasive computing environments, under different network density and composite complexity conditions. The prototype case study demonstrates this approach's feasibility on real mobile devices.

Acknowledgements

I would never have been able to finish this thesis without the guidance of my supervisor, help from group members and friends, and support from my family.

First and foremost, I must thank my supervisor Prof. Siobhán Clarke, who brought me the exquisite beauty of precision, and rigorous research. Thanks for her expertise, continuous support, caring and enormous patience over the years.

Thanks to the adaptive systems team, Vivek, Nicolás, Eamonn, Saeed, Amit and Pawel for being such a wonderful team, always being available for discussing a idea, and the useful and insightful comments to improve this work.

Finally, special thanks to the financial support from SFI and Lero who offered me the great opportunity to conduct research in the amazing country, Ireland, and also gave me the chance to meet a number of researchers through the annual workshops and meetings.

Nanxi Chen

The University of Dublin, Trinity College

October 2015

Publications Related to this Ph.D.

Nanxi Chen and Siobhán Clarke, A Dynamic Service Composition Model for Adaptive Systems in Mobile Computing Environments, *IEEE International Conference on Service-Oriented Computing - ICSOC*, 93-107, 2014.

Nanxi Chen, Nicolás Cardozo and Siobhán Clarke, Goal-Driven Service Composition in Mobile and Pervasive Computing, *IEEE Transactions on Services Computing, Volume-PP , Issue-99, February 2016*

Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
CtrlLogic	**C**on**t**r**o**l **Logic** helper
DCON	**D**ynamic **C**omposition **O**verlay **N**etworks
DHT	**D**istributed **H**ash **T**able
ExeGM	**Exe**cution **G**uidepost **M**anager
GClientE	**G**oCoMo **Client E**ngine
GMsgHelper	**G**oCoMo **M**ess**a**g**e Helper**
GoCoMo	**Go**al-driven service **Co**mposition in **Mo**bile and pervasive computing
GProviderE	**G**oCoMo **Provider E**ngine
I/O	**I**nput and **O**utput
MANET	**M**obile **A**d hoc **Net**work
P2P	**P**eer to **P**eer
QoS	**Q**uality of **S**ervice
SOC	**S**ervice-**O**riented **C**omputing
SON	**S**emantic **O**verlay **N**etworks
TTL	**T**ime-**T**o-**L**ive

Dedicated to my mother, Rui Chen

Chapter 1

Introduction

Pervasive computing environments enable access to diverse resources and information over a networked computing system. Such a system includes traditional computers as well as embedded devices, information appliances, and sensors [Brønsted et al., 2010, Weiser, 1991]. During the last decade, breakthroughs in achieving faster and energy-efficient wireless communication as well as smarter and thinner embedded devices have accelerated human users' shift away from traditional computers and towards *mobile* and embedded devices for resource access and information sharing. Mobile devices support a wide perspective on the environment with context-aware functionality, such as multi-modal mobile-sensing and ambient proximity social sharing [Conti and Giordano, 2014], which raises the potential for mobile devices to be active participants in pervasive computing environments as resource and context information providers. This potential leads to increasing attention to mobility issues in pervasive computing research.

1.1 Service-Oriented Computing

Pervasive computing environments have evolved from closed (special purpose) and static to **open and dynamic (mobile)** [Brønsted et al., 2010, Ibrahim and Mouel, 2009]. A mobile pervasive computing environment can include a large number of third-party mobile entities (e.g., wearable technologies, smart phones, etc.). As modern wireless communication technology facilitates wireless data exchange for mobile users, various information captured by smart mobile devices like news, locations, air quality, reviews,

routes/directions, and even parking/loading data, can be shared through wireless networks [Khan et al., 2013, Lane et al., 2010, Perera et al., 2015]. Service-oriented computing (SOC) is at the forefront of enabling access to such shared information for pervasive computing environments [Ibrahim and Mouel, 2009, Raychoudhury et al., 2013], packaging heterogeneous resources as services that are discoverable, accessible, and reusable. It also provides unifying interfaces for services to ease users' access via communication networks. To address a particular user requirement, a combination of multiple services may be required, and so a fully-functional service composition process will tackle complex user requests with flexible composition of value-added services. Required functionalities such as information query and concierge-like services can be complex and mutable, making it difficult for a single mobile device to handle as it may have inadequate functionality or computing resources. Open and dynamic pervasive computing environments should compose functionalities from heterogeneous mobile devices that have the potential to collaborate.

This thesis focuses on service provision through service composites, particularly in pervasive environments where service providers are mobile and communicate with each other in an ad hoc manner. A pure ad hoc network is considered in this thesis, but the process remains the same for all public environments where a central admission cannot be assumed.

1.2 Challenges

Given the environment's *openness and dynamism*, service composition processes face significant challenges (as illustrated in Figure 1.1). The primary challenges are:

C.1 Inadequate conceptual composites

A conceptual composite is an abstract model that states a specific functionality as a series of service requirements, each of which indicates that a concrete service will be used. General service composition relies on previously generated conceptual composites and assigns service providers at runtime to complete them. However, in an open environment, services of interest are independently deployed and maintained by different mobile devices so that a conceptual composite for a particular

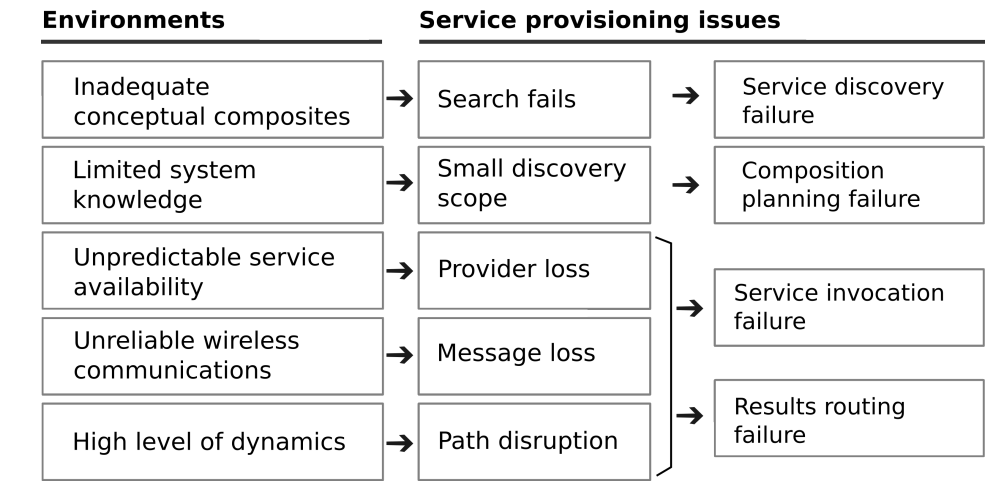


FIGURE 1.1: Composition-based service provisioning challenges in open and dynamic pervasive computing environments

functionality is not always possible. Composition users may provide the conceptual composite as a part of the service request, but such a user-defined composites is likely to be at variance with the operating environment as the environment is dynamic. In addition, a composition user's awareness on available services is limited by its communication range, so the composition user may have insufficient knowledge about usable services and cannot produce a conceptual composite for service composition. A predefined conceptual composite also removes the possibility of using services that may contribute to the user's request, but are not outlined in it.

C.2 Limited system knowledge

Service discovery searches for, and selects services that match the required functionality. A service provision system may need to dynamically reason about a functionality and find an appropriate combination of services that supports it when an individual one is unavailable. Having a global system view of the computing environment is beneficial for such a reasoning-based service discovery, since a global service knowledge will facilitate reasoning processes and increase composition success. However, obtaining service knowledge from mobile devices leads to traffic overhead because it relies on multi-hop wireless data transmission and when mobile devices have a limited communication range, there are likely to be a larger number of transmission hops. In addition, maintaining a global system view

can be expensive especially when the service and network topology are frequently changing.

C.3 Unpredictable services availability

Service providers establish a wireless network in ad hoc ways [Brønsted et al., 2010]. Any mobile service provider can offer and drop services, as well as join and leave the network at arbitrary times during execution, making services' availability unpredictable at runtime. Service execution can fail due to a previously available service's absence. New service providers can enter the network and offer new services, which brings significant potential for improving overall service quality by re-composing better services from the environment including those that may have appeared even during service execution.

C.4 Unreliable wireless communications

Mobile devices rely on wireless communication channels to exchange service information, bind services, and transfer data during service execution [Ibrahim and Mouel, 2009, Mian et al., 2009]. Wireless communication is likely to be unreliable, and the data packet transmission via the radio channel is sensitive to interference and obstruction, with packet loss comes as a result [Fok et al., 2010]. Efficient interactions between composite participants are required to reduce the dependency on such a communication channel.

C.5 High level of dynamics in execution paths

Service execution requires communications through established wireless links between successive service providers. However, service providers' *mobility* leads to network topology changes, which alters such links. Changes to the links make any cached execution path error-prone, which may further result in communication failures and execution path loss during service execution. A composite service must be adaptable to increase the chance that results can be delivered even when a communication channel to a provider drops, or a cached execution fails. Recovering the composition from a failed path in a time- and communication-efficient way is also required to achieve a successful service composition.

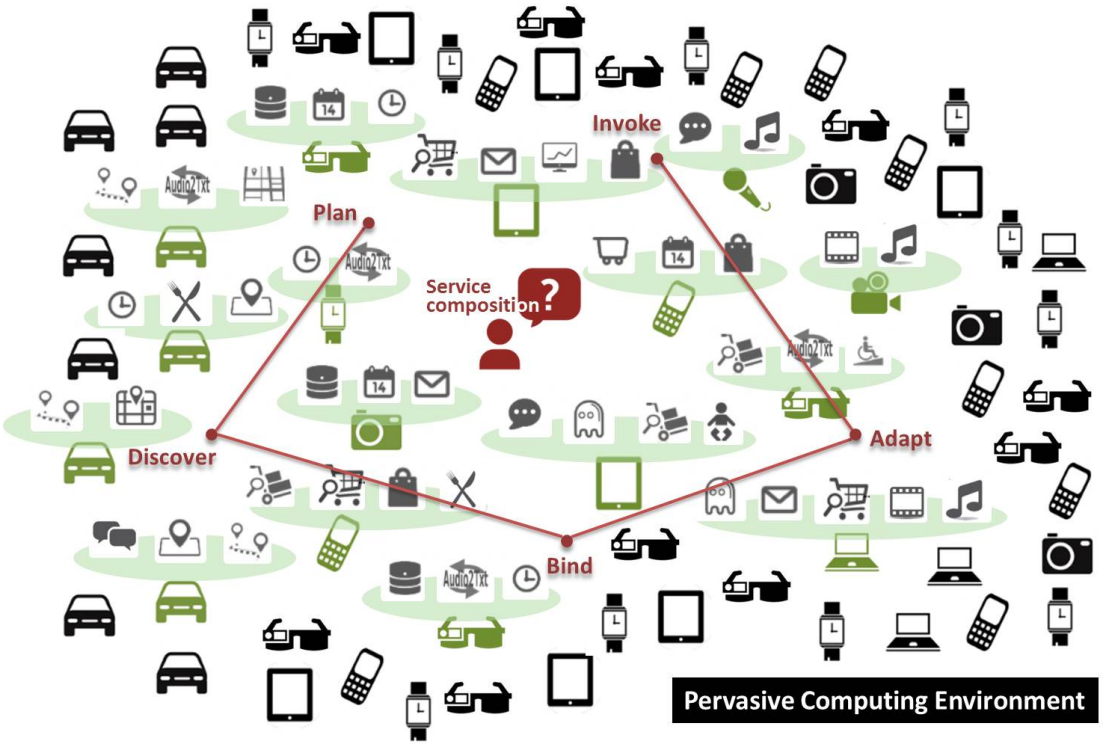


FIGURE 1.2: Motivating scenario: a smart public space system. (*A user issues a complex service request to a shopping mall's pervasive computing environment. Mobile entities offer their hardware/software capabilities and local data as services.*)

1.3 Motivating Scenario: A Smart Public Space

As a motivating scenario, Anne is in a shopping mall's car park with her 1-year-old son. She would like to order and collect a pack of nappies in a local store, and then go to the nearest baby-changing facility in the mall. She needs a order confirmation for order collection, as well as a step-free route to the store and from there to the baby-changing facility, 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 graphic route properly.

There is a smart public space system (Figure 1.2) in the mall including various embedded devices owned by customers, shop clerks, taxi drivers, or house keepers, etc. These devices can package their capabilities, like GPS, navigation, translation, 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. Thus, her smart watch may be able to directly get a composite service (Figure 1.3) via an ad hoc network that forms from different devices in the

pervasive computing environment, such as a housekeeper's phone, a nearby car's satellite navigator, a shop porter's smart glasses, a personal shopper's tablet, etc. The composite service allows Anne to input only a product name and a quantity number, and returns an order confirmation code to allow her to collect and pay for her order in-store. The service also returns an audio route stream that compacts with her smart watch, guiding her to the store and the baby-changing seat.

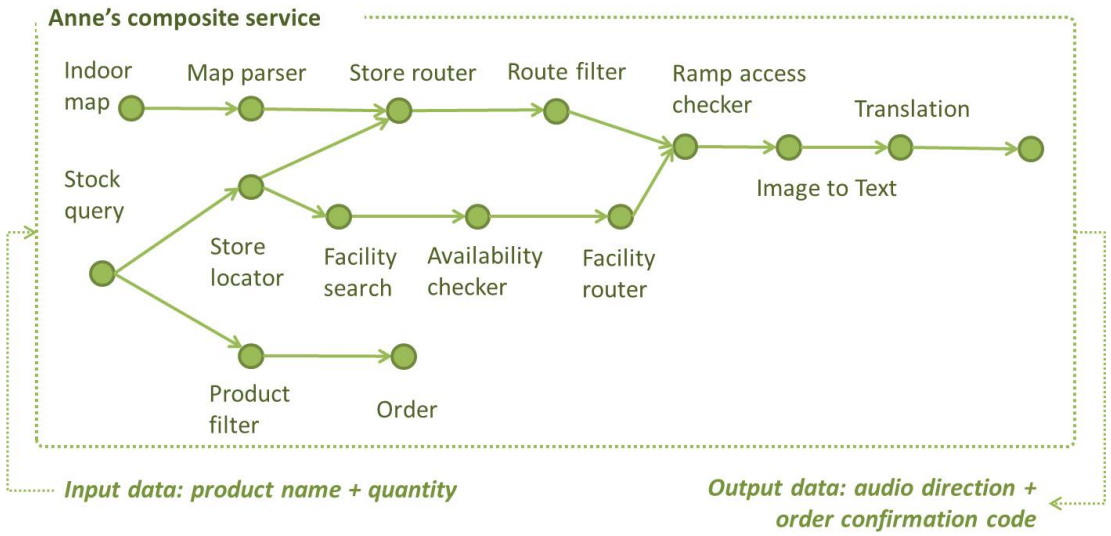


FIGURE 1.3: Anne's service composite

This smart public space has the potential to help Anne avoid browsing the website, increasing the likelihood of matching all her hardware and software capabilities to get the routing result she needs. However, Anne's requirements are complex and domain-crossing such that an exactly defined service composition task may not be flexible enough to facilitate successful matching between her requirements and the available services in the environment. To enable service provisioning in such a smart public space, a system should be capable of allowing (possibly third-party) service providers to join the environment, discovering services based on the given requirements in a flexible way, producing a service workflow for data transition, and invoking service instances for execution. Moreover, participant devices that host services are likely owned by heterogeneous third-parties, such as pedestrians, taxi drivers or housekeepers, and these device owners' activities are not under obligation to the system. The participant devices, therefore, may leave the system or drop the service before execution. This thesis assumes that services deployed on participant devices are stateless. For successful service provisioning, the system should support timely service invocation, and be aware of and engage newly

entered devices that have the potential to offer the given functionality throughout the service composition process. In short, the system should address the challenges listed in the previous section.

1.4 Existing Solutions

Service composition's success in open and dynamic environments is subject to failures that can occur in any phase (Figure 1.1) of the composition process. Support for services provision using composite services in such environments should include both composition process management and failure recovery. Composition process management investigates different ways to organize a composition process to increase composition success. Fault tolerance explores methods to predict potential failures and adapt the composite service for them, or to recover a service composition process from an emerged failure.

1.4.1 Composition Process Management

A service composition process is initiated by the service provision system when a composition request is issued. The process refers to the tasks of reasoning about a composition plan, discovering available services that match the plan, binding the discovered services and invoking service instances. Challenges when designing a service composition process include the question of when and in what order these tasks are performed, which entity in the network executes the tasks, where the service provision system retrieves necessary knowledge to support such execution, and how these tasks are managed. The state of the art for service composition in pervasive environments includes runtime composition reasoning [Hibner and Zielinski, 2007], decentralized service composition [Al-Oqily and Karmouch, 2011], dynamic service binding [Prinz et al., 2008, Wang et al., 2013], weaving the service invocation procedure to the service discovery procedure [Groba and Clarke, 2014], etc., which cope with flexibility and context dynamism. Existing solutions adopt or expand these techniques. The main techniques that addresses the challenges of the target system (Section 1.2) are planning-based service composition and decentralized composition management.

Planning-based service composition differs from traditional matching-based services composition approaches that rely on a given conceptual composite that is made up of a set

of ordered abstract services, and only finds service instances that exactly match the abstract services. Planning-based service discovery is not restricted to exactly matching a composition task, making it more flexible with the potential to a broader scope of services than matching-based services composition. Planning-based service composition solutions extend conceptual composites or use AI planning algorithms to increase flexibility. Conceptual composite extensions uses a one-to-more matchmaking scheme for service composition, which means one abstract service may match a combination of services [Al-Oqily and Karmouch, 2011, Kalasapur et al., 2007, Liu et al., 2015a, Rodriguez-mier et al., 2012, Thomas et al., 2009]. However, this model still requires a conceptual composite that is planned based on offline service information, which may be out of date. On the other hand, AI planning algorithms are goal-oriented, automatically creating abstract composites (composition tasks) during service discovery, selecting services that matches each composition task, and finalising the composites by invoking selected service instances at runtime. Some of them can support various service workflows other than sequential workflows but require planning infrastructures [Furno and Zimeo, 2014, Gharzouli and Boufaida, 2011, Khakhkhar et al., 2012, Liu et al., 2015b, Ukey et al., 2010] or a global knowledge base[Khakhkhar et al., 2012, Oh et al., 2008, Ren et al., 2011, Ukey et al., 2010].

Decentralized composition management [Prinz et al., 2008, Schuler et al., 2004, Sen et al., 2008] distributes a conceptual composite to different participants, each of which carries out a part of the abstract services. In general, overall service invocation starts only after every abstract service in the composite has been bound to a provider [Artail et al., 2008, He et al., 2008]. This protects a composition from unnecessary invocation and execution when some of the required providers are non-existent [Sen et al., 2008]. However, even when each required provider has been located and bound, the composition can still fail because of absent service providers at runtime. Minimizing the impact of service availability has been explored by employing on-demand service binding [Groba and Clarke, 2014]. On-demand service binding assigns a provider to an abstract service just before it has to execute. Existing approaches have explored on-demand service binding in two directions: one discovers a set of providers for every required service as allocation candidates and maintains these candidates until one of them is invoked for execution, and the other discovers a provider at runtime and invoke its service immediately after the provider is found. However, using allocation candidates [Prinz et al., 2008, Schuler et al.,

2004] requires additional monitoring effort to keep the list of candidates up to date, and limits the chance of engaging better providers that may appear in the environment after all the allocation candidates for a service are determined. On-demand service binding [Groba and Clarke, 2014] can obtain real-time service provider information by runtime service discovery. It reduces composition latency by coupling the composition process with the invocation of services. However, the on-demand service binding approach relies on exactly matching functionalities and predefined conceptual composites for service discovery, which reduces flexibility. Moreover, it assumes broadcast-based service discovery, and provides no means to prevent a service composition request flooding the network. Furthermore, it migrates composition processes along the direction of data flow, which may mean the system invokes a final provider a long distance from, or a bad transmission channel to, the user. This increases the cost of routing the composition result back to the user.

In summary, there is a lack of a service composition process to resolve clients' diverse composition requests, by using only participant providers' local service information. Its service discovery method should be infrastructure-free, aware of new service provider that newly emerges, and can control discovery message flooding to reduce congestion in the network.

1.4.2 Fault Tolerance

Research on fault tolerance in services composition focuses on failure prevention and composition recovery. Failure prevention approaches tend to anticipate composition failures and update the composite service, which requires continuous observation of changes in the operating environment throughout the overall composition process and the adaptation of the composite to observed changes [Prochart et al., 2007, Schuler et al., 2004]. Observations on the operating environments are normally achieved by sending probe messages to composition participants to assess the allocated services' availability and the validity of links between service providers [Prinz et al., 2008, Schuler et al., 2004]. As the operating environment is highly dynamic, an early failure prediction tends to be inaccurate. Such a prediction is likely to lead to unnecessary adaptation. On the other hand, a just-recently-detected impending failure leaves very limited time for a composite service to update itself. Failure prevention may fail if a composite has not completed

when a failure occurs, and in turn cause composition disruption. Compositions can be recovered by discovering service providers to replace failed ones [Gu and Nahrstedt, 2006, Prinz et al., 2008, Zhou et al., 2011] or by selecting a backup composite for re-execution [Hibner and Zielinski, 2007, Jiang et al., 2007]. However, discovering new service providers is inefficient and can delay the composition result. Keeping backup composites avoids the runtime cost of communication for rediscovery, but it may cost the system more effort to maintain these backups.

1.4.3 Research Gaps and Observations

Existing solutions for service composition in pervasive computing have explored composition process management and fault tolerance to address openness and dynamism. As mentioned, composition process management solutions include efficient service discovery [Kang et al., 2008, Pirrò et al., 2012], flexible planning [Al-Oqily and Karmouch, 2011, Furno and Zimeo, 2014, Kalasapur et al., 2007, Rodriguez-mier et al., 2012], dynamic binding [Groba and Clarke, 2011, 2012, 2014], decentralized composition management [Furno and Zimeo, 2014, Groba and Clarke, 2014, Prinz et al., 2008, Sen et al., 2008]. Most planning-based service composition approaches rely on discovery infrastructures and a-prior system knowledge [Al-Oqily and Karmouch, 2011, Kalasapur et al., 2007, Rodriguez-mier et al., 2012]. Decentralized composition management solutions are restricted to exact matching functionality or planning for a sequential service composite [Furno and Zimeo, 2014, Groba and Clarke, 2014, Prinz et al., 2008]. Fault tolerance either requires composition re-planning that is time-consuming [Vukovi and Robinson, 2007, Yu, 2009], or only replaces failed providers with backup ones without considering the reliability of the new execution path [Gu and Nahrstedt, 2006, Hibner and Zielinski, 2007, Jiang et al., 2007, Prinz et al., 2008, Zhou et al., 2011].

Taking full advantage of the mobile and wireless technology advances and opportunities in open, dynamic pervasive computing environments will therefore require the development of new, more appropriate, service composition models that provide:

1) flexible decentralized composition of services on mobile devices. The target network that is established by ad hoc connected mobile devices is infrastructure-less, requiring that the process performs in a decentralized manner. Models are also required to streamline to achieve efficient composition.

2) dynamic adaptation of the combination of services as appropriate to the service providers' and the environment's changing situation. Where there are constraints such as timeliness, differing data flows and combinations of services, will be appropriate under different circumstances. Models are required to quantify services' and their combinations' characteristics (e.g., service availability, execution path reliability) and to adapt as appropriate to environmental factors.

1.4.4 Research Questions

This thesis explores *the question of how to enable service provisioning through composite services in open and dynamic pervasive computing environments.* This question can be decomposed into:

RQ.1 how to discover service providers relying on limited system knowledge to trade off successful service discovery with time and traffic cost?

RQ.2 how to plan a service composite that is sufficiently up-to-date and flexible to support appropriate decomposition of a user requirement and to manage overall quality of service of the service composition such that the executing system will adapt appropriately to changing environmental factors?

RQ.3 what adaptation model will be required to re-compose service capabilities depending on not only individual service's quality but also the composite service's reliability, and that will itself, behave in a timely and communication-efficient fashion?

1.5 Thesis approach

The effectiveness of service provisioning can be affected by the complexity of the composite service, the length of service provider links, composition latency, composition distribution, and the cost of failure prevention and recovery. Specifically, a list of objectives is illustrated, based on investigations on the state of the art solutions.

O.1 A service execution path contains logically linked services. Such service providers are connected in an ad hoc manner, depending on mediators to relay messages.

With more mediators for a logical service link, more message forwarding will be required, and such a link may be error-prone because of unstable wireless channels. Further, traditional distributed service composition requires four message transmission: to route a composition request, to locate a service provider, to bind the service provider, and to invoke its service instance. Frequent message transmission over unreliable links is at high risk of packet loss. This work aims to reduce dependency on wireless transmission and the number of message routing hops.

O.2 Composition latency is also an important factor that affects the Quality of Service (QoS) and the success of service provisioning [Groba and Clarke, 2011]. Service providers are grouped to support a composite of their services. As the providers may be mobile, the group may change over time with regard to the participants' topology, and the longer each participant has to wait for execution the more deviations of the composite from the initial one are likely to occur. Such deviations may imply longer service links or even service absences, which can delay messages that pass through the service provider links, and so lower the composite service's quality, increasing the latency of the composition result [Friedman et al., 2007]. Even worse, the composite service may be invalid in terms of service availability, or service link availability. This work's goal is to design a service composition model that always use current service information to support service selection, and minimize standby duration for selected service providers.

O.3 Complex composite services are likely to have long service execution paths, or complicated control logic, which may cause time-consuming execution. In addition, given the fact that any participant has the opportunity to change their status (i.e., location, speed, and service availability) over time, a complex composite service is likely to be under threat of failure caused by network status changes. This work explores a service composition model that can appropriately decouple user requests to satisfy required functionality with a simplest possible composition result.

O.4 Composition distribution refers to the question of where the service specification is registered and who manages the composition process. Generally, the more centralized a service composition process is, the more communication overhead for its maintenance in dynamic environments, and the less communication required for service searching [Raychoudhury et al., 2013]. In a highly dynamic environment,

the benefit of efficient communication for service searching could be outweighed by the maintenance cost. This work aims to self-organise the composition process while leveraging an efficient search mechanism.

O.5 Fault tolerance handles faults that (possibly) emerge during service composition by replacing (potential) failed service providers or a fragment of the service execution path which is invalid. Preventive failure recovery requires prediction about potential failures, and continuous composition re-planning of the service execution path. Reactive failure recovery does not include a process to monitor operating environments. It adapts the composite service after a failure emerges, however, has more recovery delay than preventive failure recovery. We adapt a service composite to changing environments to prevent failures without affecting executed or currently executing services. A lightweight mechanism is considered to recover a service composite from failures.

In short, this thesis aims to select a set of services that support the required functionality and to compose them in a decentralized manner, trading off flexibly time-efficient failure recovery with cost. It explores decentralized service discovery based on AI-planning to dynamically retrieve up-to-date service specification (Objective O.3 and O.4). As later services are combined, there is less time for the service combination to change (Objective O.2), and composite service construction is explored to act only when a service is about to be invoked [Groba and Clarke, 2014]. This thesis focuses on the runtime changes to a composite service, and explores enabling dynamic adaptation for failure prevention without constantly monitoring the participants or composition re-planning, by decoupling and re-organizing the composition planning process (Objective O.1 and O.5). Execution path recovery is also investigated to recover a failed service execution path, and recompose services with minimal re-execution effort (Objective O.5).

1.5.1 A Decentralised Goal-Driven Service Composition

A composition request is resolved using a novel decentralized goal-driven services composition model that decouples the request into a series of sub-goals and uses backward resolution. By locating the final providers first and then the rest of providers, the model considers the routing cost for the delivery of the final composition result when selecting execution paths. During the backward composition resolution process, a request

for sub-goals is issued by service providers, each of which solves at least one sub-goal. This decentralized model means all the providers are responsible for the composition, and admission control migrates from one provider to its subsequent providers, through service invocation. A provider controls the composition by generating a sub-goal and forwarding it across the network during the resolving process, selecting the remaining execution path and invoking a subsequent service during the execution process.

Service binding is postponed and merged with the service invocation process, which reduces interactions for the composition to make progress. Using backward planning and forward binding/invocation, the model requires communication with a service provider only twice: first to discover whether it is a potential service provider and second to execute the composite service by performing service selection, resources locking, and service instance invocation in one communication.

It is possible, if not probable, that a sub-goal could be satisfied by different combinations of services, so a service composition model is likely to have alternative execution paths for service invocation. In the backward composition resolving process, the reliability of each fragment of a service execution path is estimated, and late service binding allows a service composition system to select the most reliable execution path according to the estimated reliability. If necessary, the composition process can go back to a previously executed provider to invoke a new execution path and replace a failed one.

1.6 Thesis Contribution

This thesis proposes a novel **Go**al-driven service **Co**mposition model for **Mo**bile and pervasive computing environments, called GoCoMo. GoCoMo performs service discovery, planning, binding, invocation and adaptation in a fully decentralized manner. The model is goal-driven, and focuses on time-efficient service provisioning to reduce interference from topology changes. GoCoMo achieves flexible service discovery by dynamically backward planning a service workflow, which supports not only sequential service composites but also complex composites such as parallel or hybrid service flows. The service workflow is made up of a set of adaptable fragments, making it more malleable for use in dynamic environments. Service execution paths' reliability and quality of service issues are considered when selecting services for invocation, which reduces the possibility of

execution failures and the effort required for maintaining backup services for composition recovery. If an execution failure emerges, composition recovery is attempted by adaptable OR-split transitions in the service workflow.

This thesis describes an in-depth study into the area of service composition in dynamic pervasive computing environments, and outlines the following extensions to existing composition models as contributions to knowledge.

- **Heuristic service discovery** (RQ.1)

  Current service discovery in dynamic ad hoc networks is achieved by n-hops[1] flooding, multicast routing based on directories, or selective routing based on overlay networks [Mian et al., 2009, Raychoudhury et al., 2013]. Infrastructure-based discovery (directories or overlays) is communication efficient at runtime, but introduces maintenance cost. n-hops flooding is infrastructure-less, and existing approaches concretely define n, which is inflexible. This thesis explores how to use user-defined composition time constraints, local topology knowledge and backward planning to trade off broadcast traffic overhead with service discovery scope in service planning. The proposed heuristic service discovery prevents a request flooding the network in two ways: *1)* estimating routing reliability for a service and stopping unreliable service routing, and *2)* anticipating if a fragment of the service path is reachable during execution according to the remaining path's length and the given time constraints, and dropping request forwarding for unreachable fragments.

- **Support for flexible decentralised composition planning** (RQ.2)

  Existing approaches towards distributed composition planning either handle only sequential service flows for such a composite service, or employ AI planning algorithms that support various service flows but require centralized planning infrastructures. GoCoMo automatically plans for potentially diverse service flows including parallel and hybrid flows, and allows for dynamic selection among them. This model engages providers that are likely to support a given functionality by forming a parallel service flow as a composition candidate which increases composition flexibility, as illustrated in Chapter 2.

[1]The n is a predefined natural number limits the maximum number of routing hops.

- **Selection of adaptable execution fragments** (RQ.3)

  Traditional service selection mechanisms that choose a full composite before execution are inflexible as the composite cannot adapt to dynamic environments. Selecting service providers during the service execution process has the potential to adapt the composite service at runtime, but existing approaches select only based on the QoS of each individual service. GoCoMo enables runtime execution fragment selection that decomposes candidate service execution paths into a set of execution fragments and weaves these fragments in a selection process. The proposed selection is according to the path's pre-estimated runtime reliability, and the fragment can include candidate providers. The selected fragment is also adaptable as *1)* when providers enter the system's scope, they automatically merge with the selected fragment to resolve user requirements, and *2)* if candidate providers included in the selected fragment are detected as unavailable, they are removed from the fragment.

1.7 Thesis Scope

This work mainly considers service execution time and the length of the link as the factors that influence path reliability. However, path reliability can also be affected by other factors, for example, throughput [Rodriguez-mier et al., 2012], service reputation, service reliability, execution price [Dai et al., 2015], energy, or bandwidth [Efstathiou et al., 2014a]. Energy and bandwidth issues have been discussed as QoS attributes to achieve QoS-based service composition in dynamic ad hoc networks [Efstathiou et al., 2014a, Li et al., 2007, Park and Shin, 2006, Wu and Huang, 2015]. Further research on modelling path reliability in the target environment with more QoS attributes will be required to support more exhaustive reliability estimation, resulting in possibly less composition disruption.

This work discovers services with a combination of multicast routing and constrained flooding. It searches for services by semantic matching the composition goal provided by a client to the input/output (I/O) parameters of a service, but no specific matchmaker (e.g., iSeM, OWLS-SLRlite, COV4SWS, etc.) or Semantic Overlay Networks (SON) is assumed in this work. The feasibility and performance of different semantic matchmakers

has been studied by Klusch [2012], and are outside the scope of this thesis. A SON may introduce maintenance overhead to the system, however, semantic dependency overlays [Chen and Clarke, 2014] have the potential to be used to support the discovery model described in this thesis, and may achieve more efficient service discovery in a particular scenario where dynamic semantic matchmaking is time intensive.

This thesis focuses on the application-level and addresses the combination of services instead of devices. Service composition can combine services from different devices or the same device. Existing research [Bianchini and Antonellis, 2008] distinguishes the link that connects services in the same device and that connects services in different devices as inter-peer links and intra-peer links, respectively. Inter-peer service connections and the management of them has not been studied. Analysis of different inter-peer link estimation and management mechanisms [Prinz et al., 2008] will be needed to improve the flexibility of GoCoMo.

1.8 Thesis Structure

State of the art Chapter 2 reviews the state of the art and explores how it meets the service composition challenges (Section 1.2) in open and dynamic pervasive computing environments. It applies an assessment metric to review the feasibility of these solutions and classifies them with regard to all the research questions and objectives (Section 1.5). The chapter also assesses how pre-execution composition processes to reduce discovery and planning failures are organised, and how the state of the art deals with message and path loss during service execution.

Design Chapter 3 describes the design objectives and lays out the system model in detail. In particular, it begins by describing the requirements that should be satisfied and the trade-offs based on the analysis in Chapter 2. It then describes the GoCoMo model as a proposed solution to the problem of service composition in open and dynamic pervasive computing.

Implementation Chapter 4 describes the implementation of GoCoMo and introduces a support middleware. It also presents two prototypes. An Android-based prototype

realizes GoCoMo as a middleware application. A C++ implementation integrates GoCoMo with the network simulator NS-3 for evaluation. It is realized as an extension module on the NS-3 platform.

Evaluation Chapter 5 evaluates how GoCoMo fulfils the identified challenges and research questions by comparing to baseline solutions from the state of the art. It begins with the introduction of evaluation metrics for a measurement of GoCoMo and the baseline approaches in the target environment. Evaluation metrics include measurements of composition success rates under various mobility models, and the composition model's scalability and performance. This chapter continues by introducing a prototype case study that demonstrates GoCoMo's feasibility on real mobile devices, and an simulation-based experiment. Simulation results illustrate both the strengths and the limitations of GoCoMo, under different network density and composite complexity conditions.

Discussion Chapter 6 analysis this thesis's achievements, summarises this work and presents a list of interesting open issues that require further research.

Chapter 2

State of the Art

This chapter surveys current research that explores, to varying degrees, dynamic, mobile-aware service composition. Existing surveys illustrate the maturity of service composition in pervasive computing environments [Ngan and Kanagasabai, 2012, Raychoudhury et al., 2013], P2P networks [Meshkova et al., 2008, Rambold et al., 2009], and mobile ad hoc networks [Cho and Lee, 2005, Christopher N. Ververidis and George C. Polyzos, 2008, Hosseini Seno et al., 2007, Mian et al., 2009]. This chapter extends current surveys with an assessment on how the state of the art approaches achieve flexible service composition in an open and highly dynamic environment. It assesses the related projects in the context of service composition process management and fault tolerance, in each of which a subset of the challenges identified in Section 1.2 is addressed. The structure of the investigation and assessment is presented in Figure 2.1. Firstly, this chapter explores service discovery protocols and the questions of how provider information is discovered, how a composition request is routed to the system, and how a flexible service composite is achieved. It then analyses service binding mechanisms with particular focus on how and when providers are selected and lock their resources for a composition. Next, it assesses how the service execution process is carried out by the systems. Finally, a discussion on fault tolerance for service composition processes is presented.

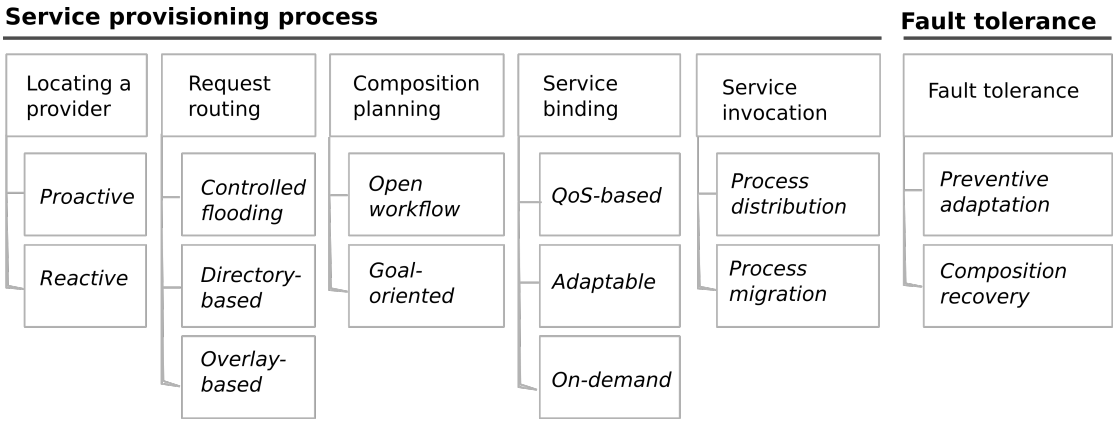


FIGURE 2.1: Structure of state of the art review

2.1 Locating a Provider

Service providers specify their local capabilities as services and make this service specification available to service provisioning systems. Service discovery in mobile ad hoc networks includes active search and passive search [Mian et al., 2009][1]. Reactive discovery (a.k.a., passive search) means that a service provisioning system searches services on an entity (or a set of entities) that previously receives providers' announcement and caches their service specifications, and proactive discovery (a.k.a., active search) model directly enquires service providers for their service specification.

Specifically, *reactive discovery* requires service providers to register service specification to the system by periodic service advertisement. Networked entities in the system cache such advertisements and manage them in a particular structure for quick query. Decentralised caches are attractive in pervasive computing environments, as they better suit mobile environments compared to their centralised counterpart [Mokhtar et al., 2008, Schuler et al., 2004, Zisman et al., 2013] that assumes the presence of a resource-rich, stable repository to cache all the providers. *Proactive discovery* allows providers to respond to a composition request by replying with their dynamic service specification to the system.

This thesis considers the process for locating providers from two perspectives: how to obtain dynamic service information for composition planning; and how to get current

[1] "Active search registers services in the local cache, and search takes place in all nodes by flooding the search message. Passive search registers their services in all nodes, and search takes place in only the local cache." [Mian et al., 2009]

service information in a timely manner when service re-composing is needed to recover a composite service from failures (Challenge C.3, C.4).

2.1.1 Reactive Discovery

DSDM [Artail et al., 2008] is a distributed service discovery model for MANETs. In DSDM, a set of devices are selected as service directories that store registered providers' information, including provider addresses, the description of service capabilities, and QoS values. Each directory keeps a copy of this information for every registered service provider. As a consequence, any of the registered service providers' information can be directly fetched by a service requester from the nearest directory. However, such duplicated directories makes the network's maintenance cost exponentially increases with a growth in the number and density of providers as all the directories have to be refreshed, which results in limited scalability.

Hierarchical service composition [Kalasapur et al., 2007] classifies service providers into four different levels according to their resources and computing capability. Co-located service providers are managed as a hierarchical overlay network. Resource-abundant providers are ranked as high-level and act as parent nodes in this hierarchical network. Parent nodes maintain their offspring nodes' I/O dependency as a graph, and each of the offspring nodes has limited resource and keeps a subset of the graph. Parent nodes have more system knowledge than their offspring. In this hierarchical model, a system looks for services in the available graph subsets in the service requester's close vicinity, and traverses the overlay network upwards if the nearby graph subsets have inadequate information to satisfy the service requirements. This model allows networked service providers to maintain a part of the system knowledge depending on their capability. Keeping system knowledge on each service provider up-to-date relies on receiving heartbeat messages from neighbouring service providers to identify their presence. However, updating this hierarchical overlay network may not only need to just modify a local service dependency graph but also to upload the graph to parent nodes, which increases network traffic.

Sadiq et al. [2014] facilitates service composition in MANETs with load-awareness and mobile-awareness. In this model, service providers publish their services to networks in their vicinity. Each participant service provider maintains a service I/O dependency

graph that includes information about neighbouring services providers, the current load of these providers and the temporal/physical distance between services. Based on the dependency graph, services with the shortest temporal distance will get selected to form a composite service. This model supports distributed global knowledge, by which a service composition system uses multiple service I/O dependency graphs stored in different service providers to resolve a composite request, but this has a propagation delay that may outdate service information.

2.1.2 Proactive Discovery

Prinz et al. [2008] relies on potential service providers to respond to service requests with their execution properties and to mark themselves as candidate providers. The request source selects the best performing service provider as the primary provider, and expects the other candidates to monitor the execution of the primary one. This model has no need for service re-discovery to recompose services for failure recovery. Once the primary provider fails, the second-best performing candidate will take on the execution, and be monitored by the rest of the candidates. This solution uses a Distributed Hash Table (DHT) to locate potential service providers, so it can prevent expensive directory maintenance. In a DHT-based service discovery system, when a service provider joins the system, it gets assigned a key (hashing value) that corresponds to its service functionality. The system hashes the key and appoints a node in the system as an index to cache the service provider's information. Functionally equivalent providers tend to be given the same hashing value, so their information is normally stored in the same index node. However, this can make a DHT-based service discovery system brittle in dynamic environments as the system will just lose the possibility of providing a functionality when the corresponding index node departs.

Recent research explores extensions to DHT-based service discovery systems to retain their lightweight approach to maintenance, while preventing functionality loss. Kang et al. [2008] introduces an optimal search that integrates a vicinity service indexes overlay into a DHT overlay network. It is not only the index node but also its neighbouring nodes that keep service provider information for a hash value. Pirrò et al. [2012] combine DHTs with a semantic overlay network, which takes services that have semantically

similar functionality to the requested one into consideration when searching for a service provider. It compromises the problem of functionality loss by expanding service discovery scope, but it comes at the cost of maintaining the semantic overlay network. Chord4S [He et al., 2013] distributes the descriptions of functionally equivalent services to varying directories to maintain continuously discoverable functionality in open P2P networks.

P2P-SDSD [Bianchini and Antonellis, 2008, Bianchini et al., 2010] enables dynamic service collaboration in P2P networks. It applies a probe-based service query, and a service request is submitted to service providers through previously established semantic links. Semantic links categorise service providers by connecting services that share similar functionality, and get updated when a service provider joins/leaves the category or enables/drops services. Service providers in this model cache the categories, based on which a group of providers for a specific functionality can be quickly located.

SSON [Al-Oqily and Karmouch, 2011] composes networked entities in a decentralized self-organizing manner and addresses seamless media delivery in pervasive environments. SSON requires no previously established discovery infrastructure. However, service providers are assumed to be aware of their geographic locations, and a service request is forwarded to providers' physical neighbours with a specific geographic angle[2]. In dynamic environments where devices' geographic locations can change quickly, frequently updating geographic location wastes local computation resources.

A cooperative discovery model [Furno and Zimeo, 2014] provides fully distributed support for unstructured P2P networks. With this model, a service provider receives a composition request and updates it by removing the parts of functionality that can be supported by the provider. Then, the service provider adds itself into a solution table, and forwards it along with the updated request to other providers. This process continues until all the required functionality are cleared and a set of service providers are recorded in the solution table. This model can be built over an existing overlay network (e.g. semantic overlay network or DHT) to improve search efficiency. Furno and Zimeo [2014] additionally proposes an overlay to further increase the search efficiency, which will be discussed in Section 2.2.3.

[2]"...a geographic angle [0, 180] determines the search scope between the media client and the media server"[Al-Oqily and Karmouch, 2011]

Opportunistic service composition [Groba and Clarke, 2014], similar to SSON, assumes no infrastructure for service discovery. This opportunistic composition model broadcasts composition requests and locates a service provider on-demand. It introduces a cross-layer service discovery that extends traditional broadcast protocols with one-hop acknowledgement messaging. With this cross-layer service discovery model, a service provider is aware of its neighbouring service topology. This topology information enables multicast-based service binding and unbinding which reduces network traffic.

2.1.3 Assessment

In general, a service discovery system can optimize run-time service search by using the previously cached service information (reactive discovery). This reduces communication overhead and discovery delay delay when locating service providers, but the system has to keep such service information up-to-date. Some approaches rely on service providers' advertisements to update service information, and if such advertisements are sporadic, the provider's information may out of date, and if they are too frequent, the energy cost of providers increases and in turn leads to a negative impact on providers' availability. Therefore, reactive service discovery is less efficient in terms of dynamic maintenance. However, service advertisement facilitates service composition in dynamic environments. A system may need to re-compose better services from the environment, including those that may have appeared even during service execution, aiming to recover from composition failures or improve overall service quality in real-time services provisioning. Thus, the system has to be context-aware. Service advertisement at runtime has the potential to cut down monitoring effort for a composition handler when detecting new, emerged services, and does not require to consider the maintenance problem.

Proactive service discovery gets the dynamic provider information directly from service providers, but may lead to high latency searching. Although there are overlay networks to improve the search speed, high latency searching still exists, especially when a composition requirement is complex. As service execution usually starts when every requested services in a composition has bound to a provider, a delay when searching for a service increases the potential that recently available providers may no longer be available. A complex composition requirement implies more required services. If a discovered provider has left the network before all the requested services are found, using

this provider's information to finish the remainder composition processes causes faults. Opportunistic service composition [Groba and Clarke, 2014] assumes a sub-service in a composite service can execute as long as its required data is provided regardless if all the sub-services are bound to their providers. It uses a service provider once it is located, which reduces the searching delay, but this approach is inflexible in other composition aspects like service composite adaptation that will be discussed in the following sections.

2.2 Request Routing

The target environment for this thesis consists of mobile providers connected in an ad hoc manner. Such an environment is multi-hop and infrastructure-less [Alomari and Sumari, 2008, Kozat and Tassiulas, 2004, Su and Guo, 2008], and a composition request needs to be routed to the networked entities that can resolve the request and handle service composition. Existing solutions are classified as controlled flooding, directory-based routing, and overlay-based routing.

Controlled flooding prevents broadcast-based requests flooding the network using imposed limitations, such as the maximum propagation limit for routing. Directory-based routing manages service providers' information in a set of directories. Depending on the size of the network and the density of available services, a central directory or multiple directories can be applied. This section discusses decentralized directories, and the question of how a request is routed in a network of directories. A centralised directory [Zhu et al., 2003] has obvious limitations in wireless environments as the scale of the network is restricted to the directory carrier devices' wireless communication range. Overlay-based routing allows service providers to link to each other through a particular connection (e.g., semantic similarity, dependency, etc.) between them. Thereafter, a service request can be transmitted via these links. This greatly reduces network traffic compared to limited flooding. Some overlay-based routing approaches also employ directories to increase search success or search efficiency. They are classified as hybrid routing and discussed in Section 2.2.4. Request routing approaches are analysed with regard to the question of how to find service providers in a timely fashion, and the cost of the discovery process (Challenge C.2, C.5).

2.2.1 Controlled Flooding

GSD [Chakraborty and Joshi, 2002, 2006] is a distributed service discovery model in pervasive computing environments. Service providers assign a service-group to each of the services they provide, according to service functionality, using a domain-specific tree-like ontology. They advertise their services and corresponding service-groups to their vicinity networks. The service information is cached only by their direct neighbouring nodes, and the service-groups are stored by all the nodes in the vicinity. Such cached service-groups can promote service search, as a node can quickly know if the required functionality can be supported by its local network according to the cached service-groups. If a service request matches a cached service or a cached service-group, the request can be directly routed to the service provider. If there is no matched service-group, the request gets forwarded to the rest of the network. Moreover, to prevent a request flooding the network, a hop-count parameter is used in service requests, which is initialised by a request source and indicates the maximum number of request forwarding permitted. A web service discovery model [del Val et al., 2014] for dynamic environments employs a Time To Live (TTL) parameter to limit request flooding by messages' transmission time. This is more flexible than GSD, but it only considers the transmission cost for discovering a single service.

Opportunistic service composition [Groba and Clarke, 2014], as mentioned in Section 2.1.2, is based on broadcast for service discovery, but it is not like GSD that uses service advertisement to previously cache service information and build directory networks. Instead, opportunistic composition employs *acknowledge messages* to collect local network's topology information while routing request messages. A composition requester can know its communication routes to all the candidate service providers through receiving an acknowledge message from them. The composition requester uses the routes information to release[3] the candidate service providers that will not be executed. However, there is still a risk of flooding the network with request messages and additional communication overhead is introduced by involving acknowledge messages.

[3]Opportunistic service composition requires service providers to lock their resources for a composition as long as they have received the request of the composition and have committed themselves as candidate providers.

2.2.2 Directory-Based

Kozat and Tassiulas [2004] distribute a service directory, which achieves scalable, efficient service discovery in MANETs. This work introduces a virtual backbone layer that organizes service directories as a P2P overlay network and builds it above the general service overlays. If a provider registers its service to the network, its registration message will be forwarded to every directory in a virtual backbone layer. In other words, a service specification is duplicated and cached by different directories. Given that MANETs are dynamic, the service information kept in the virtual backbone layer must be renewed when new providers are registered. The virtual backbone layer uses asynchronous updating, which means directories in the layer are asynchronously renewed by spreading registration messages. A service discovery process depending on such a virtual backbone layer requires a service request to flood the backbone layer until a matching service provider is found. Updating directories has to keep up with network changes. However, if the network frequently changes, keeping the whole directory-based virtual backbone up-to-date can be extremely expensive.

Likewise, Hexell [Tyan and Mahmoud, 2005] applies networked directories to maintain a virtual backbone layer, but without requiring all the directories to keep a copy of any individual service specification. Hexell divides an environment into hexagon cells, the size of which is determined by wireless devices' communication ranges. The service provider that is closest to the centre of a cell is selected as a directory. A selected directory caches information about the service providers that are currently in its responsibility cell. Similar to GSD [Chakraborty and Joshi, 2002, 2006], Hexell groups services using a hierarchy ontology, and organizes services according their groups in a directory. A directory updates and informs the rest of the directories about available groups in its responsibility cell. As a result, a directory located in another cell knows what functionality can be extracted from service providers in each cell through the directory-based virtual backbone layer, and can support global service query for service requesters in its own responsibility cell. This group-based mechanism allows Hexell to update the directory layer potentially less frequently than the network changes, because there is no need for a directory to update the directory layer, if a new service is functionally similar to a service that has already registered to the directory. Hexell's hexagon cells also

provide a service discovery system with a clear view of the communication distance between service providers through the number of cells a communication crossed, by which a communication-efficient composite of services may be selected. However, a directory node is mobile, and when it is roaming, it is likely to leave the centre of its responsibility cell or even move to another cell. Hexell provides no clarification on how this will affect the service discovery process and how to manage it.

VSD [Kim et al., 2006] encourages resource-rich devices to volunteer for directory duty. Directories are isolated in VSD. They are known mutually only when they have co-registered services. A link is built between directories when they know each other's presence, and service requests are forwarded to directories based on such links. VSD also has the issue of directory loss due to mobility. Similar to VSD, Kalasapur et al. [2007] considers resource-rich devices as directories. As described in Section 2.1.1, directories are maintained in a hierarchical way, in which service discovery requests travel bottom-up to find a service composite. However, resource-rich nodes are not always possible in mobile environments, which makes VSD [Kim et al., 2006] and the hierarchical directory solution Kalasapur et al. [2007] feasible only in some particular scenarios.

2.2.3 Overlay-Based

Using Distributed Hash Table (DHT)-based overlay networks (see Section 2.1.2) for service discovery has been investigated in pervasive computing. In a DHT-based overlay network, a hash value is calculated for the functionality requested. According to this hash value, a destination to route the request is determined. The destination is a node that previously cached information about the service providers that support the requested functionality. The destination node receives the request and forwards it to these service providers using multi-casting. However, as analysed in Section 2.1.2, DHT-based overlay networks store information about functionally equivalent providers in the same node, so functionality loss may occur if a directory departs or crashes. In addition, even though DHT-based overlay networks prevent service requests flooding a service providers' network, it can end up with a long routing path, which in turn delays the request to be routed to potential service providers. Kang et al. [2008] integrates service caches to a DHT overlay network, which removes a part of a redundant request route on

the physical network layer, and increases time-efficiency for DHT-based service discovery. But this comes at a cost of maintaining service caches.

SON [Crespo and Garcia-Molina, 2005] has been proposed for content-based search optimizing in P2P networks. The core notion of a SON is to bunch similar peers, meaning query processes can discover bunches instead of individual peers for faster locating. P2P-SDSD [Bianchini and Antonellis, 2008, Bianchini et al., 2010] introduces inter-peer semantic links into SON, which supports fine-grained (i.e., not a composite) functionality to be provided and more possible compositions of services. P2P-SDSD supports two policies for request routing: minimal and exhaustive. With the minimal policy, a service request is forwarded through semantic links in a SON until a matching service provider is located for the request. The exhaustive policy is designed for finding a service provider with the optimal non-functional features (e.g., reliability) among a set of candidate providers. To achieve this, a system stops request routing only when all the candidate providers are discovered. In short, the minimal policy is efficient, but cannot find an optimal result, while the exhaustive policy may support the composition of services with good QoS, but it needs to route a request to all the potential providers, which is inefficient.

A cooperative discovery model [Furno and Zimeo, 2013, 2014] uses historical composition processes to construct a tree-like overlay network to increase the efficiency of service discovery. The overlay network is formulated by networked super-peers. For each historical composition process, one super-peer is elected among the participant service providers to cache these participants' information and the composite service's functionality. Thereafter, this super-peer responds quickly to service requests with similar functionality to the composite service it maintains. If a request can be solved by a number of different super-peers, yet another super-peer is elected which maintains links to these ones. Thus, a tree-like overlay network is built over time, with super-peers as branches. By referring to super-peers, normal peers can locate provider compositions for their service requests if these requests have been resolved (or at least partly resolved) in the network. Specifically, a service discovery process will firstly examine super-peers and then probe the normal service providers if the requested service cannot be fully found in the super-peers. However, it assumes the network has super-peers 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 capability of the planning solution to cope with dynamic adaptability, such as dynamic re-planning.

2.2.4 Assessment

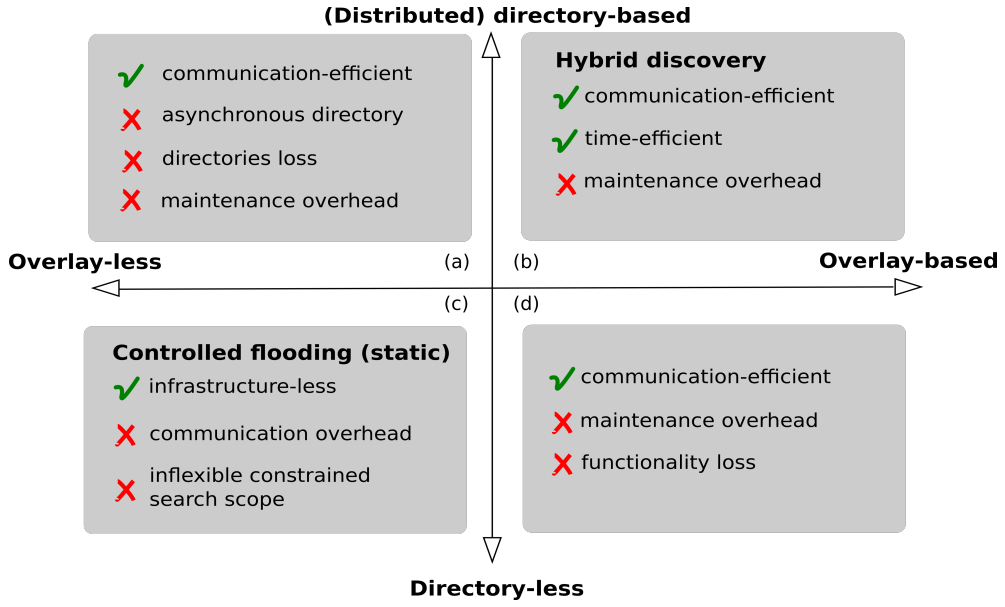


FIGURE 2.2: Analysis on composition request routing solutions: *(a) distributed directory-based routing, (b) hybrid routing, (c) static controlled routing, (d) overlay-based routing.*

Request routing solutions include (static) controlled flooding, distributed directories, and overlays. Hybrid routing approaches that integrate service caches/directories into a overlay network [Furno and Zimeo, 2014, Kang et al., 2008] are most efficient in terms of network communications for service discovery. However, as shown in Figure 2.2(b), existing hybrid routing solutions relying on service advertisement [Kang et al., 2008] or the previous composition records [Furno and Zimeo, 2014] to maintain the discovery infrastructure. The former requires frequent service announcement for mobile service providers and overlay maintenance cost, and the later may lead a composition request to an inaccurate destination. Directory-based solutions (Figure 2.2(a)) and overlay-based solutions (Figure 2.2(d)) promote search efficient with short routing paths or less network traffic, but they also introduce maintenance cost of their support infrastructures. Optionally, the maintenance cost can be reduced by controlled flooding. Controlled flooding-based routing (Figure 2.2(c)) is infrastructure-less, and so has the potential to

support highly dynamic environments. However, existing approaches rely on static flood controlling to route a request, which is inflexible and creates unnecessary communication. There is a need for a more flexible, infrastructure-less, dynamic controlled request routing to support service discovery in the target environment.

2.3 Composition Planning

Existing service composition techniques investigate dynamic composition planning mechanisms to reduce composition and execution failures while dealing with complex user requirements. Examples of such investigations applied to pervasive computing include open service discovery approaches, and goal-driven planning approaches, both of which automatically discover a combination of multiple services to support a user goal or a functionality when a single matched service is unavailable. They are different in specifying composition plan. Composition plan can be specified by low-level requested service description[4], high-level requested service description[5], and workflow-provided service description[6][Raychoudhury et al., 2013]. Open service discovery use workflow-provided service description while the goal-driven planning relies on high-level requested service description. They are flexible compared to those that rely on a predefined conceptual composite to find services that can exactly match its service requirement (i.e., low-level requested service description) [Raychoudhury et al., 2013]. This section discusses how flexible these planning approaches are, and how to self-organize the planning process (Challenge C.1, C.2).

2.3.1 Open Service Discovery

A graph-based service aggregation method [Zhenghui et al., 2009] targets flexible planning in pervasive computing environments by modelling services and their I/O parameters in an aggregation graph based on parameter dependence in the services. Complex user requirements in this method are resolved according to predefined conceptual composites. The approach dynamically composes services to finalize an abstract service in a

[4] "... the requested service is specified as a workflow, given the set of atomic services to be composed"[Raychoudhury et al., 2013]
[5] "... the requested service is specified as a goal to be achieved."[Raychoudhury et al., 2013]
[6] "... it is assumed that service providers are able to specify workflows in which they can take part."[Raychoudhury et al., 2013]

conceptual composite when no single service can support it independently. A composite service is found if the aggregation graph contains a path to link the abstract service's output parameter to its input parameter. Similarly, a dependency graph [Rodriguez-mier et al., 2012] was used for service aggregation, but its usage differs from that in [Zhenghui et al., 2009] since it directly maintains service dependency relations rather than their I/O parameters' relations. An open workflow [Thomas et al., 2009] has been proposed to support service composition in MANETs. It models workflows that already exist in the environment as a supergraph, and discovers services through the data flow in the supergraph. Liu et al. [2015a] proposed a parallel approach for service composition in pervasive environments that relies on function graphs to model complex composition requirements. This approach resolves a parallel function by simplifying its split-join logic, breaking the parallel branches. Specifically, it decomposes the corresponding parallel function graph as a set of sub-functions that require sequential service flows, and then searches for services to satisfy each sub-function. However, the above approaches require central entities or clusters[7] 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 [Al-Oqily and Karmouch, 2011], 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, as discussed in Section 2.1.2, assumes 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.

[7] A cluster collects service information locally.

2.3.2 Goal-Oriented

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 [Oh et al., 2008] proposes a novel AI planning-based algorithm for large-scale Web services. This work is based on an analysis of complex networks. The EQSQL-based planning algorithm [Ren et al., 2011] applied rank-based models to promote service composition efficiency. Ukey et al. [2010] modeled 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 that combines a forward-chaining approach and a backward-chaining approach to find a path with the smallest cost from the dependency graph. Liu et al. [2015b] also proposed a bidirectional planning algorithm. It introduced tag-based semantics to specify composition goals, addressing effective service query. Khakhkhar et al. [2012] improved bidirectional planning and 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 [Hibner and Zielinski, 2007] 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 [Gharzouli and Boufaida, 2011] 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 [Jun et al., 2010] [Dai and Wu, 2004]. Geyik et al. [2013] propose a distributed implementation for their composition planning model. This approach uses a backward search scheme in a wireless sensor network to generate a service dependency graph that includes all the possible service paths from the services that can directly provide the requested output to every sensor node that does not require any input data. Each service provider in

the graph only maintains links to its neighbouring services. After that, an composition graph can be generated by forwarding messages through the dependency links in the network. Although this approach supports distributed planning, it does not state that how a data parallel task is resolved, and it invokes services only after all the potential service providers are discovered, which can delay the composition process and cause communication overhead. Similarly, a cooperative discovery model [Furno and Zimeo, 2013, 2014] provides fully distributed support for unstructured P2P networks. This work also included a bi-direction search model 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 information, which implies resource-rich devices.

2.3.3 Assessment

Composition planning uses service availability information to reason about an executable service workflow. Table 2.1 discusses service composition solutions that use open service discovery or goal-oriented service discovery, and illustrates how these solutions meet the flexibility criteria. Generally, open and dynamic environments increases the possibility of diverse service flows being built for a single composition request. Goal-driven service planning is more flexible as it composes whatever usable services in the environment instead of using a pre-defined abstract composite to find only matched services. However most goal-driven solutions still require a centralized planning engine and a service repository to support complex service workflows, which may plans faster, but any service information change during service execution is likely to make the primary planning result unreliable. Updating the service repository when changes occur may be a solution, but in a highly dynamic scenario, frequent updates on the service repository are expensive. Decentralized solutions tackle these problems by partitioning the planning process to allow local service providers or a set of brokers to collaboratively solve a composition goal. Unfortunately existing decentralized goal-driven service composition approaches are not flexible enough to support parallel service flows.

2.4 Service Binding

A service composition model selects a group of services and binds their resources for invocation. To tackle dynamic environments, such a service selection could benefit from predictions for services' availability or service execution paths' strength (reliability). It also worth considering services' Quality of Service (QoS) attributes of services. On-demand binding mechanisms are investigated here to address dynamism in another aspect, that is, to keep service selection flexible until a service has to be selected for immediate use. The analysis in this section includes consideration of how to select service providers and when to lock these service providers' resources to maximize the possibility of successful service invocation, while not impacting the possibility for, candidate providers to be selected by other compositions (Challenge C.3, C.5). Note that approaches that assume random selection at an early stage of composition [Bianchini and Antonellis, 2008, Bianchini et al., 2010, Ridhawi and Karmouch, 2015], and those assume providers follow a schedule to act [Sen et al., 2008] are not considered in this assessment.

2.4.1 QoS-Based Selection

Mokhtar and Liu [2005] take non-functional properties of services into consideration when selecting service providers in pervasive computing environments. In this work, service providers predict their own QoS before service execution, and enclose this QoS information into their service advertisement messages. The approach uses a set of QoS metrics to quantify QoS information, which includes a probability value for service availability and a value that indicates execution latency. This work uses probability values to define service availability, which only caters for when a service is withdrawn by providers, and not when the provider moves out of range. de Medeiros et al. [2015] predict the quality of entire service composition instead of individual service provider. They model service compositions' cost and reliability, defining 6 cost behaviours and 4 workflow reduction models for different service flows. However, they do not consider the reliability and the cost of communication among successive services.

HOSSON [Li et al., 2007] maintains a user perspective service overlay network (PSON)

for all the candidate service providers. The PSON is graph-based, and performs service selection using multiple criteria decision making to find the optimal service execution path with minimal service execution cost and good reputation[8]. Zhenghui et al. [2009] also considers the quality of overall service execution paths when selecting service providers. They adopt path measurement including reliability, execution latency and network conditions. However, the above approaches assume the QoS information is predefined by service providers, and as network conditions are likely to change, this assumption is not safe in a dynamic environment.

Zhou et al. [2011] select services at an early stage in a service discovery process, depending on the strength of service links. During service advertisement, if a service provider announces its service specification, a neighbouring node caches the service specification and also maintains information relating to the strength of the path to the service provider. When a node receives a service request, assuming information about multiple functionally equivalent service providers that match the request is cached, the node selects the with the strongest path to itself. When a selected destination is unreachable, a backtracking mechanism can be performed to recover service binding. Though path strengths are involved in binding decision making, and a binding recovery mechanism is proposed for failed routing, this approach is still unsafe as beforehand cached path strength information may be out of date, and the backtracking based recovery is expensive itself. Surrogate Models are used to facilitate service composition in mobile ad hoc networks [Efstathiou et al., 2014b].

2.4.2 Adaptable Binding

ProAdapt [Aschoff and Zisman, 2011] monitors operating environments and adapts service composites to a list of context changes, including response time changes, availability of services and availability of service providers. FTSSF [Silas et al., 2012] applies a monitoring and fault handling process for service provisioning in pervasive computing. Monitoring is concurrent with a service delivery process, allowing for service re-selection if the execution crashes. [Prinz et al., 2008] allocate a service provider to the best performing candidate in terms of the QoS while keeping a group of backup providers. A backup provider can replace a previously bound provider if its execution fails. This

[8]Reputation is a ranking value that is given by previous service users to represent a service's dependability. [Li et al., 2007]

model is more efficient than ProAdapt and FTSSF, as it monitors the service composite only when it is executing, which means there is no need for a service composition system to continuously monitor the operating system throughout the composition process.

Wang et al. [2013] models service composition as a problem of finding a service provider for each abstract service in a predefined conceptual composite. Service binding adaptation in this approach is based on automatic QoS prediction. Specifically, a service composition system firstly allocates a set of services to form an execution path, the QoS of which conforms to the QoS constraints that are defined in the composition request. The system then predicts the failure probability of the path, and finally finalizes the service composite according to the prediction result. If necessary, re-selection for providers can be performed. This solution regards the service execution path's reliability as an important criterion for service selection and allocation instead of considering each of the service providers independently. However, the approach relies on a centralized composition handler to perform its prediction.

2.4.3 On-Demand Binding

OSIRIS [Schuler et al., 2004] selects service providers on demand at execution time. This selection of service providers depends on run-time device load and dynamic invocation cost. Service discovery is performed offline to find candidate service providers, but repeated to detect new service providers. However, the approach depends on a central repository to store service specifications, which is limited in dynamic systems.

Opportunistic service composition [Groba and Clarke, 2011, 2012, 2014] also proposes an on-demand service binding mechanism. Unlike OSIRIS, opportunistic service composition does not use central repositories to keep service specification. Instead, it discovers service providers on-the-fly relying on request flooding. After service providers are located, the approach asks for permissions to lock service providers resources, and then invokes the service. This model ensures an available service provider will be invoked for execution, but refinements are required to further consider result routing when binding a service provider, at the same time reducing the cost of its flooding-based service discovery and increasing the flexibility of composition planning.

2.4.4 Assessment

QoS-based service binding addresses dynamic environments by self-describing services' runtime properties (e.g., availability, reliability, response time, etc.). Existing approaches Li et al. [2007], Mokhtar and Liu [2005], Zhenghui et al. [2009], Zhou et al. [2011] depend on QoS descriptions provided by service providers that predict their own service performance. Although this mechanism is lightweight for a service composition system as no monitoring effort is required, such QoS descriptions are likely to be inaccurate. Moreover, mobile service providers may have to frequently update their QoS description. Adaptable binding detects changes on operating environments and adapts a service composite accordingly. Detecting changes or failures requires different levels of monitoring. Execution time monitoring is the most efficient, but failures can only be detected after they occur. Recovering a composition from emerged failures can introduce additional time and communication cost. On-demand binding selects service providers using up-to-date service information, requiring no extra environment monitoring or infrastructure maintaining efforts. It would be interesting to improve the existing on-demand service binding mechanisms to support reliable service composition, and at the same time, allowing the bound service smoothly and seamlessly adapt to context changes.

2.5 Service Invocation

This section analyses how service execution processes manage mutable environments. Centralized service invocation has been studied in many solutions [Miraoui et al., 2011, Mokhtar and Liu, 2005, Sousa et al., 2006, Vukovi and Robinson, 2007], however, supervising service providers' execution with a central entity limits mobile-awareness, and consequently is less flexible. Research on distributed service execution includes the following solutions: Fragments distribution approaches partition a service composite and distribute each of the abstract services in the composite as a task to selected service providers. Service composition is managed in a decentralized way by these selected entities. Process migration approaches only partition the service execution process, and issue a full composite to the first provider (or a primal networked broker). The composite is partially resolved by the first provider and the rest of the composite is handed over to the subsequent providers (or backup networked brokers). This analysis of service

invocation focuses the question of how a service composition model self-organizes the composition process (Challenge C.4).

2.5.1 Fragments Distribution

As described in Section 2.1.2, Prinz et al. [2008] distributes the execution process to a set of selected service providers, and maintains references to the other candidate providers as backups. The execution of selected providers is monitored by backup providers. Once a selected provider fails, one of the backup provider takes over the execution process. This approach achieves flexible decentralized service execution, but its service discovery and selection process still require a central composition engine.

Fdhila et al. [2009] target service compositions that contain complex control logic, which is decomposed using a dependency table. In particular, the composition model slices a conceptual service composite into a set of sub-processes, each of which is realized by only one service instance. These sub-processes are allocated to associated service providers, and executed independently. However, this model assumes a global knowledge of available service providers, which is difficult in a dynamic environment. OSIRIS [Schuler et al., 2004] decomposes a central conceptual composite description into a set of execution fragments, and distributes these fragments to different service providers. It also supports complex control logic, but assumes a continuously available node to manage a parallel service execution.

2.5.2 Process Migration Approaches

Chakraborty et al. [2005] propose distributed broker-based model for service composition in MANETs. The approach dynamically selects a group of networked entities in a MANET, which relay a composition process from one to another. A requester initially assigns a composition task to one broker. This broker performs service composition until, potentially, a failure occurs. If the composition stops at this broker because of a failure, its current state and any intermediate execution results are frozen and handed over to another broker that executes the remaining process. The new broker's address is sent to the requester. Selecting brokers and creating a network with the selected brokers are key to this approach. Anything like brokers' mobility or reliability can affect the

service invocation process. However, it is difficult to create a good broker network in the target environment where a global view or a central controller is impossible.

WFMS [Atluri et al., 2007] is a decentralized workflow management model that self-describes a workflow and relays the workflow from one service provider to its subsequent service providers, resolving the workflow partially on each participating provider. Similarly, continuation-passing messaging [Yu, 2009] is a decentralized execution model that passes a service execution process from the primal service provider to the rest of providers. This model selects a node in the network to handle execution faults, named the scope manager. When a service provider detects a failure, it sends the failure information to the scope manager. The scope manager receives the failure information and re-executes services. Service execution in this model is fully decentralized, however, the failure recovery is limited by the resource and communication range of the scope manager.

A process instance migration approach [Zaplata and Hamann, 2010] allocates the full control logic of a conceptual composite to every participating service provider. It allows participants to keep composition information so that each participant can select a granularity of fragmentation at runtime. The approach is required to be further improved to support a decentralized composition adaptation model.

2.5.3 Assessment

Process migration allows service providers to know a part of a request, and gives them the chance to alter the part of the process they know by using their knowledge about the local networks. Fragments distribution decouples an execution process into execution fragments and allocates them to corresponding service providers. Fragments distribution minimizes process duplication, but requires a central entity to be responsible for service discovery and allocation. Both of the methods resolve service invocation in a decentralized manor, which is suitable for ad hoc networks. To achieve success execution and protect run-time data flow, decentralized service invocation needs to efficiently adapt to environment changes, and prevent the composition process from exposing the full data flow. In terms of dynamic adaptation, process migration is more flexible than the fragments distribution approaches when supporting functional service replacement. This means when a functionality is no longer required before its matched services get

invoked, a process migration manager will have the potential to remove the functionality from the composition before it hands over the process to the subsequent manager, while a service provider in a fragment distribution approach can only replace a failed service by another that supports the same functionality. On the other hand, migrating a composition process from one service provider to another may reveal the composition's control logic and data flow to those providers. Generally, process migration is more flexible to realise dynamic adaptation, and fragments distribution can protect the overall dataflow from being observed by a single third-party.

2.6 Fault Tolerance

This thesis classifies fault tolerance mechanisms in terms of how to deal with the potential failure and actual failure throughout the composition process. Preventive adaptation focuses on anticipating potential failures and taking actions to prevent them. Composition recovery should have a fast, communication-efficient model to recover a faulty composition, without disruption. This section discusses how to detect and handle failures in a timely fashion, and if the additional cost for fault tolerance is affordable for mobile and pervasive environments (Challenge C.3, C.4, C.5).

2.6.1 Preventive Adaptation

The OSIRIS approach [Schuler et al., 2004] decomposes a central service flow description into 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 another. 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. The approach provides no means to allow a potentially new provider that may appear in the network to participate in the composition. MAIS [Ardagna and Pernici, 2007] is an adaptive service composition model for web services. It reduces service invocation failures by negotiating QoS in advance of composition selection and

execution. Although MAIS provides strategies to decrease negotiation overhead at runtime, its negotiation uses previously cached information about services and QoS which is likely to be changed in a dynamic environment.

A fuzzy-based service composition approach [Prochart et al., 2007] has been introduced for MANETs, which considers resource-constrained devices and error-prone wireless communication channels. Each node maintains its neighbours' service information and gets real-time QoS information during service discovery. A fuzzy TOPSIS method [Cheng et al., 2011], unlike Prochart et al. [2007]'s approach that has no support for adapting composite services at runtime, adapts service bindings according to real-time user preferences. TOPSIS can prevent unnecessary adaptation through fuzzy analysis, but it has no support for adaptation triggered by operating environment changes. A distributed dynamic composition model [Geyik et al., 2013] allows a service provider participating in the composition graph to notify its successive services' providers when it changes its own service information, and raises the potential of adapting a service composite before an invocation failure occurs. This model states a way to find an alternative service from a distributed service dependency graph. However, it is not clear that how this can be achieved during the service composite's execution.

2.6.2 Composition Recovery

Yu [2009] uses acknowledge messages to monitor service execution. They also allow a service execution process to retry a service invocation after it fails. Although this policy targets unreliable wireless channels, giving a service provider a second chance to offer its services, it has limited and inflexible support for composition recovery. Prinz et al. [2008] provide a recovery policy which is more flexible than the one proposed by Yu [2009]. Instead of retrying failed invocations, Prinz et al. [2008] keep candidate service providers during a service execution, allowing them to monitor the primary service provider's execution. When the primary one fails, one of the candidate service providers takes over the service provisioning process. However, this policy requires primary service providers to push heartbeat messages to backup providers to indicate its execution status. If a service's execution has high latency, pushing heartbeat messages can quickly exhaust its providers' battery, and in turn reduce the availability of the service.

TLPlan [Vukovi and Robinson, 2007] allows a system to plan an abstract solution from a pre-existing abstract service repository, and then to discover and bind services for execution. In each step of the service composition, TLPlan provides on-the-fly rollback mechanisms to deal with potential faults like composition failures, service discovery failures, or service execution failures. TLPlan, however, plans for a composition offline, relies on central composition engines that have not yet been applied on mobile devices, and re-generates a new plan when a composition recovery is required, which is time-consuming and not suitable for dynamic environments.

A minimum disruption service composition model [Jiang et al., 2009] investigates the types of composition failures in MANETs, and provides network-level and service-level adaptation, as well as recovery mechanisms. During service execution, the approach quantitatively estimates the one-hop forward service execution path's lifetime according to the distance to the next service provider, and composes a new service path if the next service provider is missing.

A cache-based service execution and recovery model Zhou et al. [2011] for MANETs caches backup services. The approach adopts a notion of a magnetic field to underpin an adaptation policy for cached service information in the vicinity. A service provider initializes a magnetic strength value that determines the border of the vicinity, which indicates the maximum transmission hops for its advertisement, and counts down by one for each message relay hop. A service provider periodically advertises its services, and its physical neighbours cache the provider's functionality and a magnetic strength value. The closest neighbour will rank this service provider the highest. During service discovery, if a service request reaches one of its neighbours, the neighbour can route the request to the service provider, and if there are multiple service providers cached by the neighbour, according to their magnetic strength values, the service provider with the minimal transmission hops is selected for request routing. A backtrack strategy is applied when a selected service provider has moved away before the request has routed to it. A fail message is routed backward via the routing mediators towards the request source. If one of the routing mediators has cached another service provider that supports the same functionality as the missing one, the new service provider is selected and the request is re-routed from this routing mediator. This backtracking method allows for a short recovery path and less communication overhead, but it requires frequent updates

to the cached service provider information if the network topology changes quickly, and it is unclear how the backtracking works if a routing mediator leaves the network.

2.6.3 Assessment

Preventive adaptation is timely as a service execution will not be interrupted by failures and recovering processes. However, existing solutions rely on monitoring infrastructures or central decision-makers to determine adaptation actions, which is infeasible in our target environment. Composition recovery aims to explore efficient model to recover a faulted composition. Current solution includes keeping backup service providers [Prinz et al., 2008, Zhou et al., 2011] for quick replacement, dynamic path re-generation [Jiang et al., 2009], and rollback composition [Vukovi and Robinson, 2007]. Using backup service providers is efficient but inflexible because the backup service providers are previously found and may become unavailable at runtime. Rollback composition re-plans for composition, which is flexible. However, re-planning on the fly is likely to be time-consuming and delays the composition result. A new fault tolerance model is needed to dynamically re-planning for (a part of) execution path, without affecting currently executing services.

2.7 Summary

This chapter reviewed the state of the art of composition process management and fault tolerance from the perspective of openness and dynamism. The most related solutions include efficient service discovery [Kang et al., 2008, Pirrò et al., 2012], flexible planning [Al-Oqily and Karmouch, 2011, Furno and Zimeo, 2014, Kalasapur et al., 2007, Rodriguez-mier et al., 2012], dynamic binding [Groba and Clarke, 2011, 2012, 2014], decentralized composition management [Furno and Zimeo, 2014, Groba and Clarke, 2014, Prinz et al., 2008, Sen et al., 2008] and dynamic fault tolerance [Gu and Nahrstedt, 2006, Hibner and Zielinski, 2007, Jiang et al., 2007, Prinz et al., 2008, Zhou et al., 2011].

Figure 2.3 illustrates the extent to which these approaches satisfy the set of criteria listed to the right of the Figure. The criteria are defined from the requirements (Section 1.5) outlined for service provision in pervasive computing environments. As illustrated in Figure 2.3, most planning-based service composition approaches rely on discovery

infrastructures and a-prior system knowledge [Al-Oqily and Karmouch, 2011, Kalasapur et al., 2007, Rodriguez-mier et al., 2012]. Real time information about services should be used for composition planning. Decentralized composition management solutions are restricted to exact matching functionality or planning for a sequential service composite [Furno and Zimeo, 2014, Groba and Clarke, 2014, Prinz et al., 2008]. *There is a lack of a more flexible decentralized planning model that can cope with data-parallel tasks.* Fault tolerance either requires composition re-planning that is time-consuming [Vukovi and Robinson, 2007], or only replaces failed providers with backup ones without considering the reliability of the new execution path [Gu and Nahrstedt, 2006, Hibner and Zielinski, 2007, Jiang et al., 2007, Prinz et al., 2008, Zhou et al., 2011]. *A dynamic composite service adaptation model is required, which should be aware of dynamic context changes, and depends on the end-to-end QoS to select backup paths during adaptation.* In summary, open gaps with the current service composition solutions are:

1) They are neither flexible nor lightweight enough to compose services in dynamic pervasive environments. Mobile service providers need to self-organise a service composition process which can flexibly use local service knowledge to plan for a global composition result. The process of composing relevant services is itself likely to be time-consuming, and so models are required to streamline or otherwise ensure that the service composition can be done within time constraints.

2) They support limited adaptation for service composites. An appropriate granularity of adaptation model is essential for a service provision model to quickly adapt a service composite while without introducing heavyweight adaptation model.

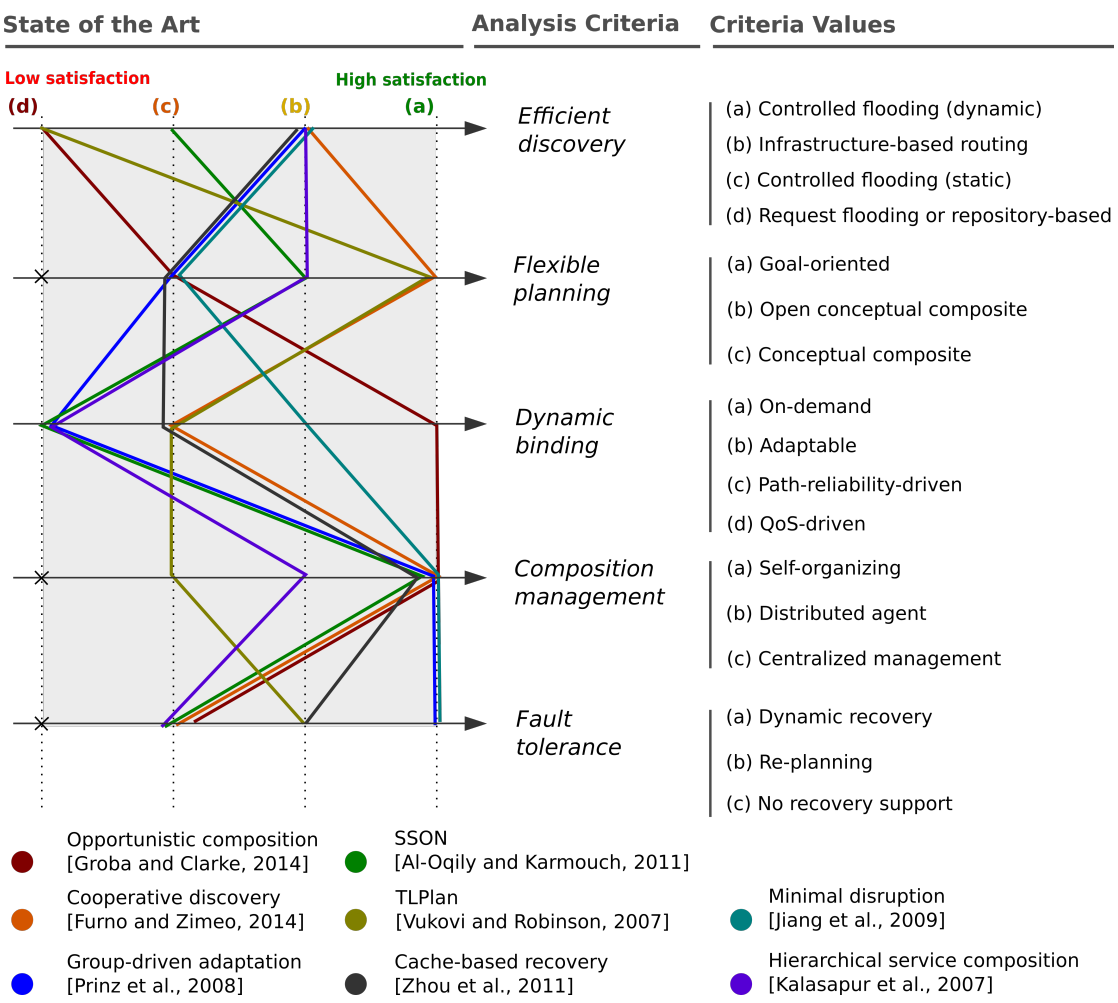


FIGURE 2.3: The state of the art review shows how most related solutions address five reference criteria.

TABLE 2.1: Literature review for composition planning models. The highlighted columns represent the required features regarding a model's flexibility.

	Criteria						
	specification		**service flow**		**management**		
Approach	open discovery	goal-driven	sequential	parallel	centralised	decentralised
Zhenghui et al., 2009	⋆				⋆	
Rodriguezmieret al., 2012	⋆				⋆	
Thomas et al., 2009	⋆		⋆	⋆		
Liu et al., 2015a	⋆					
Al-Oqily and Karmouch, 2011	⋆		⋆			⋆
Oh et al., 2008		⋆	⋆		⋆	
Ren et al., 2011		⋆	⋆		⋆	
Ukey et al., 2010		⋆	⋆		⋆	
Liu et al., 2015b		⋆		⋆	⋆	
Khakhkhar et al.,2012		⋆	⋆		⋆	
Hibner and Zielinski, 2007		⋆	⋆		⋆	
Gharzouli and Boufaida, 2011		⋆	⋆			⋆
Greyik et al.,2013		⋆	⋆		⋆	⋆
Furno and Zimeo,2013, 2014		⋆	⋆			⋆
Required		⋆	⋆	⋆		⋆

Chapter 3

Design

The review of the state of the art, presented in the previous chapter, has defined a number of limitations in current service composition approaches in the domain of open, dynamic pervasive computing environments. Open issues with current research on service composition are that i) existing service composition models are not sufficiently flexible to cope with mobile service providers; and ii) the composition models have limited support for handling composite service adaptation. This chapter introduces a service composition solution named **Go**al-driven service **Co**mposition in **Mo**bile and pervasive environments, GoCoMo for short. It begins by discussing the design objectives of GoCoMo, from which a list of requirements is presented. This chapter then presents the design of GoCoMo, and shows how the design addresses the requirements. It continues with a description of the system model and GoCoMo composition algorithms. Finally, it describes an in-depth discussion of the solution and the contribution.

3.1 Design Objectives and Required Features

As described in Chapter 1, the requirement of this thesis is two fold: to design and build a flexible decentralized service composition model that is time-efficient and infrastructureless, and to design and build a service composite adaptation model which adapts the combination of services as appropriate to the service provider's and the environment's changing situation and timeliness constraints.

To meet these requirements and the thesis challenge, this work aims to build a service composition model that supports the following required features, mapped to the challenges outlined in Chapter 1.

Feature 1: **Dynamic composition planning**

In a dynamic environment with no conceptual composite (Challenge C.1), a service composition model must appropriately decouple user requests to satisfy required functionality with the simplest possible composition result.

Feature 2: **Self-organizing**

Networked entities are likely to have only limited system knowledge (Challenge C.2), and so service providers must use local knowledge to self-organize a composition planning and execution process.

Feature 3: **Minimal failure recovery delay**

Composition latency is an important factor that affects the success of service provisioning (Challenge C.3), and can be affected by dynamic nature of the service provider network, which could lead to provider failures. Fault tolerance is required for dynamic environments, which handles faults that (possibly) emerge during service composition by replacing (potential) failed service providers or a fragment of the service execution path which is invalid. A lightweight and time-efficient mechanism is required to recover a service composite from failures.

Feature 4: **Locality-driven selection**

Service providers are connected in an ad hoc manner, depending on mediators to relay messages. With more mediators for a logical service link, more message forwarding will be required, and such a link may be error-prone because of unstable wireless channels (Challenge C.4). A service composition model should reduce dependency on wireless transmission and the number of message routing hops.

Feature 5: **Short standby time**

Wireless links are dynamic, which may affect an established service composite (Challenge C.5). In particular, the longer a bound service provider has to wait for execution, a disruption to the link to invoke the service provider is more likely to occur. This work's goal is to design a service composition model that always uses

current service information to support service selection, and minimizes standby duration for selected service providers.

3.2 System Model

This thesis focuses on service composition that is supported by mobile devices in pervasive environments. This work assumes open and dynamic networks, where mobile devices offer services that can be composed to create value-added functionality. Mobile devices are independent, and owned by heterogeneous third-parties. In other words, they do not follow any authority in the environment requiring them to schedule services for service requesters. In addition, mobile devices do not give prior notice of their movement or service availability to any entities in the environment. They will be cooperative and prepared to process composite requests and organize service composition as well as execution processes.

Mobile devices of interest use wireless communication channels to exchange information, and communicate with each other in ad-hoc mode. This work regards mobile ad hoc networks (MANETs) as the target network for the mobile devices, and discusses service provision in MANETs with high dynamism, where the network topology is likely to change faster than it takes to entirely complete service composition or execution.

This work assumes that composition clients have limited resource and communication range, which are insufficient to aggregate enough service providers for a service composition task, and manage the execution of a service composite. The solution proposed in this thesis addresses mobility by providing a fully decentralized self-organizing/adapting composition model.

3.3 GoCoMo Concept

GoCoMo targets the required features by a series of design concepts that address the problems of search failures, small discovery scope, provider loss, message loss, and path disruption (Figure 1.1) throughout a service composition process. This section describes these concepts from the perspectives of service searching, service selection, and service execution.

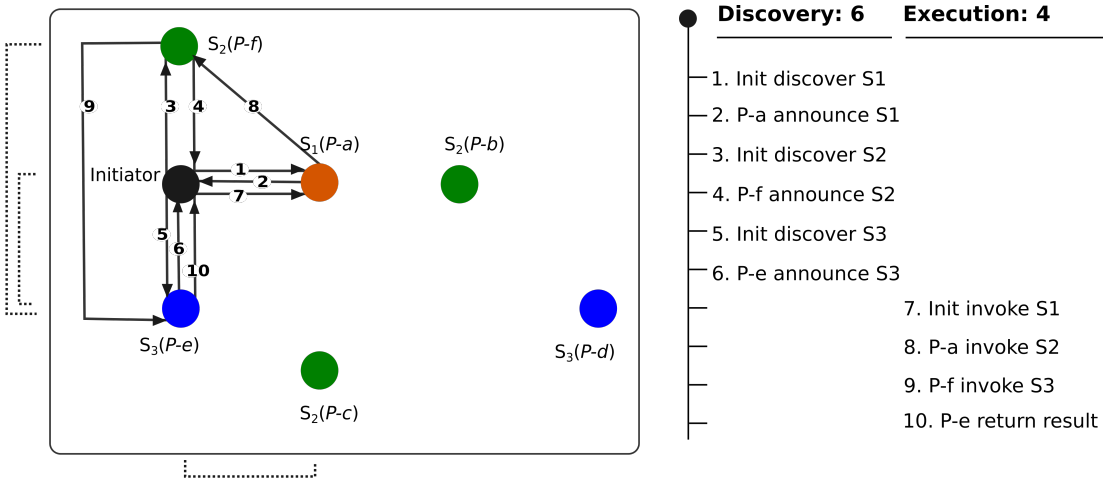
3.3.1 Service Searching

Service discovery approaches that rely on proactive search, as discussed in Section 2.1, are efficient at obtaining current service provider information. The approaches usually require service providers to announce the services they offer. Proactive service discovery distinguishes two methods in terms of how they distribute composition request issuers: semi-decentralized discovery employs a group of composition brokers who issue requests and accumulate information about service providers that have the potential to support a composition process; fully-decentralized discovery considers each service provider as a request issuer that discovers providers for its subsequent services.

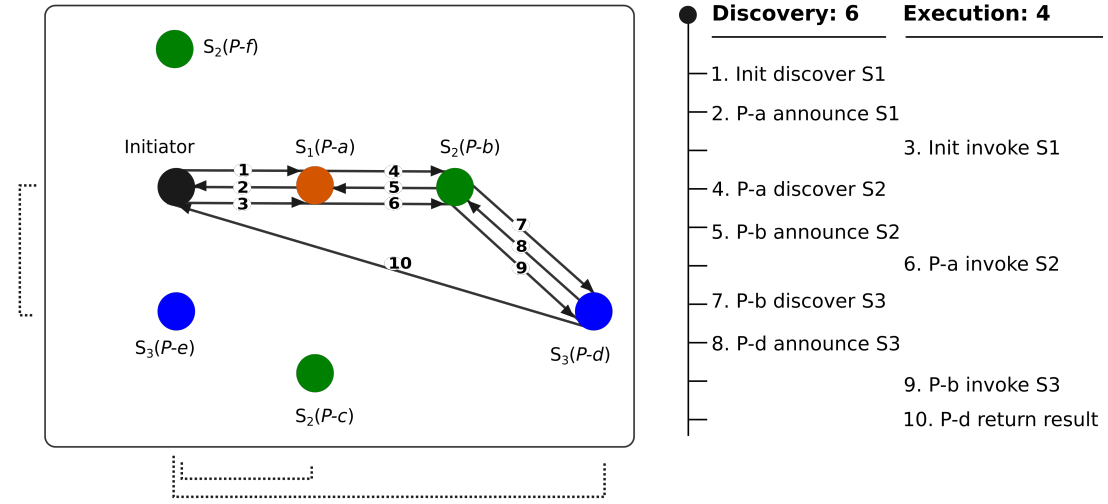
Figure 3.1 (a) and (b) illustrate the composition processes of semi-centralised service discovery [Prinz et al., 2008, Sen et al., 2008] and decentralized interleaved service composition [Groba and Clarke, 2014] for a sequential composite, respectively. Note that Figure 3.1 depicts the interactions only to discover and invoke the primary service providers, and those to find other candidate service providers and to bind service providers are omitted for simplification.

GoCoMo shortens the routes in a service execution path and reduces the communication by minimizing the geographic distance in one-hop routing and the number of hops in multi-hop transmissions, proposing a mechanism named decentralized backward planning-based announcement (Figure 3.1 (c)). The mechanism introduces the concept:

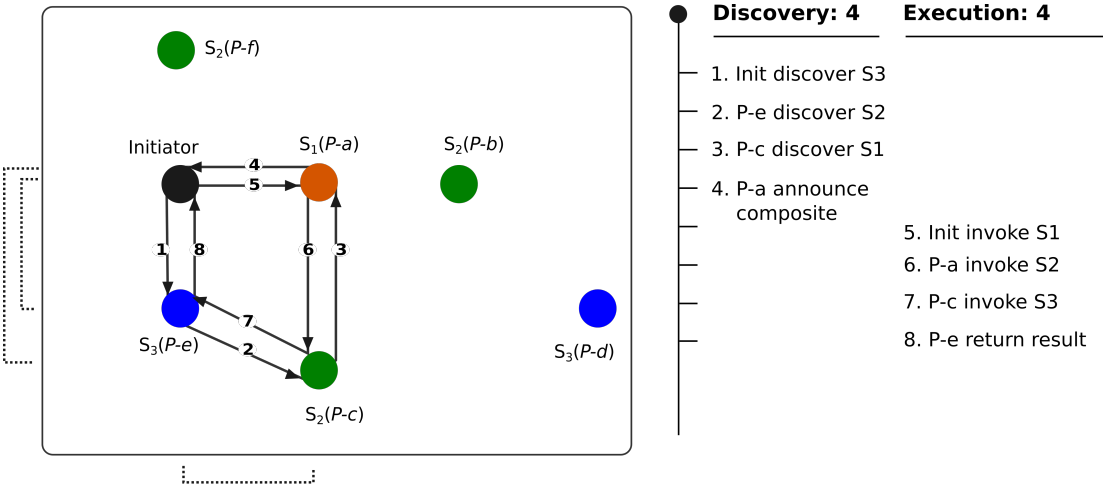
> **Concept 1: Planning-based composition announcement**
>
> The proposed protocol removes the announcement for each individual service, and uses a dynamic composition planning process during service discovery to allow a composition initiator to directly gain information about service composites from service providers. If a service provider receives a composition request and discovers that it partially fits the request's requirement, instead of announcing its own service to the requester, it forwards the parts of request that cannot be satisfied by its local knowledge to other service providers. Service providers do not contact to the requester, unless they can completely solve their received composition request. The composition initiator is only informed of completed service composites from service providers, and selects the most reliable (short routes and small execution time) service composite for execution.



(a) Semi-centralised



(b) Decentralised interleaved



(c) Decentralised backward planning-based announcement

FIGURE 3.1: Service composition for a sequential composite $S_1 \rightarrow S_2 \rightarrow S_3$. *Scale marks on the vertical lines show the number of interactions for service discovery and execution (**Communication-** the fewer, the better). The dash half-braces show the distance between successive service providers (**Locality-** the shorter, the better).*

As illustrated in Figure 3.1 (c), the proposed protocol has less interaction between primary service providers, and finds the shortest execution path. This protocol is based on a dynamic composition planning algorithm to resolve composition requests.

Dynamic composition planning has been discussed in Chapter 2.3, and there are various different planning mechanisms, such as graph-based planning [Rodriguez-mier et al., 2012, Ukey et al., 2010, Zhenghui et al., 2009], decentralized composition planning [Al-Oqily and Karmouch, 2011, Furno and Zimeo, 2014, Gharzouli and Boufaida, 2011], etc. Given the infrastructure-less nature of the target network, decentralized composition planning has the potential to support the planning-based composition announcement. Decentralized composition planning includes forwarding planning [Al-Oqily and Karmouch, 2011, Gharzouli and Boufaida, 2011], backward planning [Gharzouli and Boufaida, 2011], and bi-direction planning [Furno and Zimeo, 2014]. However, existing approaches search for services depending on pre-established semantic service overlay networks [Furno and Zimeo, 2014], assuming service providers have local geographic information [Al-Oqily and Karmouch, 2011], or simply flood a request to the network [Gharzouli and Boufaida, 2011]. These methods are expansive in MANETs. In addition, they only reason about sequential service flows. GoCoMo's service discovery model uses controlled flooding and backward planning, and extends them to reduce network traffic and to support complex compositions, for instance a service composite including parallel or hybrid service flows.

Controlled flooding, as discussed in Section 2.2.4, is infrastructure-less, but current approaches either use static control mechanisms or require service providers to have a-priori topology knowledge about their local networks. GoCoMo explores a novel dynamic controlled flooding:

Concept 2: Dynamic controlled flooding

Dynamic controlled flooding allows different nodes in the network to independently decide how to route a composition request depending on a combined context, including their current local network properties, the cost that has been paid by other nodes for routing this request and the discovered service providers' properties. A request routing process can flexibly stop, if a service provider is considered to be unreachable in a service execution process. Unreachable services are those that

will introduce a high cost for routing execution results (i.e., more routing hops), which is in conflict with global execution time-constraints

A composition request reflects user requirements which may include data-parallel tasks, such as multi-source data aggregation. GoCoMo needs to model user requirements in a flexible way to cope with various, diverse service networks. Generally, one way to enable service composition for such data-parallel tasks is to specify the tasks as abstract parallel workflows [Thomas et al., 2009] [Groba and Clarke, 2014]. An abstract parallel workflow includes control logic and ordered sub-tasks. Each of the sub-tasks can match with either one basic service, or trigger a process to generate a composite service when there is no service that suffices independently. This way, as analysed in Section 2.3, is inflexible at resolving data-parallel tasks. On the other hand, a composition process, in some cases, should be able to add new requirements into the original composition to satisfy data dependencies. Consider a navigation task as an example. The navigation has a service query 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. Under this kind of circumstance, a composition request is likely to be resolved by a parallel or hybrid service flow.

Assuming that no single node can maintain full knowledge of the network for execution planning, GoCoMo resolves a data-parallel task through interactions between service providers, without a-priori knowledge of the task's inner data transaction or the global knowledge of available services. Moreover, this work adapts composition requirements, depending on available services to flexibly reason about a composition result that may be sequential, parallel or hybrid.

Concept 3: Decentralized flexible backward planning

Decentralized flexible backward planning relies on service providers receiving a composition request, and generating the potential fragments of the global execution plan. A composition request can be gradually solved through backward planning processes on each participating service provider, and a service provider can adjust a composition request when new data is required. This mechanism reasons about,

if necessary, sequential, parallel, or hybrid service flows, using control logic, locally generating execution branches and managing the service execution process.

3.3.2 Service Selection

To reduce execution latency, a composition process selects service providers that can form the shortest service flow. Moreover, the execution time of each service should be considered as a factor of execution latency. Routes between successive service providers are also required to be short, which reduces the possibility of failures when invoking service providers or routing intermediate execution results.

Concept 4: Path reliability-driven selection

GoCoMo selects services based on the path's robustness (reliability) . A path's robustness value is calculated using multiple criteria including the execution time of each individual service, the length of service flows, and the routing hops during service discovery, which indicates whether a remaining execution path is likely to be reliable in a period of time.

On-demand service binding is flexible, reducing standby time for service providers (Section 2.4.4). Current on-demand binding approaches [Groba and Clarke, 2014] bind multiple functionally equal service providers to one composition process during service discovery, and send a token message to release them after a service provider is selected for invocation. Although the duration between binding and releasing may be small, many of the service providers that support the required functionality may become unavailable for other composition processes in the network during the time of standby. On the other hand, releasing service providers introduces traffic overheads.

Concept 5: On-demand binding

GoCoMo proposes a dynamic composition overlay that organizes service providers that are currently participating in composition processes. The dynamic composition overlay is temporary and only exists when a composition process is performing in the environment. Using this overlay network, a service provider is bound only when it has to be invoked.

3.3.3 Service Execution

Most service composition approaches that assume periodic service announcement are expansive because of frequent service matchmaking and multi-hop broadcasting. Normally, service announcements are used to collect information about locally available services and create a directory or overlay structure to maintain such information for service discovery. If a new service provider joins a network by announcing its services, all the neighbouring nodes of the service provider have to analyse the announced service specification (service matchmaking). Service announcement usually relies on multi-hop broadcasting messages that contain service specification, but multi-hop broadcasting is expensive itself [Karaoglu and Heinzelman, 2010, Lipman et al., 2009].

Concept 6: Runtime service announcement

GoCoMo distinguishes service providers, allowing the service provider that is currently participating in a composition process to analyse a new service and decide whether to invite its provider to join the composition. The service providers that are not engaging in any composition process (idle providers) use announcement messages only to obtain a sense of local network properties, such as service density, to get prepared for future compositions. GoCoMo allows service providers to announce their services to the network, using one-hop broadcast. The idle providers receive service announcement messages, instead of calling a matchmaker to launch an expansive semantic service matching process[1], they compute only network properties.

An execution path for a composition may need to be adapted to the environment. A global knowledge of available services is infeasible in GoCoMo's target environment, and so a composition adaptation process should be performed locally, without affecting the global QoS of the composition.

Concept 7: Local execution path maintenance

GoCoMo creates a fragment of an execution path on each individual service provider. Path fragments are maintained locally by merging new service providers or removing the parts of a path fragment that become invalid.

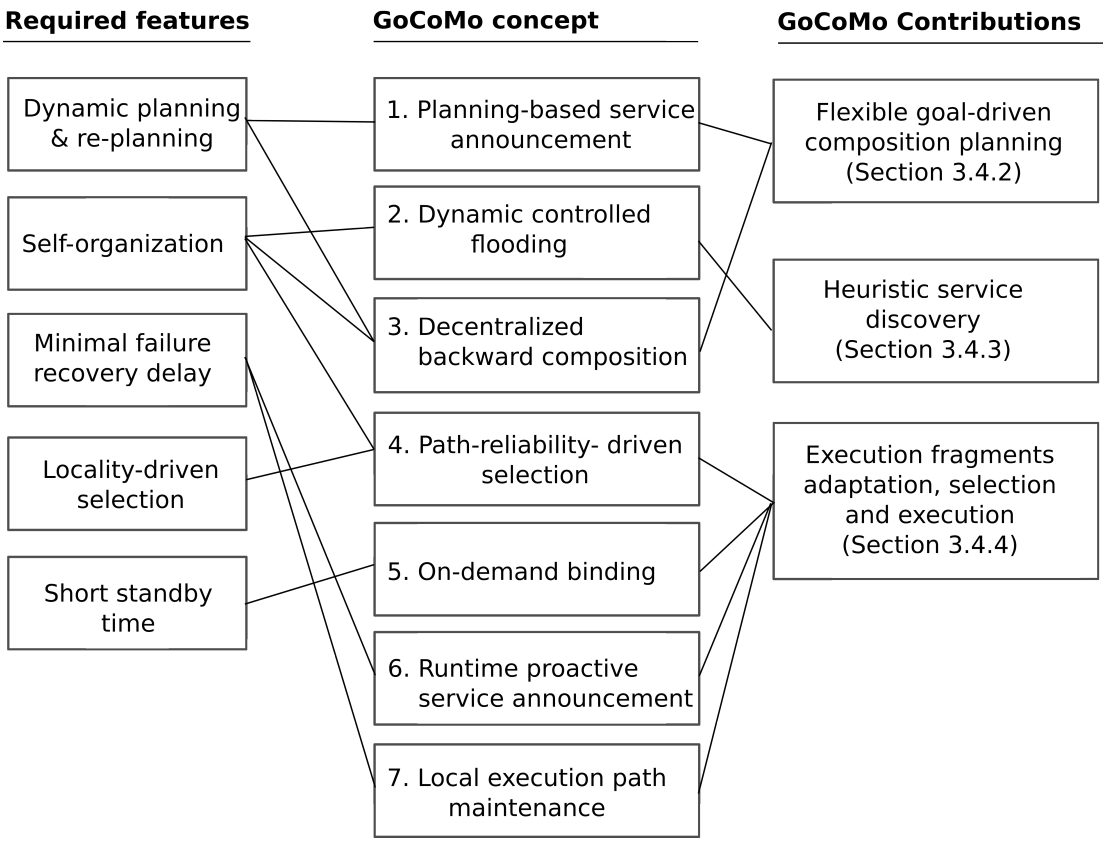


FIGURE 3.2: GoCoMo design concepts address design objectives and requirements

Figure 3.2 illustrates how the above GoCoMo concepts map to the required features. In particular, this thesis encapsulates these concepts and present the contributions: a flexible goal-driven service planning approach that applies Concept 1 and 3, a heuristic service discovery approach that uses Concept 2, and a service execution path selection and adaptation model that applies Concept 4, 5, 6 and 7. The rest of this chapter introduces the detail of GoCoMo model and the design of the GoCoMo contributions.

3.4 Service Composition Model

GoCoMo models a service composition process as AI backward-chaining. Classic AI backward-chaining processes [Hibner and Zielinski, 2007], also known as goal-driven reasoning [Furno and Zimeo, 2014], have been explored to ease the resolution process in the domain of service composition. A general goal-driven reasoning process for service composition is shown in Figure 3.3. The process starts by searching for the knowledge

[1]Semantic matchmaking takes 0.08-10.66*(s)* to return a result depending on matchmakers [Klusch, 2012].

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 $a \rightarrow c$). The process finds a solution when all the antecedents are reached. Specifically, an initiator issues a service request $a \rightarrow d$ to start a goal-driven reasoning 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.

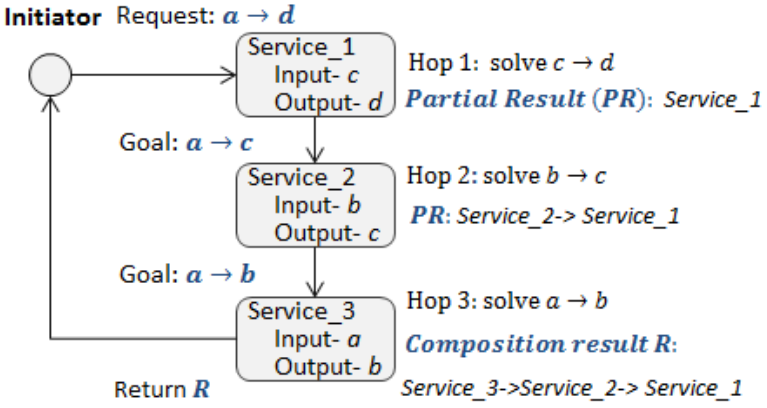


FIGURE 3.3: General distributed backward-chaining model for service composition

A goal-driven reasoning 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 run-time service availability. For example, in Anne's scenario (See Figure 1.2), 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. The planning resolves first to AudioNavigator, back to RampAccessChecker, then to StoreRouter as well as FacilityRouter, to StoreLocator, and finally to StoreQuery. A composition result is shown in Figure 1.2 (c)

Modelling service composition as such a process has been investigated in infrastructure-rich networks where composition planning is based on infrastructures like repositories [Zou et al., 2014] or pre-existing overlay networks [Kalasapur et al., 2007]. However,

this kind of infrastructure, as mentioned in Section 2.3, 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, as mentioned at the begining of this section, GoCoMo extends general goal-driven reasoning model, and handles the following issues:

- **Dynamic goal-driven composition planning**- 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 3.4.2)
- **Heuristic service discovery** - 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 3.4.3)
- **On-demand execution fragment selection**- is handled via online adaptable reasoning to create awareness of and compose potentially better services that may appear during service execution. (Section 3.4.4)

3.4.1 Service Model

This thesis assumes services' functions and I/O parameters are semantically annotated using globally understood semantics and languages and able to match a service request with semantic matchmakers [Chen and Clarke, 2014]. Such semantic service annotations are assumed to be kept in local service providers (mobile 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 user task. Service specification is defined as follows:

Definition 1. 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 QoS criterion as delay in composition and execution can cause failures [Groba and

Clarke, 2014]. A service composition model should select services with short execution time to reduce delay in execution.

A service composite for user tasks can be modelled as a restrictive data transition in which the system data change 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. User tasks can be modelled as a composition request:

Definition 2. A **composition request** is represented by $R = \langle R_{id}, \mathcal{I}, \mathcal{O}, \mathcal{F}, \mathcal{C} \rangle$, 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) $\mathcal{O} = \{\langle \mathcal{O}^S, \mathcal{O}^D \rangle\}$.

For service providers that participate in a backward composition process, each composition request can be resolved partially (or completely), and the remaining discovery request is forwarded to their neighbouring nodes to continue the discovering process. In this composition protocol, any remaining request is enclosed in a discovery message that is forwarded between composite participants.

Definition 3. A service provider in this thesis is a service deployment device that has a wrapped functionality exposed through a service interface and could therefore be used remotely as a service.

Definition 4. 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 Definition 6 on page 69), and h is a criterion value for request forwarding and service allocation (see Section 3.4.4 and 3.4.3).

3.4.2 Dynamic Goal-Driven Composition Planning

GoCoMo includes a goal-driven reasoning algorithm to support a fully decentralized service composition process, which is modelled as a state-transition diagram in Figure 3.4

TABLE 3.1: Composition model notations

Event	*msgIn(x)*	Receive a message x
	msgOut(x)	Send out a message x
Message	*A = req*	Composition request
	B = dscvMsg	Discovery message
	B'= dscvMsgNew	Updated discovery message
	C = cpltTok	Complete token to represent a complete solved goal
	D = tok ∪ *mediateRslt*	Allocation token to invoke a service Mediate results
	E = cpltRslt	Final results
Condition	∃ *exePath*	Complete execution path exists
	allResultsIn	Requester gets all the execution results
	participate	Service hosts are ready to providing services.
	usable	Services on the participant matches the request's goal
	cost	Discovery cost is still affordable (Heuristic discovery checking)
	end	An original goal has been matched
	joinService	Invoked service requires more input data to execute
	finish	The next-hop services exist
	∃ *nxtService*	Service execution is finished

and Figure 3.7. A transition between these illustrated states is triggered by message communication events (e.g., msgIn, msgOut) or local conditions (e.g., participate, usable). Table 3.1 clarifies the notations in these figures that represent message communication events and local conditions.

Figure 3.4 illustrates a composition process from a client's perspective. 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 candidate service providers who are capable of reacting to and reasoning about a composite request. They are also responsible for invoking their subsequent services during

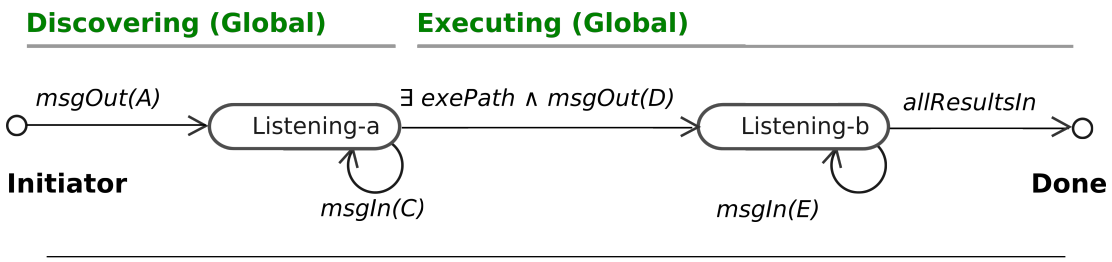


FIGURE 3.4: Protocol for global service composition. (Composition initiator's view)

execution. A client initialises a Time-To-Live (TTL) value $T_{discovery}$ to manage a global service discover process, when it issues a composition request. If the $T_{discovery}$ for a composition request reaches zero, the client selects, if it is possible, a completed composite to execute. The client then waits for execution results. Global service discovery fails if no completed composite has been received by the client when the service discovery process expires. If no execution result has been returned to the client after a previously defined QoS_{time} for the execution process expires, the global service execution fails.

The global service discovery process establishes a temporary overlay network called dynamic composition overlay (Figure 3.5), which contains all the reached usable composite participants. Such a network only lasts for the duration of a composition. In the network, a set of *execution guideposts* (See Definition 5) manages composite participants and controls the discovered service flow. As shown in Figure 3.5, each guidepost is maintained by a composite participant, linking the corresponding service to who sent the discovery message.

Definition 5. An **execution guidepost** $G = \langle R_{id}, \mathcal{D} \rangle$ maintained by composite participant P includes a set of execution directions $\mathcal{D}$ and the id of its corresponding composite request. For each execution direction $d_j \in \mathcal{D}$, d_j is defined as $\langle d_j^{id}, \mathcal{S}^{post}, \omega, \mathcal{Q} \rangle$, where d_j^{id} is a unique id for d_j, and the set $\mathcal{S}^{post}$ stores P's post-condition services that can be chosen for next-hop execution. The set ω 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 robustness of this direction, e.g., the estimated execution path strength and the execution time (Section 3.4.4).

A brief example for GoCoMo's basic composition protocol is shown in Figure 3.6. In a network consisting of one initiator and three service providers, the initiator's composition goal can be resolved by the service providers in a decentralized way. The initiator send

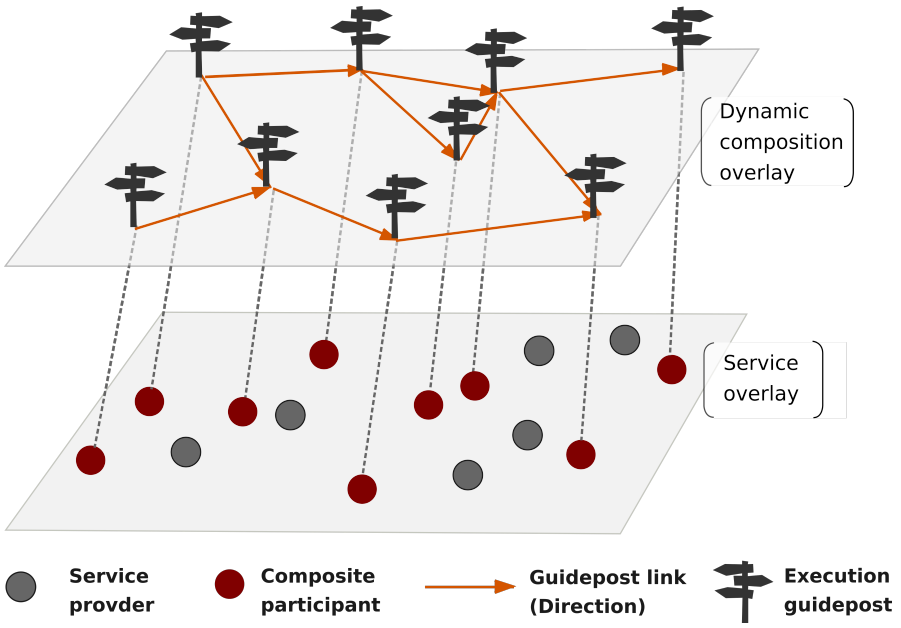


FIGURE 3.5: Dynamic composition overlay

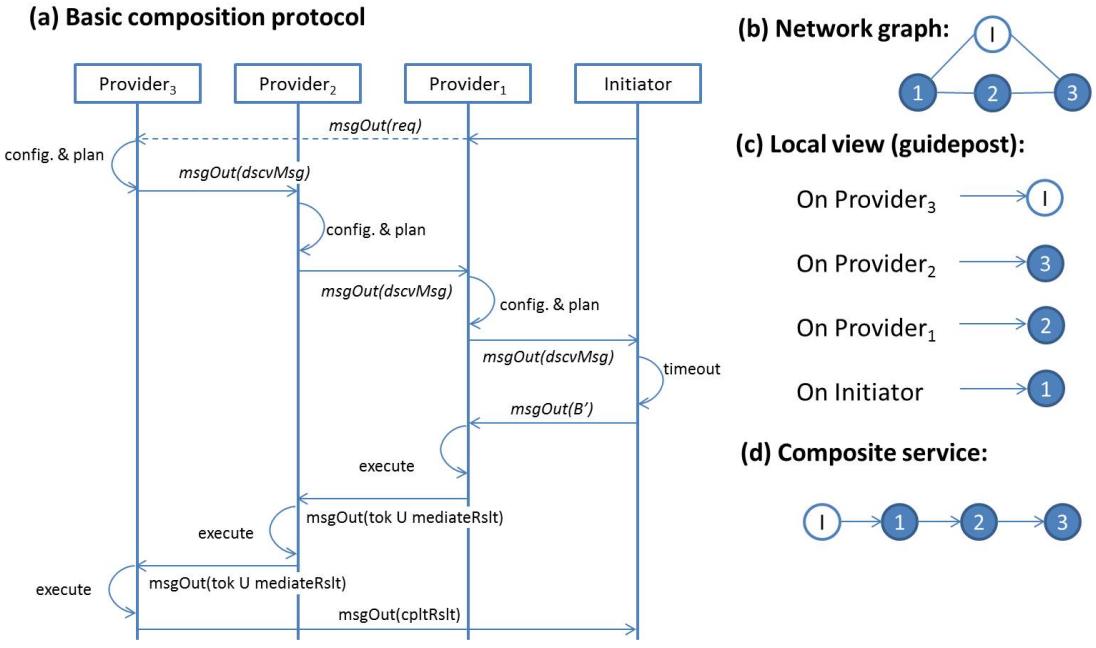


FIGURE 3.6: GoCoMo's basic backward planning protocol

a request (req) including the composition goal to its neighbours. The service providers resolve the goal backward by decomposing it into a list of sub-goals and matching services with the sub-goals. During this process, every candidate service provider creates an execution direction and stores it in its own guidepost (G). In each candidate service provider, such a process of resolving a sub-goal and generating an execution direction is called local discovering. After a composition goal is completely resolved, the initiator is acknowledged by the first service provider in a candidate execution path. The initiator generates an execution direction ($\rightarrow Provider_1$) to keep the information about the candidate execution path (i.e., the first service provider's address, the path's robustness property, etc.). When the initiator's discovery time expires, the initiator uses such information to decide which candidate execution path will be selected for execution, and an input data will be sent to the first service provider. As shown in Figure 3.6 the initiator selects $Provider_1$. $Provider_1$ gets invoked and then $Provider_2$ and $Provider_3$. In each selected service provider, such a process of invoking and selecting a subsequent service provider is called local composing. The following sections will introduce the two local processes (local discovering and composing) in detail. GoCoMo named the local behaviour in service providers to react a discovery message or a request and generate the direction as local discovering, and the process on service provider that invoke and execute a service instance and use directions to handover execution, as local composing.

3.4.2.1 Local Service Planning

From a composite participant's perspective, a (local) GoCoMo composition process includes a loop for service discovery to find and link all the reachable and usable services, and to generate/ adapt an execution guidepost. It also includes a flow for service composing process, based on the discovery result, to select, compose and execute services hop-by-hop on demand. In particular, as can be seen in Figure 3.7, the *local* discovering loop starts in the *listening* state and ends when the composite participant is invoked (the invoking state), while a *local* composing flow, starts in the *invoking* state and ends in a *composition-handover* state.

Algorithm 1 (*Planning* state) and Algorithm 2 (*Discovery-handover* state) describes a local discovering loop for composite participants. Note that the state transiting conditions like $\neg end$, *cost* and *usable* are not shown in these algorithms since they have

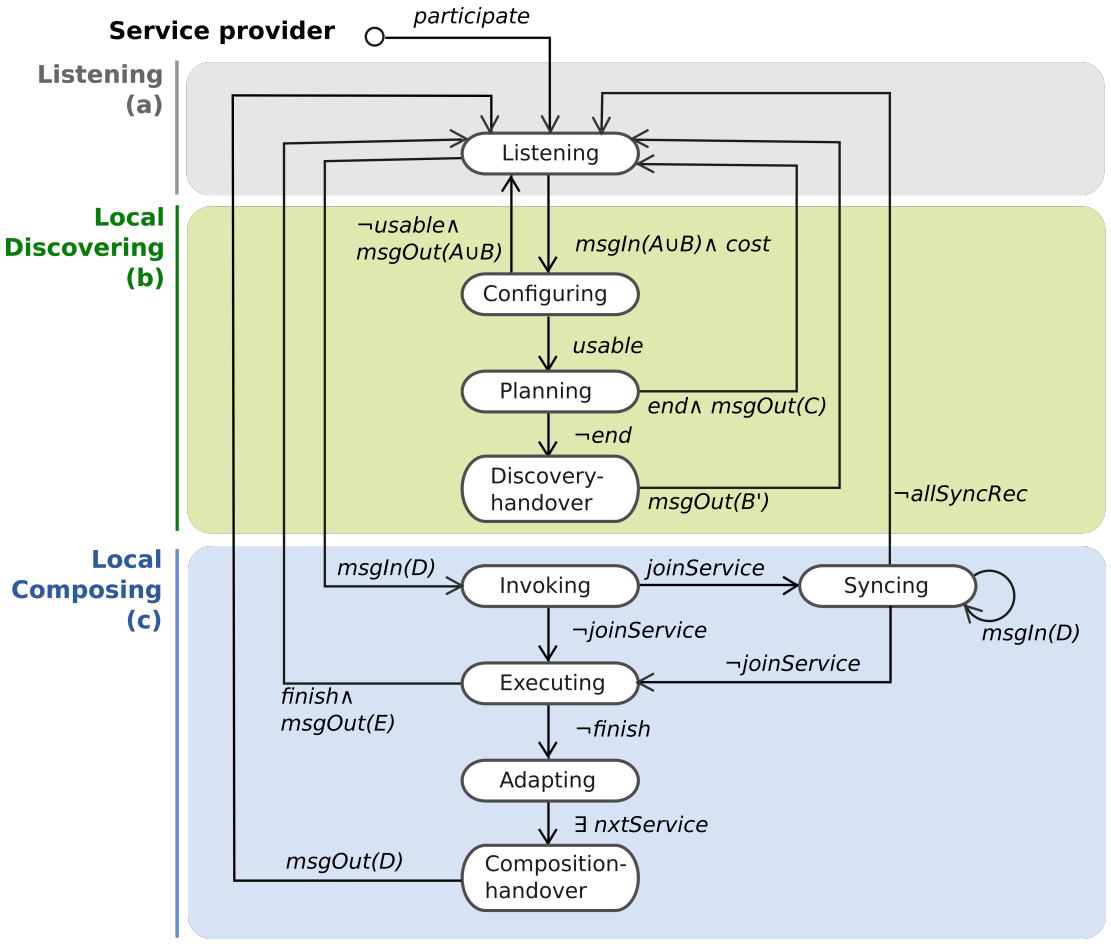


FIGURE 3.7: Protocol for local service composition. Service providers start and end in the listening state.

been illustrated in Figure 3.7. 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.

In the configuring state, a composite participant checks the goal matching level for a composite request (Line 2 in Algorithm 1). As the ultimate goal of a composite request is to produce the final, required output, if a service produces output that matches the one the composite request, the service is usable for this composite. A service $S = \langle S^f, IN, OUT, QoS^{time} \rangle$ matching a composite request R's goals ($R = \langle R_{id}, \mathcal{I}, \mathcal{O}, \mathcal{F}, \mathcal{C} \rangle$)[2]

[2]Find the definitions of the parameters on page 59.

is ranked from different matching levels:

$$GoalMatch(S, R) = \begin{cases} usable & \text{if } OUT \supseteq \mathcal{O} \\ usable^{+} & \text{if } OUT \subsetneq \mathcal{O}, OUT \neq \emptyset \\ unusable & \text{otherwise} \end{cases} \tag{3.1}$$

where $usable$ represents that S supports all the goal of R; $usable^{+}$ means that S supports only a part of the goal; and $unusable$ represents that S mismatches R. To prevent repeatedly checking for the same composite request, composite participants maintain a log for composite requests (Line 4 and 6 in Algorithm 1).

Data : Receive message $DscvMsg$ from Y. Receiver X hosts $S = \langle S^f, IN, OUT, QoS^{time} \rangle$.
Result: an execution guidepost

```
1  /* Configuring */;
2  GoalMatch(S, R);
3  /* Planning (when usable)*/;
4  if ∄DscvMsg^log then
5  |  New 𝒟;
6  |  DscvMsg^log ← DscvMsg;
7  |  if usable^+ then
8  |  |  Event← addJoin
9  |  else
10 |  |  Event← add
11 |  end
12 else
13 |  if DscvMsg^log and DscvMsg have matched or partially matched cache
   |  value then
14 |  |  Event← addSplit;
15 |  |  if matched then Remove matched cache;
16 |  |  if partially matched then Update cache
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 ← ⟨Rid, Y⟩;
31 |  endsw
32 |  case add||addJoin: d_y ← ⟨Rid, Y⟩;
33 endsw
34 //When a branch's resolving is finished
35 Initiate cpltMsg' = ⟨R', cache, h⟩;
36 Send cpltcvMsg';
```

Algorithm 1: Local planning algorithm

During service planning, a matched participant create a guidepost for the composition if there is no one, and then a direction to link to the request sender (Line 5 and 32 in Algorithm 1) into the guidepost. Or, based on different goal-matching results, a

planning process generates corresponding events to adjust the directions maintained in an execution guidepost (Line 4-24 in Algorithm 1).

For example, when a composite participant a receives a discovery message for a composition R for the first time, if a's service is *usable* to the composition, a will generate a execution guidepost G, model its link to the sender b of the discovery message as a direction $\langle R, b\rangle$, and add the direction into the G. If another discovery message for R is received by a from the sender c afterwards, and a's service is *usable* as well, a new direction $\langle R, c\rangle$ will be added to G.

Data : Receiver X hosts $S = \langle S^f, IN, OUT, QoS^{time}\rangle$.
Result: A $DscvMsg$ containing the remaining request R'

```
1  /* Handover */;
2  if (Event!=add)&&(RemainReq) then
3  |  Initiate DscvMsg' = ⟨R', cache', h'⟩, R' = ⟨I', O', F', C'⟩;
4  |  O' ← IN;
5  |  F' ← F − S^f_matched;
6  |  C' ← C − QoS^time;
7  |  Calculate h;
8  |  if GoalMatch(S, R) = usable+ then
9  |  |  Initiate cache_i = ⟨S_id, m, c⟩;
10 |  |  S_id ← P^rec, m ← OUT ∩ O;
11 |  |  c ← ⟨ num(m)/ num(O)⟩;
12 |  |  cache' ← cache + cache_i;
13 |  end
14 |  Send DscvMsg';
15 end
```

Algorithm 2: Local service discovering algorithm (handover)

In the discovery handover state (Algorithm 2), a new discovery message can be created that depends on its matching level for the in progress composite request. 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 the discovery message.

In the case of requiring other service providers to work together with S to support the goal, discovery messages keeps a *cache*. The *cache* stores information about a node Y which sends a request R to the composite participant S, in which the goal is only partially matched (Line 8 in Algorithm 2), i.e., when $GoalMatch(S, R) = usable^+$.

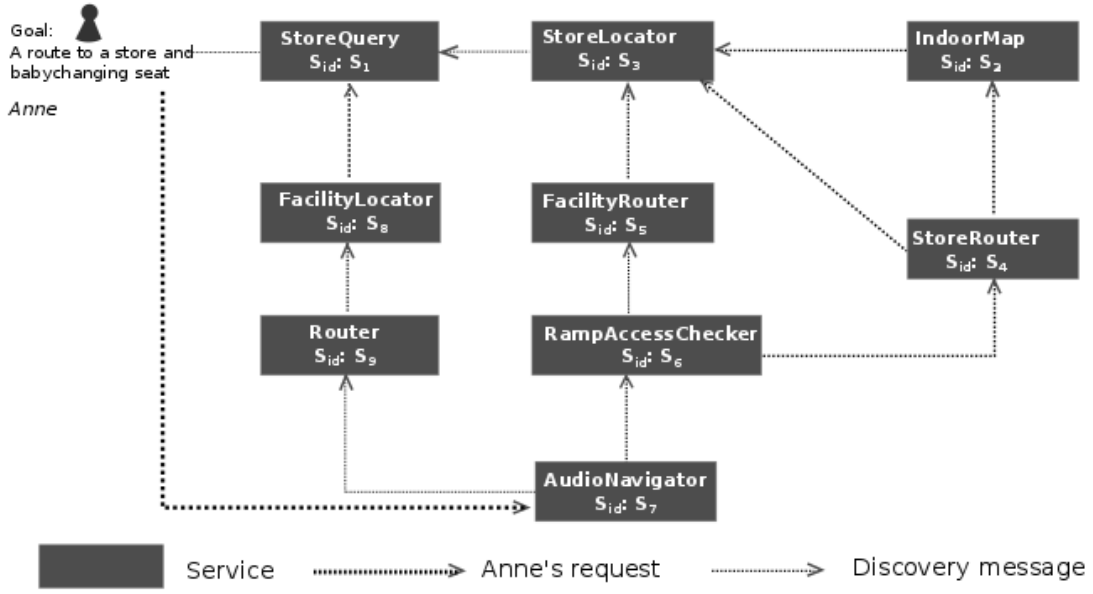


FIGURE 3.8: A backward planning example.

In other words, a composite participant caches a request sender when the participant meets only a part of the request's goal. The *cache* enables composition planing based on fine-grained goals for a composition request, allowing GoCoMo to flexibly use service providers to a composition that cannot satisfy the goal independently. This increases the scope of service discovery.

Definition 6. In a *cache*, a cached request sender is represented as a $C_i \in cache$ ($C_i = \langle S_{id}, G^{matched}, \rho \rangle$), 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 ρ ($\rho \in (0, 1)$) captures the progress of addressing the partially matched goal (e.g., how many goals of the request can be satisfied).

As shown in Figure 3.8, taking a compromise goal from Anne's scenario (See Figure 1.2 and 1.3) as an example, Anne requires a route to the nearest store that sells nappies and from where to a babychanging seat. During the process, an *IndoorMap provider* 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 can only provide the *local map*, the *StoreRouter*'s goal is half matched, so the *IndoorMap* caches the request sender by adding $C = \langle StoreRouter,\ localmap,\ 0.5 \rangle$ into the *cache* set. C will be forwarded, along with the discovery message, to the composite participant's neighbours until the required service that supports *store address* is found.

3.4.2.2 Complex Service flows

GoCoMo resolves a data-parallel task by allowing a composite participant to recognize and generate parallel execution branches. In particular, a composite participant recognizes AND-split-join control logic for a data-parallel task, and then generates a corresponding direction for it in the planning state.

AND-split-join control logic in a composition is detected using *GoalMatch* function (Formula 3.1 on page 66) and the analysis on received *cache*. The *cache* set in a discovery message also gets updated according to how the set matches the one in an old discovery message (Line 15-16 in Algorithm 1). A *matched cache value* is defined for two caches $C_1 = \langle S_1, G_1, \rho_1 \rangle$ and $C_2 = \langle S_2, G_2, \rho_2 \rangle$:

$$C_1 \textit{ and } C_2 \textit{ are} \begin{cases} \textit{matched} & \text{if } (G_1 \cap G_2 = \emptyset) \wedge (\rho_1 + \rho_2 = 1) \\ \textit{partially matched} & \text{if } (G_1 \cap G_2 = \emptyset) \wedge (\rho_1 + \rho_2 < 1) \\ \textit{mismatched} & \text{otherwise} \end{cases} \tag{3.2}$$

As mentioned (Line 7-22 in Algorithm 1), in the planning state a composition participant produces a event which triggers modification on execution directions. Composition participants generate event *addsplit*, if a new received $DscvMsg$ contains a *cache* matching or partially matching the existing one (*matched* or *partially matched* using Formula 3.2). Composition participants create event *addjoin*, if it is $usable^+$ to a new received $DscvMsg$ (Formula 3.1). Event *addsplit* and *addjoin* triggers the creation of two types of directions for execution guideposts:

Definition 7. 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.

Figure 3.9 illustrates how a data-parallel task is generated. In a network consisting of one initiator and three service providers, one of the service provider ($Provider_3$) depends on $Provider_1$'s and $Provider_2$'s output data for execution. During $Provider_1$'s and $Provider_2$'s local discovering process, the sub-goal generated by $Provider_3$ cannot be independently satisfied by neither $Provider_1$ nor $Provider_2$. So $Provider_1$ and

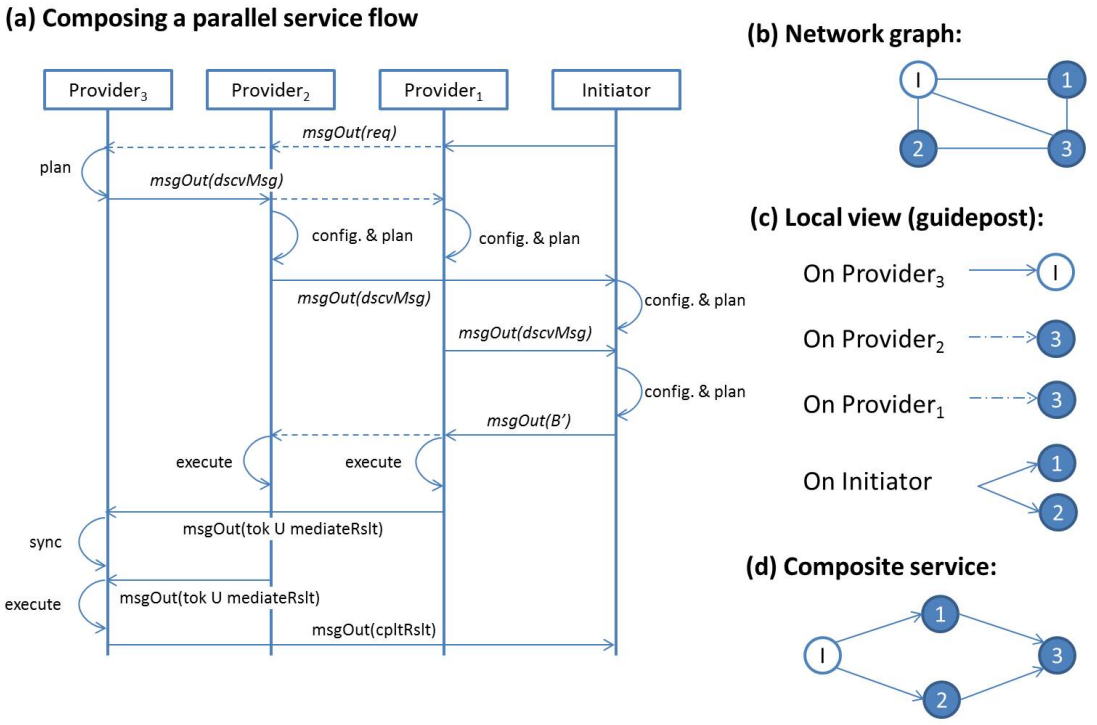


FIGURE 3.9: GoCoMo's backward planning protocol for composing a parallel service flow

$Provider_2$ cache their remaining sub-goals and forward it. They also create an AND-joining direction that links to $Provider_3$. The initiator receives the remaining sub-goals and finds them mergeable. Then an AND-splitting direction is created on the initiator. During service execution, the execution path is splitted into two branches at beginning, and then the branches are merged on $Provider_3$ by synchronizing the input data that comes from $Provider_1$ and $Provider_2$. Figure 3.10 illustrates an example in Anne's scenario for a dynamic composition overlay which contains AND-splitting directions and AND-joining directions. An AND-splitting direction on the service *StoreLocator* links to the *FacilityRouter* as well as the *StoreRouter*, while a AND-joining direction to this service indicates that a data syncing process will be needed on its subsequent execution.

3.4.3 Heuristic Service Discovery

The service discovery model finds services hop-by-hop based on service data dependency. During service discovery, service providers relay a passed-in composite request when they cannot match it. 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

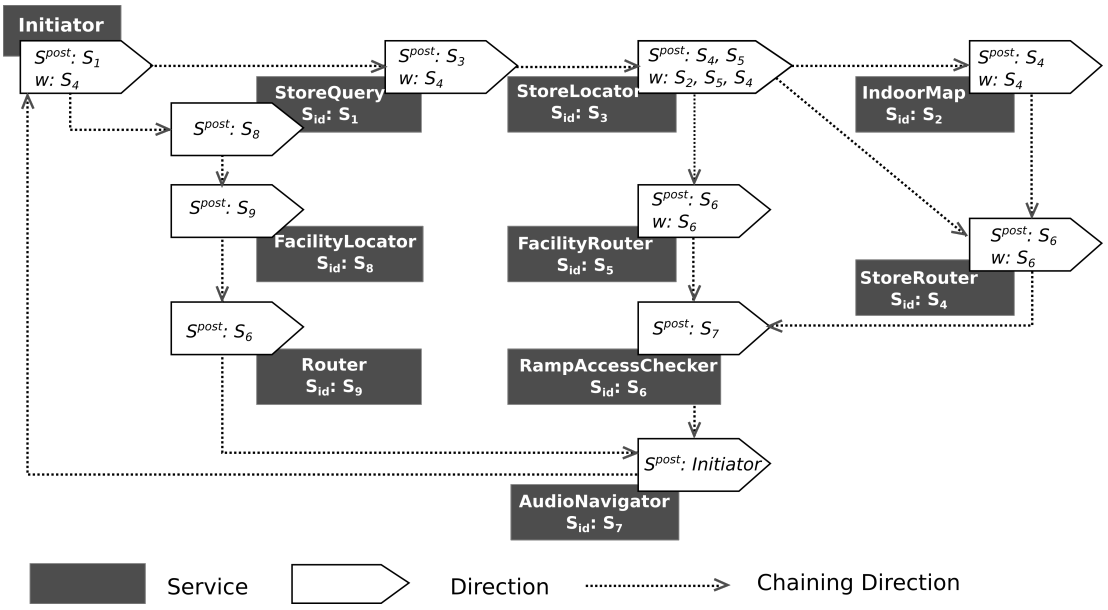


FIGURE 3.10: An example for dynamic composition overlay networks (DCONs). S_{id} represents an unique id for services. S^{post} indicates the next service

enough usable services in a reasonable period of time[3] by employing a practical number of composite participants.

GoCoMo uses a function $r(n)$ to calculate the relaying cost (relaying hops) from the last composite participant to current composite participant S_n (i.e., the n^{th} composite participant to resolve the composition) or a relay node R_i (for $r(i)$) (i.e., the i^{th} node to forward the composition request). These calculations are based on the heuristic value h in a discovery message.

$$h_n = \sum r(n) + n \tag{3.3}$$

The discovery cost is defined as

$$d(n) = \mu(h_{n-1} + r(i)) \tag{3.4}$$

where μ is a local communication channel parameter that is defined by local composite participants and relay nodes, which indicates the one-hop transmission delay. 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 smaller than a threshold τ: $T'_{discovery} = T_{discovery} - d(n) < \tau$, the node/ participant will stop relaying/processing

[3]A tolerable waiting time for a simple information query is about 2 second [Nah, 2004]. For operating tasks, the waiting time should be within 15 seconds [Miller, 1968].

the composition request. The threshold is determined by the remaining time constraint value on participant S_n, represented by

$$\tau = \frac{\mathcal{C}_n}{d} \tag{3.5}$$

where d is a weight value that determines the degree of interference in service discovery, and $\mathcal{C}_n$ is the execution time constraint on S_n.

The interference degree d is defined according to current service density. For example, in a network that has limited available services for a composition, the composition model sets up a low interference degree (large threshold) to enlarge the scope of service discovery, finding more service providers for the composition. In a service-abundant network, a high interference degree (small threshold) is preferred to reduce system traffic for service discovery. The selection policy for setting up the interference degree and the threshold value will be discussed in Chapter 5, based on network simulation results from scenarios with different networks and service configurations. This heuristic discovery check trades off the service discovery scope with 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.

Consider a brief example of heuristic service discovery (Figure 3.11). In a network including one initiator and ten service providers, the initiator wants to resolve a composition goal that has the potential to be satisfied by a composite service X. The initial service discovery time $T_{discovery}$ and the execution constraint C are $1s$. This example assumes that the time spent on one-hop-messaging is $= 0.1s$, and the relaying time on each participating node in a multi-hop messaging is omitted for simplicity. It also assumes that a candidate service provider needs $t = 0.01 - 0.05s$ to resolve a goal and $QoS^{time} = 0.01 - 0.1s$ to execute a service. As GoCoMo relies on backward planning, to discover X, $Provider_0$ needs to find $Service_2$ that has been deployed on $Provider_3$. According to the network graph, possible routes are: $Provider_0 \rightarrow Provider_1 \rightarrow Provider_2$ and $Provider_0 \rightarrow Provider_1 \rightarrow Provider_3 \rightarrow Provider_4 \rightarrow Provider_5 \rightarrow Provider_2$. During service discovery, every relay node checks the remaining discovery time before forwarding a discovery message, and the interference degree of the check is 1.5. As shown in Figure (d), on $Provider_4$ $T_{discovery}$' equals to $1 - 0.1 * (1 + 3) = 0.6$ and is smaller

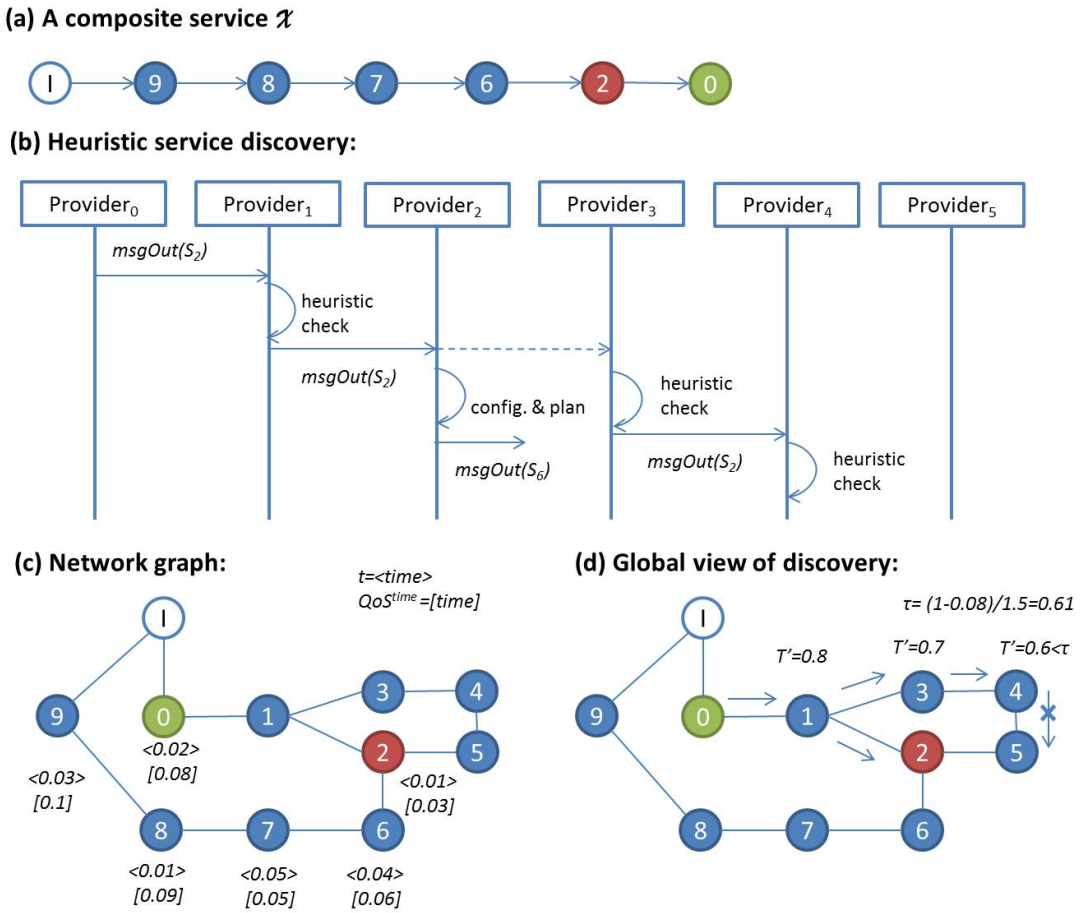


FIGURE 3.11: GoCoMo's heuristic service discovery protocol

than τ. $Provider_4$ stops relaying the message. This example shows how heuristic service discovery prevents a discovery message flooding the network.

3.4.4 Execution Fragments Selection and Invocation

In a global view, as shown in Figure 3.12, the general goal-driven approach *(a)* returns a set of independent service composites for selection, and GoCoMo *(b)* uses a control element to generate a flexible service composite that includes run-time selectable execution paths.

Control elements are enabled and managed locally by execution guideposts. Each execution guidepost that maintains possibly multiple directions acts as a OR-split[4] control

[4]An OR-split (split-choice) control element can have more than one outbound paths, and only one of them is selected for invocation [Liu and Kumar, 2005].

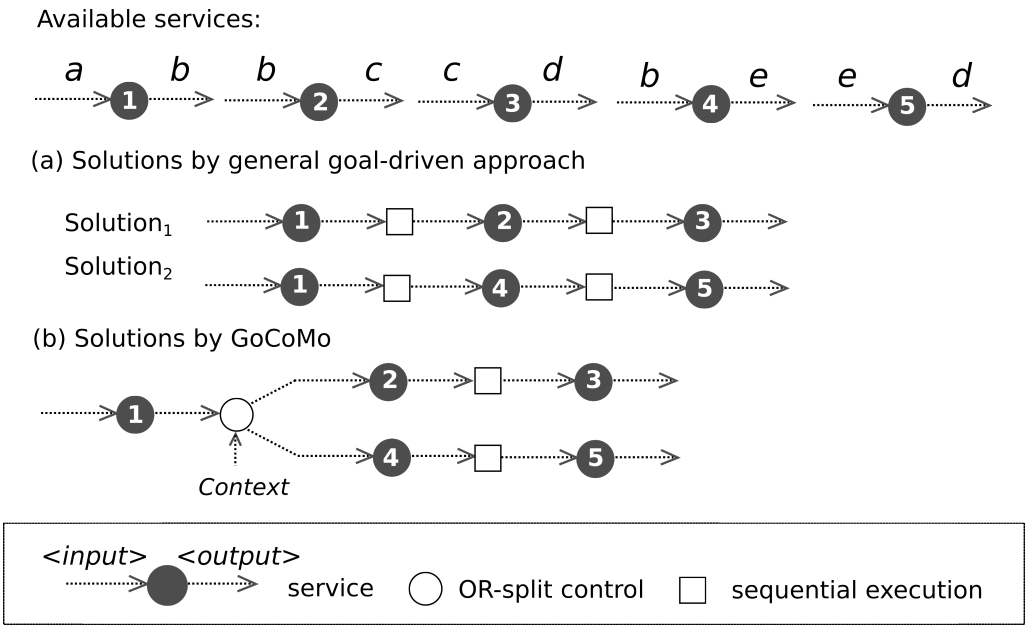


FIGURE 3.12: An example of GoCoMo's execution path, in contrast to the one in a general decentralized service composition model

element. This OR-split logic equipped execution guidepost allows composition participants to select the best path according to the current system context, like node availability or the robustness of the remaining path. It also allows a service provider to handle a path failure by recomposing execution path from the nearest execution guidepost other than the initiator.

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.

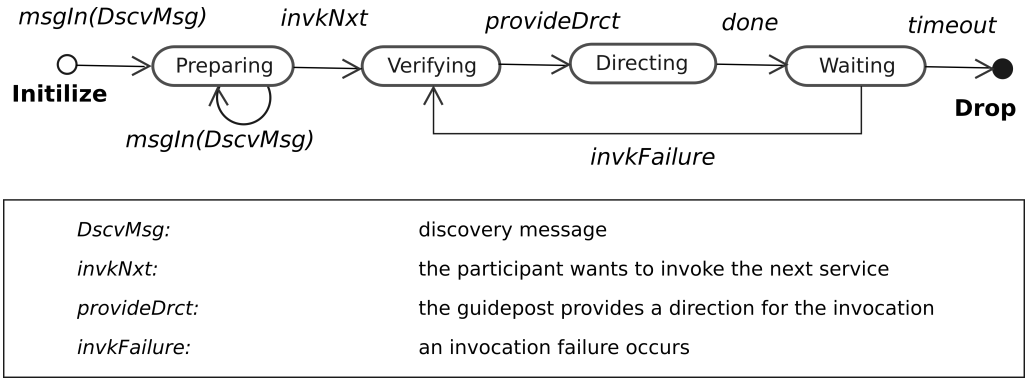


FIGURE 3.13: An execution guidepost's life cycle

Operations on path creation and selection are managed in an execution guidepost's life cycle. Execution guideposts have a life cycle with four phases: preparing, verifying, directing and waiting. Figure 3.13 shows the life cycle, and how it spans the duration of a participant's local discovering and local composing processes.

A composite participant S_n assigns a lifetime T_n for an execution guidepost when establishing it. The T_n is determined according to the remaining time constraints ($\mathcal{C}_n$) and a heuristic value (Section 3.4.3). Once an execution guidepost is initialized, it starts to prepare for composition by collecting directions and maintaining them. As mentioned in Algorithm 1 (Line 25-30 on page 67), the guidepost updates and verifies the maintained directions, in responding to an *event*. In an local composing process, an execution guidepost fetches the address of the first service provider in a primary direction. After that the first service provider gets invoked. The execution guidepost transits to a waiting state. Subsequently, if its lifetime T_n is still not expired, the guidepost will live for a while in case the remaining execution needs a rollback for failure recovery (see Section 3.4.4.2). If T_n has expired, the participant drops the guidepost for the composition request, removing itself from consideration in the composition. The overall composition process continuous to work with the dynamic composition overlay (see Figure 3.7 (a)).

3.4.4.1 Service Composite Selection and Invocation

The GoCoMo service composition process binds services on demand and releases them after execution. This means a composite participant's computing resources is 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. As mentioned, a client selects a service composite for invocation after its global service discovery process times out. Note that the actual information about the full composite is not sent to the client. Instead, the client receives the information about each service composite's reliability value and the first service provider's address. Based on the reliability values, the most reliable composite can be selected for invocation. The client sends a message to the first service provider to starts a global service execution process. The message contains an invocation token to bind service providers and the input data for the composition. Service providers receive the message and trigger their own local service composing process, as shown in Figure 3.7 (c).

A local service composition process is a transition from the invoking state to the composition handover state (Figure 3.7). 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 robustness.

$$\mathcal{Q} = \alpha * \sum QoS_n^{time} + \beta h_n \tag{3.6}$$

where α 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 β derives from a path duration estimation scheme [Sadagopan and Bai, 2003] to estimate an execution path's robustness in mobile ad hoc networks.

$$\beta = \lambda_0 v / \mathcal{R} \tag{3.7}$$

where λ_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 [Sadagopan and Bai, 2003]. GoCoMo service providers do not have a global view, and so such parameters are calculated using the properties of a service provider's local network (a.k.a., vicinity). Specifically, v and $\mathcal{R}$ are based on a service provider's own features, and λ_0 is determined by the number of received service announcement messages (see Section 3.4.4.2) in a particular period. 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.

3.4.4.2 Service Execution and Guidepost Adaptation

To make use of run-time proactive service announcement, GoCoMo allows composition participants to receive service announcement messages and invite new emerged service providers to participate in a composition.

Definition 8. A service announcement message is described as $SA = \langle P_{address},\ OUT_p \rangle$, where $P_{address}$ represents the unique address of the service provider, and the OUT_p is the output data can be provided by the service provider.

To reduce communication overhead and the time spent on dynamic service matchmaking, the service announcement message uses only the output data instead of the entire service specification defined in Section 3.4.1(Definition 1 on page 59).

When a composition participant receives a service announcement message from a new service provider, it uses the $GoalMatch$ function (Formula 3.1) to compute if the service provider is usable to the composition. The composition participant invites usable service providers to join in the composition process by sending a previously logged discovery message. The new service provider receives the discovery message and decides whether or not to participate in the composition according to its local resources and service availability. If the new service provider decides to engage in the composition, it performs the local service discovery process (see Figure 3.7), adding the inviter (the composition participant who invites the new service provider) as a direction in its own maintained execution guidepost. Inviting and composing a new service can occur at any stage of a composition process as long as the inviter has not yet been executed.

During service execution, the availability of the first service on a direction is known by a composition participant through a invocation token. When a service is found to be 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 D in Figure 3.7 and Table 3.1.). 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. This process is illustrated in Figure 3.14. If the link between $Provider_2$ and $Provider_3$ gets lost when executing the primary composition service $Provider_1 \rightarrow Provider_2 \rightarrow Provider_3$, $Provider_2$ detects such link loss and allows the service execution process to back-jump to $Provider_1$ who has a backup direction $\rightarrow Provider_4$ that is able to invoke another execution path $Provider_4 \rightarrow Provider_5 \rightarrow Provider_3$. An execution can ultimately fail if no execution path is available.

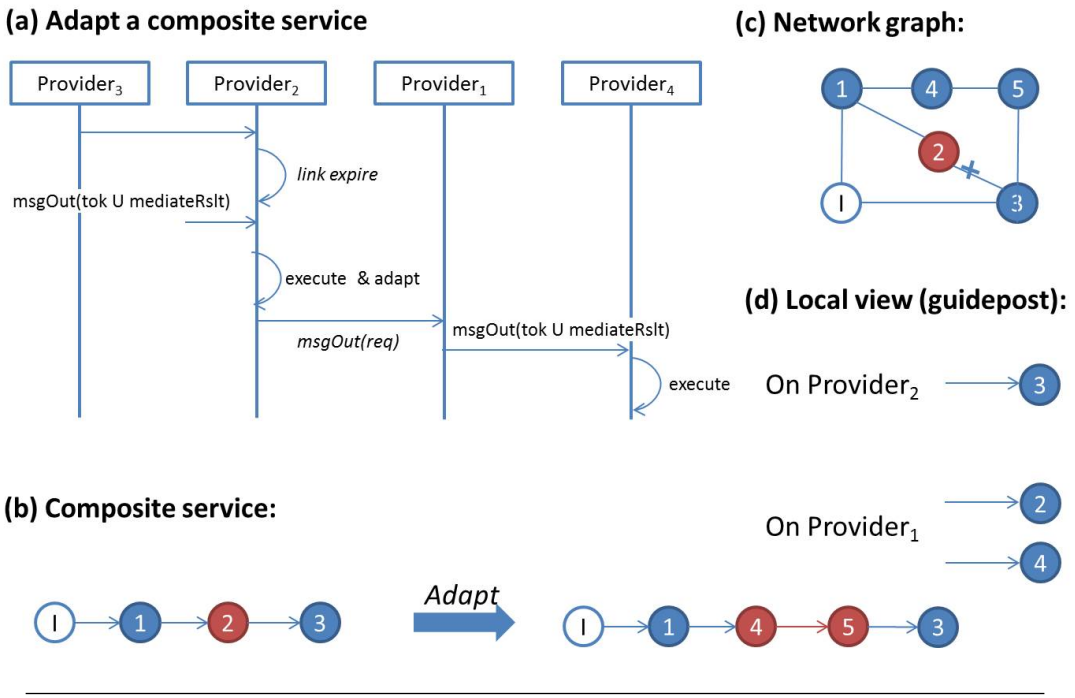


FIGURE 3.14: GoCoMo's heuristic service discovery protocol

3.5 Quantitative Analysis on GoCoMo

A quantitative analysis on GoCoMo are conducted, and the result is presented in Table 3.2. A set of parameters reflect the topics of GoCoMo's theoretical performance and scalability, including the resource consumption on each participant, the complexity of composition results, how service density affects a service composition process, and how many consumers or composition requests can be supported by the service composition system in a period (*Workload*). Generally, GoCoMo's message size and the number of the messages are affected by the length of the composed service flow. A long service flow can make messages larger and requires more messages to coordinate participant service providers. The composed service flow's logic and participants' mobility also have an influence on them. The more branches a service flow has, the bigger a discovery message are, and the faster the context changes, the more messages are needed to adapt the composition from any potential composition failure.

Composition delay represents the time spent on executing a composite service. It depends on each service's execution time, the time spent on one-hop transmission, and the length of a service composite's execution path including the number of transmission hops between successive services and the number of hops to route the execution result

TABLE 3.2: Quantitative analysis on GoCoMo's theoretical performance and scalability. *(avg.=average)*

	Best case	Worst case	Avg. Case
Avg. discovery message size	S	SL	N/A
Message/participants	2	$1+2N$	N/A
Overall messages	$2L$	$(1+2N)L$	N/A
Composition delay	$MLT\sum_{i=1}^{L} t_i$	$2MNLT\sum_{i=1}^{L} t_i$	N/A
Participant providers	L	NL	$NLe^{-\alpha D}$
Workload	BN	N	N/A
N/A = Not applicable Average size of a subgoal request = S Average candidate service providers per subgoal = N Average composition length (No. of subgoals) = L Service distance (No. of communication hops between two neighbouring services providers)= M Heuristic interference degree= D Buffer size of provider (the amount of composition information a service provider can maintain synchronously)= B Time spent on one-hop transmission= T Service i's execution time= t_i			

back to the composition requester. It also relates to mobility issues, and back-jumping a composition process to recover failures caused by a missing service provider increases the composition delay. In the best case, GoCoMo can invoke a service composite in which any service provider is in its successive services' vicinity. Note that, in a real world scenario, package loss during message transmission may increase failures and delay a composition process. Package loss rate is usually hard to be accurately and precisely predicted, and is affected by signal strength, network congestion, the receiver's message buffer size, etc.

The participating providers indicates the DCON's size, which means how many service providers join in the DCON to create a guidepost using their local resources for a service composition process. Less participant providers for one composition can expand globally available resources for the other compositions. However, if more service providers participate a composition, this composition has more backup execution paths when the primary one fails. The number of participant service provider is determined by heuristic service discovery's interference degree and package loss rate. Given the use of execution guideposts and on-demand service binding, GoCoMo allows resource-rich service

providers to support multiple composition requests synchronously, by maintaining a set of guideposts. The maximum workload depends on service providers' local resources (e.g., the buffer size).

3.6 Design Summary

This chapter introduces GoCoMo, and how GoCoMo tackles open issues as service provisioning failures and composition overhead. GoCoMo is a decentralised model designed for service composition in mobile and pervasive environments.

GoCoMo includes a flexible composition discovery model that supports planning-based service announcement and decentralized backward service discovery. Service providers cooperate to backward resolve a composition request from the goal to its initial state. The proposed composition discovery model generates a set of execution guideposts to enable decentralised service invocation and composite adaptation. This model also supports complex compositions. In other words, a service composite can include parallel service flows, when it is necessary .

In addition, GoCoMo introduces a heuristic service discovery model to achieve dynamically controlled request flooding. GoCoMo uses an infrastructure-less design for the heuristic service discovery model, and controls request floods by a threshold. Particularly, when the cost on a request routing process has exceeded the threshold, the process is stopped by service providers. Instead of assigning one uniform threshold to the global network, GoCoMo allows each individual service provider (router) to dynamically select a threshold, according to its own network property.

Moreover, GoCoMo provides adaptation and selection on fragments of execution paths (execution fragment). An execution fragment is maintained by a service provider, which includes information about the service provider's next participant provider, some important waypoints (i.e., execution branches' join node, see Definition 7 on page 70) on this path to the finial service provider, and a value to indicate the reliability of service providers on this path as well as the connection between them. Such an execution fragment is adaptable when the service provider detects a new execution path that has the potential to support the composition goal. Since execution fragments use Or-split logic to include different execution paths, when the primary path in a fragment fails,

an alternative can be rapidly re-composed for execution. GoCoMo performs execution path selection dynamically on service providers. A service provider finishes its own service execution, and then selects and invokes an execution fragment for the subsequent execution.

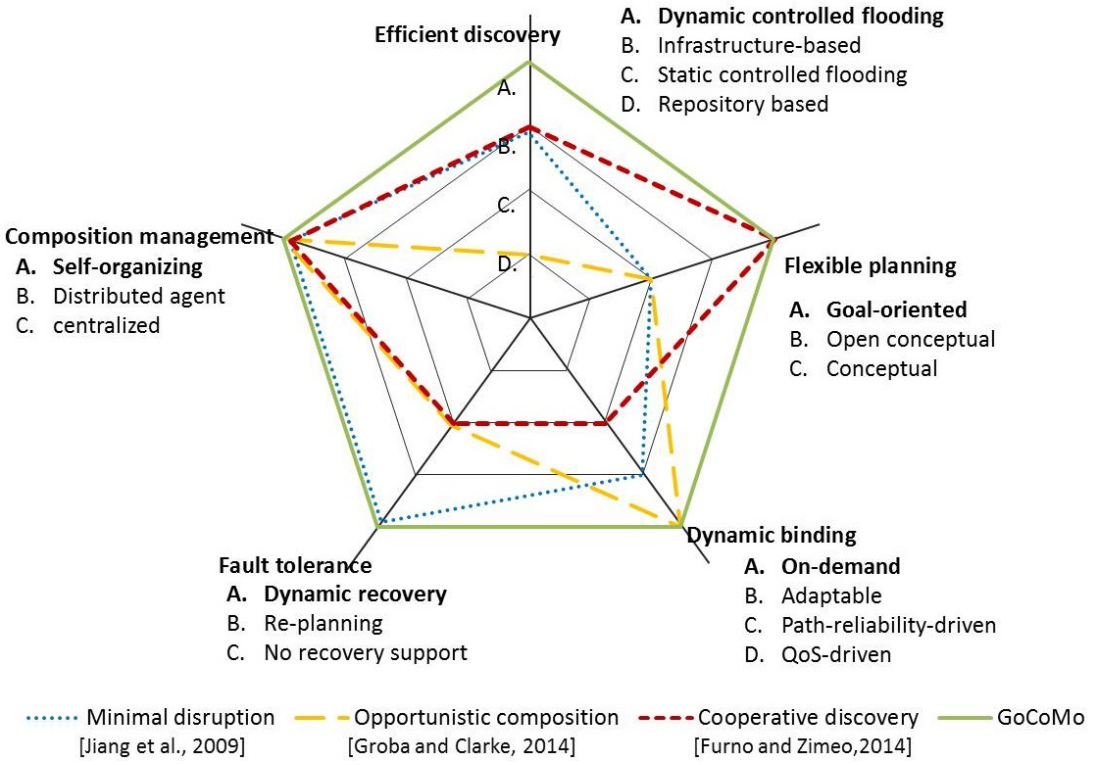


FIGURE 3.15: Kiviat diagram: GoCoMo's features comparing to 3 closest approaches

In summary, service composition can apply the GoCoMo process to dynamically plan for service composites, to self-organise a composition process, to reduce failure recovery delay, and minimise standby time for service providers, as illustrated in Figure 3.15. The remaining of this thesis describes how to implement GoCoMo model, evaluates how GoCoMo satisfies the design requirements outlined in this chapter, and discusses the limitation of GoCoMo.

Chapter 4

Implementation

The previous chapter describes the design of GoCoMo, and illustrates how it addresses the challenges of service provision in pervasive computing environments. The GoCoMo composition process is supported by a middleware, named GoCoMo middleware. The GoCoMo middleware contains the generic implementations of GoCoMo's algorithms and processes. This chapter describes in detail how to realise the GoCoMo middleware, and presents two prototypes, one for Android-based devices and the other for NS-3 platforms.

An overview of the GoCoMo middleware's architecture is presented in Section 4.1, which includes a description of the modules that comprise it, and an explanation of each module's responsibility in the GoCoMo service composition process. Section 4.2, 4.3, 4.4 and 4.5 describe the inner module implementation details and how the GoCoMo composition behaviours are realised by these modules. Section 4.6 introduces the GoCoMo middleware's prototypes.

4.1 GoCoMo Architecture

The GoCoMo service composition process executes in the context of the GoCoMo middleware. Figure 4.1 illustrates a high-level structure of the GoCoMo middleware, consists of 10 modules, including 6 major modules, 3 utility modules and 1 GUI module. Any participant in the GoCoMo environment will install all the middleware modules. The major modules' responsibilities are listed as follows:

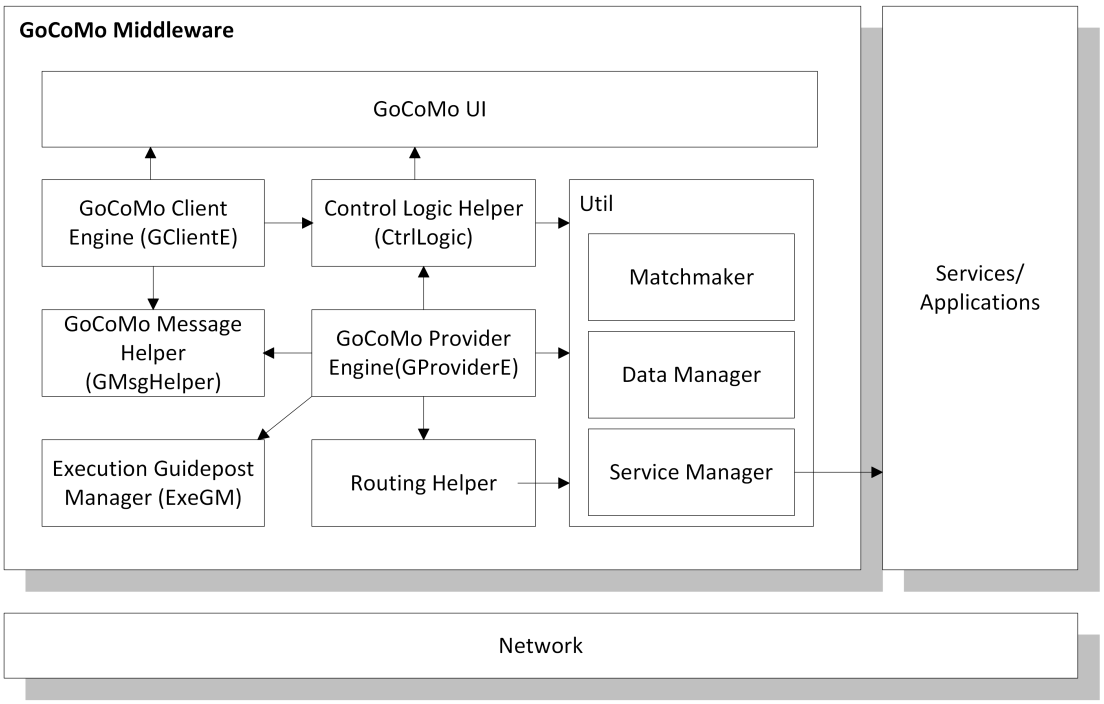


FIGURE 4.1: The GoCoMo middleware's architecture includes six major modules: *one UI module and three utility modules. Arrow lines represent interactions between these modules.*

GoCoMo Client Engine (GClientE) is responsible for managing global composition processes for service clients. Its operation includes issuing composition requests, managing the discovered service composites and selecting one of them for invocation.

GoCoMo Provider Engine (GProviderE) is responsible for managing local service composition processes for service providers, through interacting with other modules to manipulate such a service composition process's state transition.

Execution Guidepost Manager (ExeGM) underpins GProviderE by managing the information about the discovered service providers and service composites. It is responsible for overseeing the local execution guideposts' life-cycles and working out the best execution path for service invocation.

Control Logic Helper (CtrlLogic) is responsible for deciding if a composition request matches a local service and how the local service can engage in the composition. CtrlLogic, if necessary, also reasons about control logic in a service execution path, such as AND-split-join. In other words, it permits ExeGM to include parallel service flows to an execution path.

Routing Helper assists GProviderE to determine whether to continue a request routing operation, which implements the heuristics service discovery model introduced in GoCoMo.

GoCoMo Message Helper (GMsgHelper) is responsible for parsing or generating messages for information exchange between composition participants. Such messages include composition requests, DscvMsg (discovery messages), complete tokens, invocation tokens, recovery token, and service announcement messages.

A set of utility modules and a GUI module in the GoCoMo middleware collaborate with the major modules, facilitating composition processes and user interaction.

GoCoMo UI is a user interface that obtains user requirements from human users for GClientE to issue a composition request, and receives service registration information from service providers.

Service Manager provides an interface for service-related operations. Specifically, it allows services to be registered with the GoCoMo middleware on its own device, to enable service provisioning. It also invokes local services for execution at runtime, and fetches execution outcomes from the invoked service instance.

Data Manager is responsible for caching/fetching support data for a composition process, such as historical discovery messages and network property data.

Matchmaker is responsible for matching two entities and returning the matchmaking result to CtrlLogic. Matchmaking method can be syntactic or semantic depending on the domain of the operating environment.

These GoCoMo middleware modules collaborate with each other to *1)* search for service composites on behalf of composition clients and *2)* let mobile devices participate in a composition process as a service provider.

From a composition client's perspective (i.e., composition requester), the GoCoMo middleware receives a composition requirement from GoCoMo UI and formats the requirement as a service composition request, using GMsgHelper. The request is then sent to the local network to initialise a global service discovery process. Before the service discovery process finishes, any discovered service composite pushed to the client will be

verified by GClientE. GClientE keeps only the valid ones. After the discovery process, GClientE selects a service composite from those stored previously, and invokes it by sending the required input data to the first service provider in the composite, and then waits until an execution result arrives.

The GoCoMo middleware shares a provider's services in two ways: planning and advertising. In a planning process (see Section 3.4.2), a composition request, parsed by GMsgHelper, is firstly checked by CtrlLogic to find out whether a provider's local service satisfies the composition request. In particular, CtrlLogic uses Matchmaker to match the composition request's goal to the local service's output, and returns matchmaking outcomes to CtrlLogic. If the outcomes show that the service is "usable", CtrlLogic generates an event that suggests how the provider can engage the composition. GProviderE gets the event from CtrlLogic and asks ExeGM to initialize or update an execution guidepost, depending on the event.

A provider device advertises a service using Service Manager. If a service provider that engages a composition process as a participant, receives a service advertisement message, the participant's GMsgHelper draws service information from the message. Then, GProviderE fetches a locally cached passed-out discovery message from Data Manager, and calls CtrlLogic, examining if the service matches the composition's goal. As mentioned in Section 3.4, the matched services' provider gets invited to join in the composition by the participant.

During service execution (see Section 3.4.4), Service Manager invokes a local service, and passes the input data to GProviderE. After the local services are executed, a direction that indicates the subsequent execution path is selected by ExeGM, and GProviderE sends the execution result to the next service provider in the execution path. The service execution process will be continuously performed by the subsequent service providers.

More details about the interaction between these GoCoMo middleware modules and the inner module implementations will be presented in the rest of this chapter.

4.2 GoCoMo Client and Provider

GoCoMo App, as mentioned, has GClientE and GProviderE that realise the management of global service composition and local service composition, respectively. This section describes the inner implementation of these two modules, and their interactions with other GoCoMo middleware modules.

4.2.1 GoCoMo Client Engine

GClientE is a group of classes that enable GoCoMo's global service discovery and execution for composition clients (requesters). A global service discovery and execution process (see Figure 3.4 on page 62) includes the behaviours of issuing composition requests, receiving composition results, selecting a service composite from the results, and invoking the selected composite for execution. GClientE deals with all aspects of the client behaviors in a GoCoMo composition process.

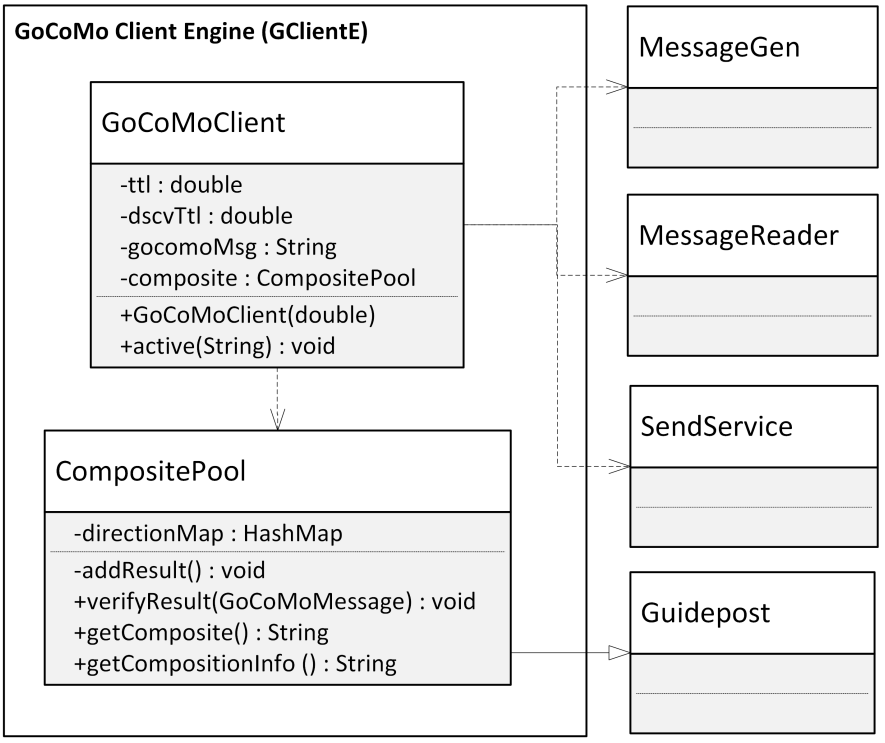


FIGURE 4.2: GoCoMo Client Engine Implementation: *class diagram of GoCoMo Client Engine and its relations to other GoCoMo modules*

Figure 4.2 illustrates the class diagram of GClientE. GClientE implements two classes *GoCoMoClient* and *CompositePool*. A *GoCoMoClient* object receives a GoCoMo message and conducts overall global composition management. It includes an active() method to atomically execute service searching or service composite updating. *CompositePool* extends the *Guidepost* class to get access to and maintain the composite data. That data is expressed as a *Direction* object, the implementation of which will be detailed in Section 4.4.

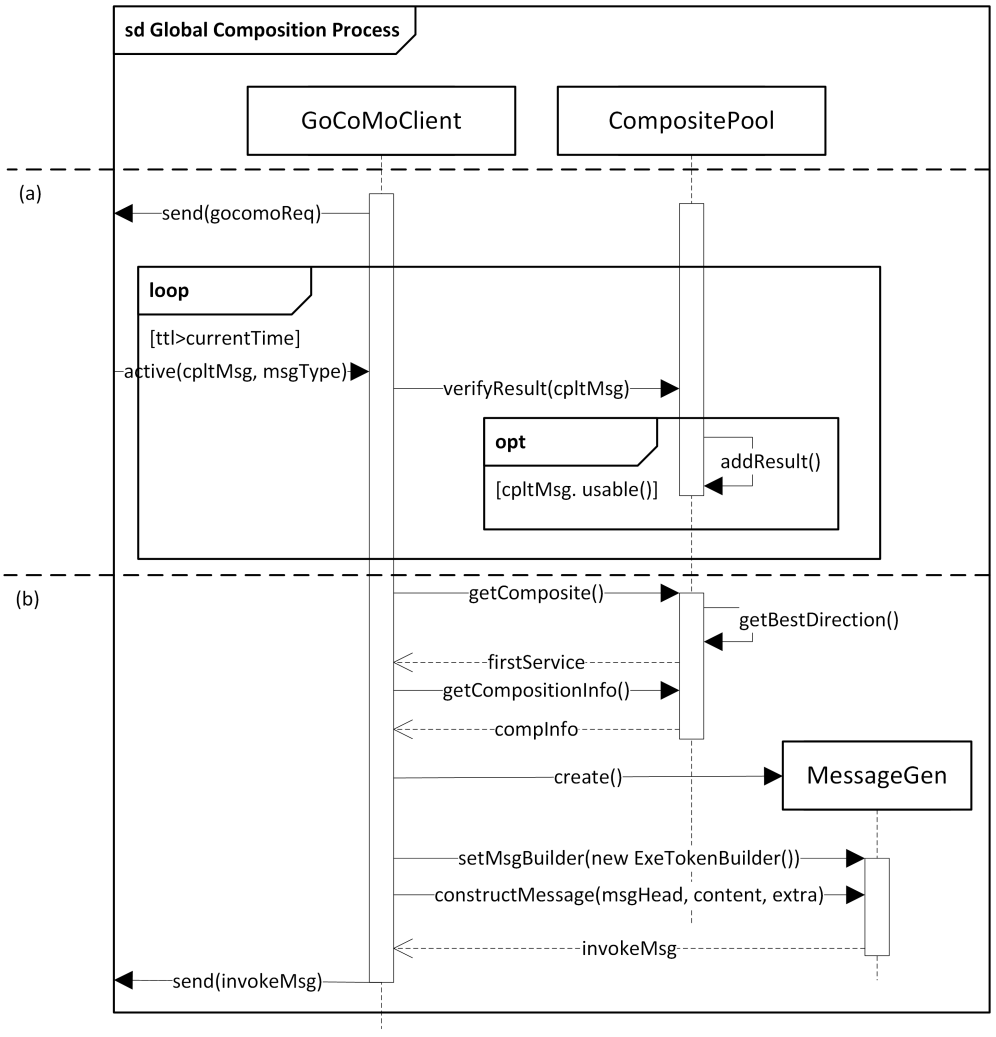


FIGURE 4.3: Global Service Composition Process Implementation: *high-level sequence of global service discovery (a) and invocation (b)*

Figure 4.3 illustrates the major interactions between *GoCoMoClient*, *CompositePool*, and the classes involved in (a) a global service discovery and (b) an execution process. When the GoCoMo middleware receives a GoCoMo composition request, a *GoCoMoClient* object is created, and at the same time a TTL value (see Section 3.4.2) is assigned

to the object that represents the object's lifetime. *GoCoMoClient* initialises a *CompositePool* instance, and calls a *send()* method in *SendService* class to pass the request to other peers in the network.

GoCoMoClient receives *cpltMsg* from a service provider that contains information about a discovered service composite, if the request is resolved by the service provider. *GoCoMoClient* retrieves a composite data, and calls a *verifyResult()* method in *CompositePool* to check the validity of the composite (i.e., whether or not this composite satisfies all the functional and non-functional requirements in a GoCoMo request). *CompositePool* keeps valid composites by modelling them as *Direction* data and storing this data in a key-value pair *directionMap*.

After the global discovery process times out, *GoCoMoClient* invokes the getComposite() method in *CompositePool* to obtain the address of the first service provider in the best candidate service composite. It then fetches extra information about the composite, for example a waypoint service provider in the execution, using the *getCompositionInfo()* method. *GoCoMoClient* calls the *getInvocationMsg()* in *GoCoMoMessage* class to obtain a *invokeMsg*, getting ready for the composite's execution.

4.2.2 GoCoMo Service Provider

The GoCoMo Provider Engine (GProviderE) is implemented as a *GoCoMoProvider* class. Instances of *GoCoMoProvider* locally control all the request resolving and service routing behaviours. The GoCoMo middleware invokes a *GoCoMoProvider* instance to handle a composition request, and launch a local service composition process. Figure 4.4 shows how a *GoCoMoProvider* object interacts with other GoCoMo classes to enable local service composition processes.

In local service discovery, a composition discovery message (*dscvMsg*) received by the GoCoMo middleware is first managed by a *GoCoMoMessage* object that analyses the type of, and retrieves the content of this message. The processed message is then passed to *GoCoMoProvider* (Figure 4.4 (a)). *GoCoMoProvider* first calls *genLogic()* in *LogicController* class that returns an event data to trigger the locally cached execution guidepost's adaptation. As mentioned in Section 3.4.2 (page 60), GoCoMo defines four events, each of which triggers an adaptation behaviour defined in *GuidepostManager*.

Adaptation behaviours include *add, addJoin, addSplit* and *adapt*. The event data is sent to an instance of *GuidepostManager* to generate/update an execution path. *GoCoMoProvider* afterwards chooses to keep forwarding the remaining request or send a discovered composite to the client, depending on whether or not the passed-in request has been completely resolved.

Service advertisement messages are also handled in local service discovery processes. As shown in Figure 4.4 (b), *GoCoMoProvider* receives a service advertisement from other devices, asks *LogicController* to measure if the service's output meets the composition's goal, and then invites the provider of the service that satisfies the composition's goal to engage in the composition by sending out a *dscvMsg*.

When a global service execution process reaches a local service, this service's execution is invoked by an *exeMsg* data. As introduced in Section 3.4.4, the *exeMsg* data includes the input data of the local service and the address of the last guidepost that maintains multiple directions for this composition. Besides, if such a service is in a parallel execution flow, its join node(waypoint)'s address is also included. Figure 4.4 (c) shows how *GoCoMoProvider* invokes a local service and how the subsequent service composite is selected. Similar to *GoCoMoClient*'s behaviour shown in Figure 4.3 (c), *GoCoMoProvider* obtain the next service provider's information and receives an *exeMsg* in the *getExeMsg()* callback. The *exeMsg* is then sent out to invoke the next service provider.

4.3 Routing Controller

In a local service discovery process (Figure 4.4(a)), *GoCoMoProvider* uses a *RoutingController.verify()* method to get permission to continue the local service discovery process. The *RoutingController* class, as shown in Figure 4.5, has methods to compute discovery cost and generate new heuristic values. The *verify()* method in *RoutingController* returns a boolean value, and a "true" value represents that the discovery cost is still affordable and the local service discovery may continue. Heuristic values were introduced in Section 3.4.3. See Formula 3.3 (page 72) for more details.

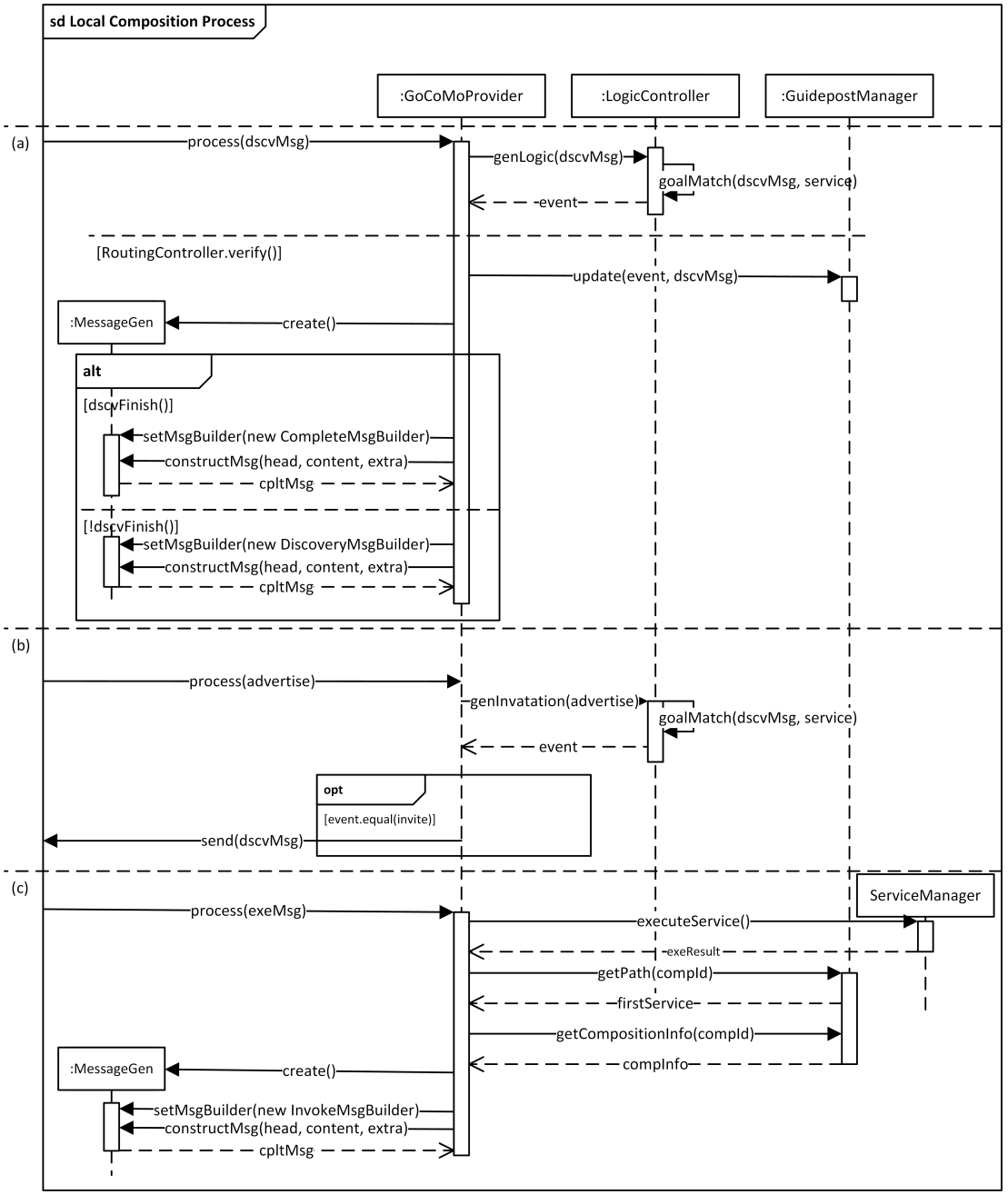


FIGURE 4.4: Local Service Composition Implementation: *(a) high-level sequence of local service discovery, (b) inviting new service providers, and (c) invocation.*

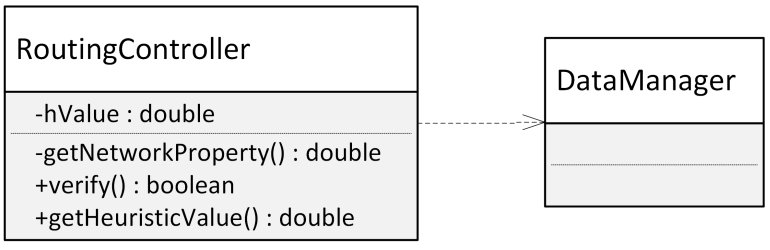


FIGURE 4.5: Class diagram of RoutingController.

4.4 Guidepost Manager

A guidepost is a networked element in a dynamic composition overlay network (see Section 3.4.2, Definition 5). A guidepost is associated with one GoCoMo composition process, caching information about candidate service composites. Such candidate service composites' information are modeled as directions. Each direction has an unique id, and contains a list of service providers that would require input data from the guidepost's host device for service execution, a list of way-point nodes, and a value that indicates the reliability of its corresponding service composite (see Section 3.4.2, Definition 5 and 7 on page 70 for more details).

The GoCoMo middleware uses Guidepost Manager to oversee a guidepost's life-cycle. Guidepost Manager is a group of classes supporting a series of behaviors including creating execution guideposts, verifying a guidepost, updating directions, etc.

Figure 4.6 depicts Guidepost Manager's class diagram, which contains classes *Guidepost*, *Direction* and *GuidepostManager*. *Direction* is a complex type containing *id, postConditionNode, qos,* etc. *Direction* objects for the same composition are maintained in a *List*, distinguished by their id attributes, and mapped to one *Guidepost* object. In other words, *Guidepost* objects and *Direction* objects are saved as paired key-value sets. Once a *Guidepost* object is destroyed, all the *Direction* objects maintained in the *Guidepost* will be dropped. *GuidepostManager* consists of methods that get data from or update a *Guidepost* object.

4.4.1 Adapting a Guidepost

A key point for GoCoMo is that a guidepost can be dynamically adapted, which allows a participant to *1)* cache newly available service providers for a composition, *2)* locally merge multiple execution branches to form a parallel service composite, and *3)* replace service providers by one that provides better QoS. As mentioned in Section 4.2.2, *GoCoMoProvider* passes an event (*add, addJoin, addSplit* or *adapt*) to *GuidepostManager* to trigger adaptation on an execution guidepost (see *update(event,dscvMsg)* in Figure 4.4(a)). Figure 4.7 outlines diverse events and their accordant adaptation behaviours.

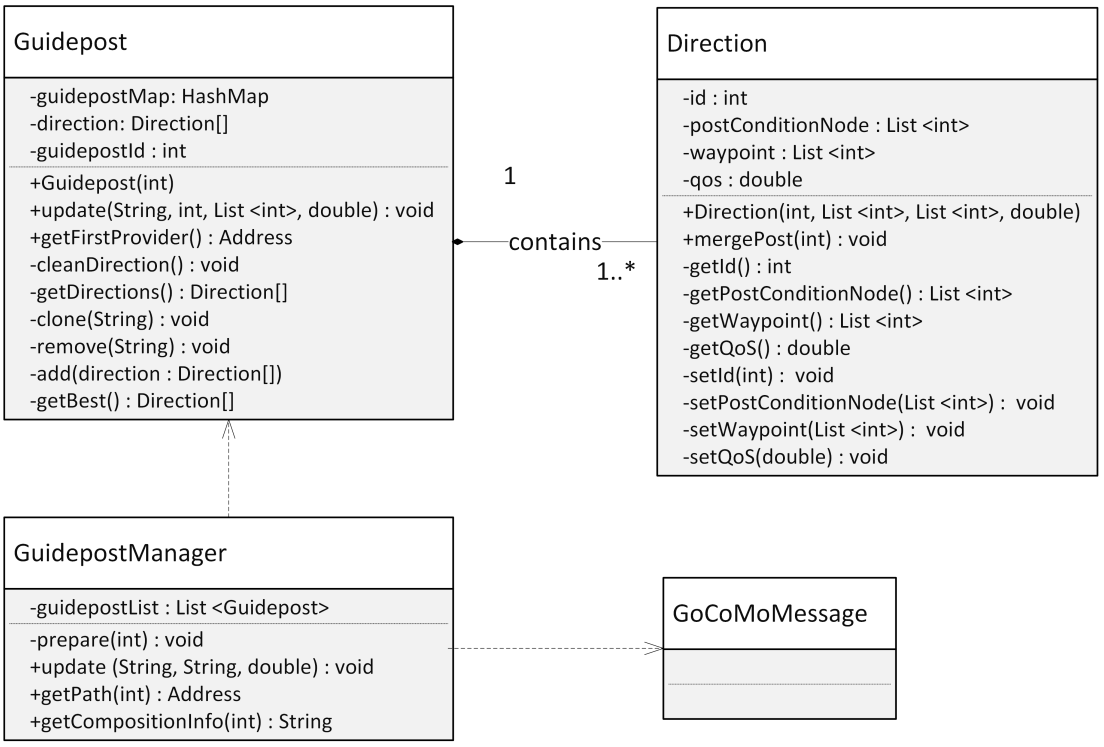


FIGURE 4.6: Guidepost Manager Implementation: *a class diagram of Guidepost Manager.*

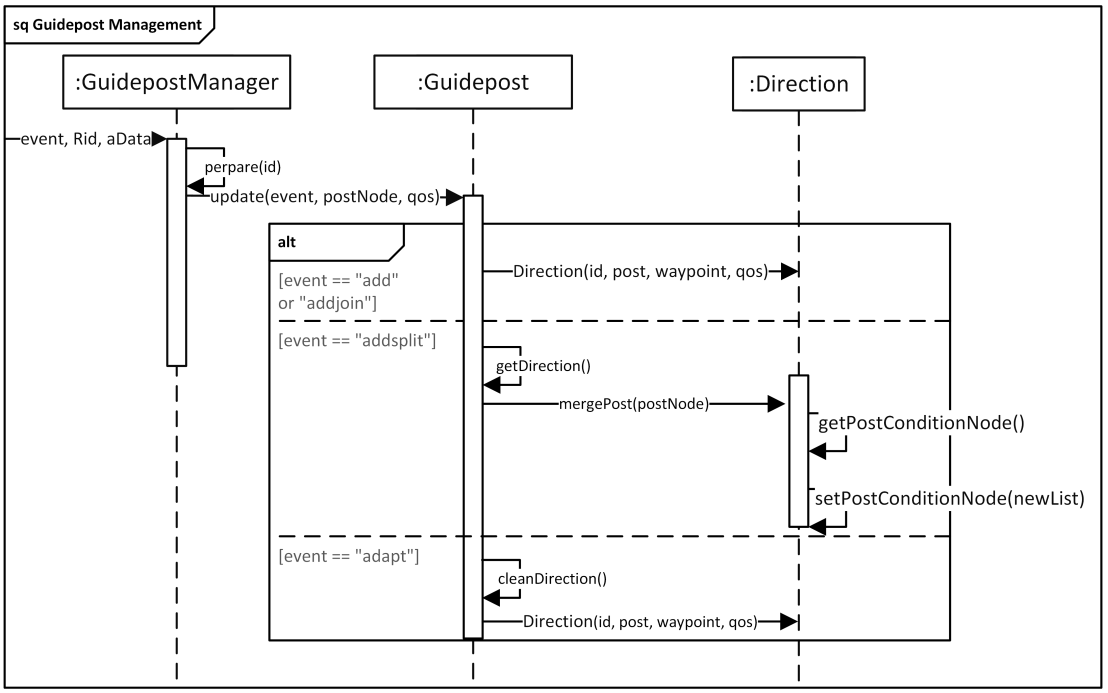


FIGURE 4.7: Guidepost Manager Sequence: *high-level sequence of Guidepost adaptation behaviours. Interactions related to the HashMap object are not illustrated.*

GuidepostManager receives this event, a composition process id and the adaptation data, and then fetches the corresponding *Guidepost* using the composition process id. The *Guidepost* gets activated, and updates the direction it maintains based on the event. For event *add* and *addjoin*, *Guidepost* obtains the *postCondition*, the *waypoint* list, and the *qos* from the adaptation data, and uses them to create a new *Direction* object and a key-value pair to store the object. Note that *Guidepost* inserts the post condition node into the waypoint before generating direction (see Definition 7 on page 70, AND-joining direction) for an *addjoin* event. For event *addsplit*, *Guidepost* updates currently stored directions by add a new node into their postNode list. For event *adapt*, *Guidepost* calls the *cleanDirection()* to remove the stored *Direction* data, and then creates a new *Direction* mapping to the composition's *Guidepost*.

4.4.2 Guidepost Data in Service Execution

A *Guidepost* object is used in a service execution process to provide composite information for *GoCoMoProvider*. The main operations to get composition information from *GuidepostManager* has been introduced in Section 4.2.2, and depicted in Figure 4.4 (b). In the *GuidepostManager* class, the *getFirstProvider()* method is called to return an address data of the first service provider in the best execution path.

4.5 GoCoMo Message Helper

GoCoMo Message Helper (GMsgHelper) is a group of classes that define different GoCoMo messages data and support message generating and parsing. As depicted in Figure 4.8, this thesis uses the builder pattern to realize GMsgHelper. A GoCoMo message consists of header data, content data and extra data. The header data includes information about the message type, request id, the sender's address and the receiver's address. The content data, depending on the message type, can include a composition request or service information. The extra data provides additional content data, like the address of the nearest service provider that hosts a Or-split guidepost. An example of GoCoMo messages is presented in Figure A.3 on page 137 in Appendix A.1.

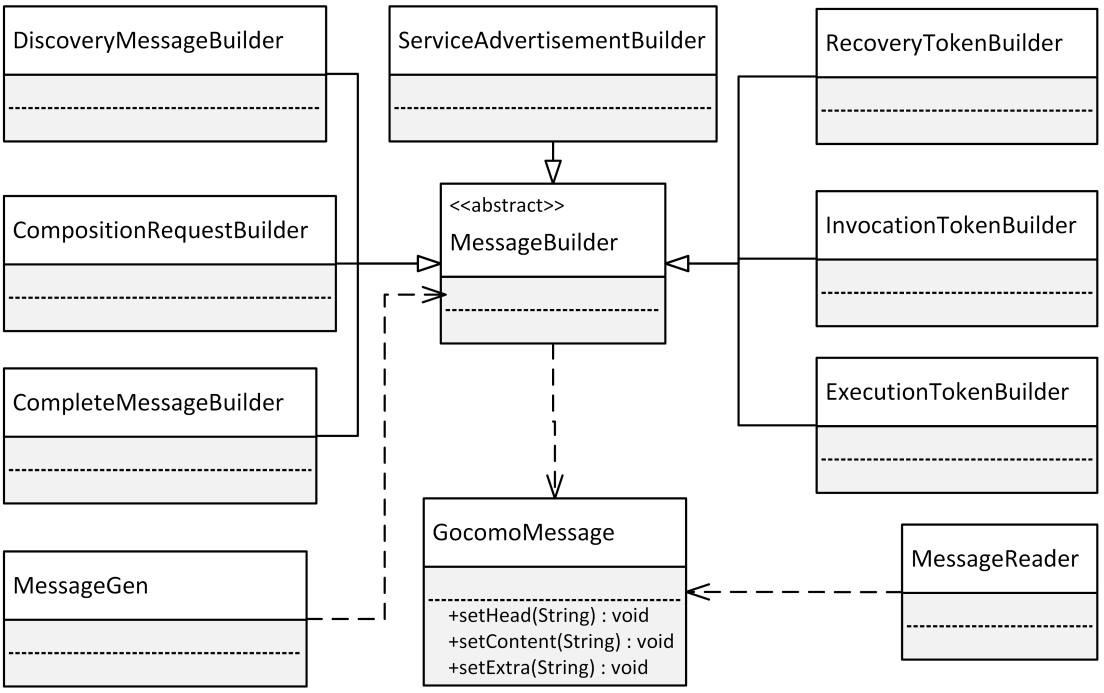


FIGURE 4.8: GoCoMo Messages Implementation: *class diagram of GoCoMo Message Helper*

4.6 GoCoMo Prototypes

Two GoCoMo prototypes were implemented to support a proof of concept. One is a middleware application on Android platforms and the other is an extension module on the NS-3 platform. Both of the prototypes realise GoCoMo composition processes, and are designed for the purpose of evaluating GoCoMo's feasibility and performance.

4.6.1 GoCoMo Prototype on Android

As one of the most popular mobile OS, Android has been installed in millions of mobile devices [Burnette, 2009, Butler, 2011, Gronli et al., 2014, Okediran, 2014, Pandey and Nakra, 2014]. More than 1.5 million applications are available in the Android application market (Google Play) by August 2015 [AppBrain, 2015], covering various categories like education, entertainment, business, health, travel& local, etc. This suggests considerable potential for widespread availability of services in pervasive environments. Research on Android-based services in pervasive computing environments has led to investigations on cooperating multiple Android mobile devices to support a composed functionality,

such as using smart phones and smart bracelets to enable mobile sensing for a health-care environments [Pigadas et al., 2011, Postolache et al., 2011]. The Android OS and Android applications require an effective tool to ease cooperation between devices and services.

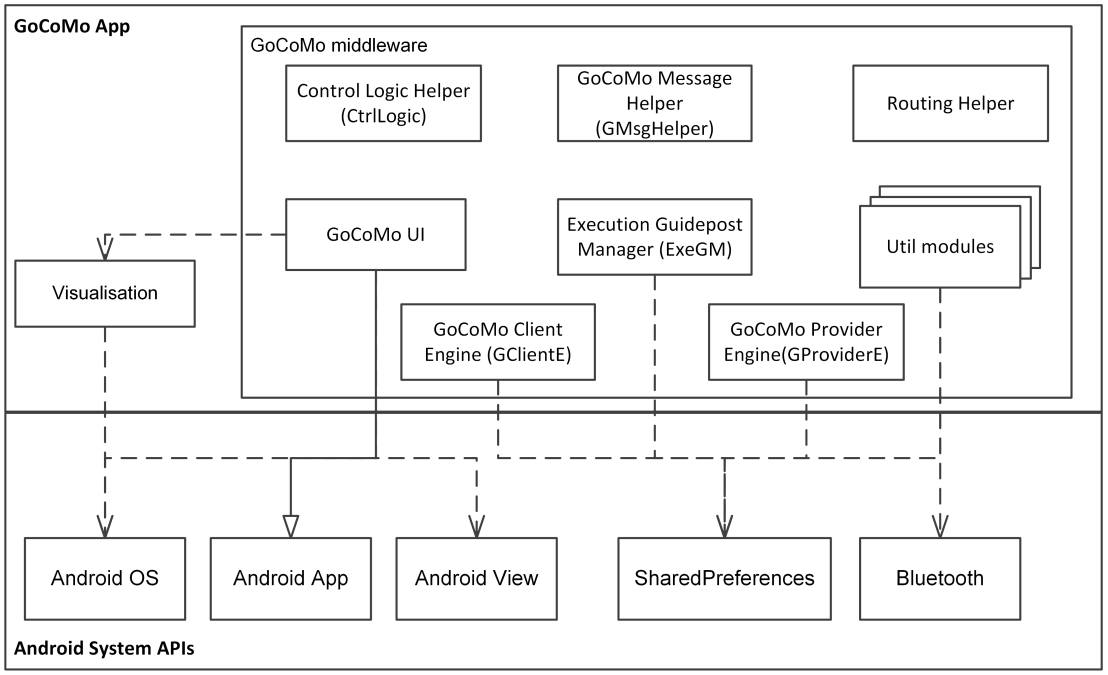


FIGURE 4.9: GoCoMo prototype on Android: *GoCoMo App*

The GoCoMo middleware prototype implemented for Android-based devices, GoCoMo App for simplicity, contains all the major modules introduced above, and supports GoCoMo composition processes. As for the utility modules in the GoCoMo middleware, GoCoMo App relies on syntactic matchmaking for Matchmaker, and realises the Data Manager and Service Manager. Implementing semantic matchmaking is out of this thesis's scope, and will be discussed in the future work chapter. Figure 4.9 briefly illustrates how GoCoMo App implements the GoCoMo middleware to slot its modules in the Android application framework. As GoCoMo App was designed for the purpose of evaluating GoCoMo in real world, in addition to the GoCoMo middleware, GoCoMo App realized a visualization module that renders GoCoMo App's real-time information that indicates how a GoCoMo composition process is performed at runtime. To enable a service provisioning network, GoCoMo App employed and implemented BlueHoc [Hinojos et al., 2014], a bluetooth-based ad hoc network for Android distributed computing. Execution guideposts generated during a GoCoMo composition process are serialised to JSON data and saved using the SharedPerference APIs provided in Android.

More details about GoCoMo App are in Appendix A.1. Note that GoCoMo App performs the GoCoMo composition process on composition clients' and service providers' devices with an Activity[1] running to display such a process's run-time information. This is of use when evaluating GoCoMo since it allows device owners to monitor their own GoCoMo App's performance. In real world scenarios, keeping a screen active is battery-consuming, which may reduce a device's availability, and so GoCoMo App should be extended to run in the background to reduce energy cost. In addition, GoCoMo App used a bluetooth ad hoc network. Bluetooth-based networks have very limited communication ranges, making them a less-competitive technique for wireless communication. A WiFi-based network is preferred because of its large communication range, but enabling WiFi ad hoc requires re-configurations on Android devices, for example, rooting a device or installing a third-party application. GoCoMo App does not go into the above issues as they are out of this thesis's scope, but these issues should be considered when deploying GoCoMo App for real world use.

4.6.2 GoCoMo Prototype on Ns-3

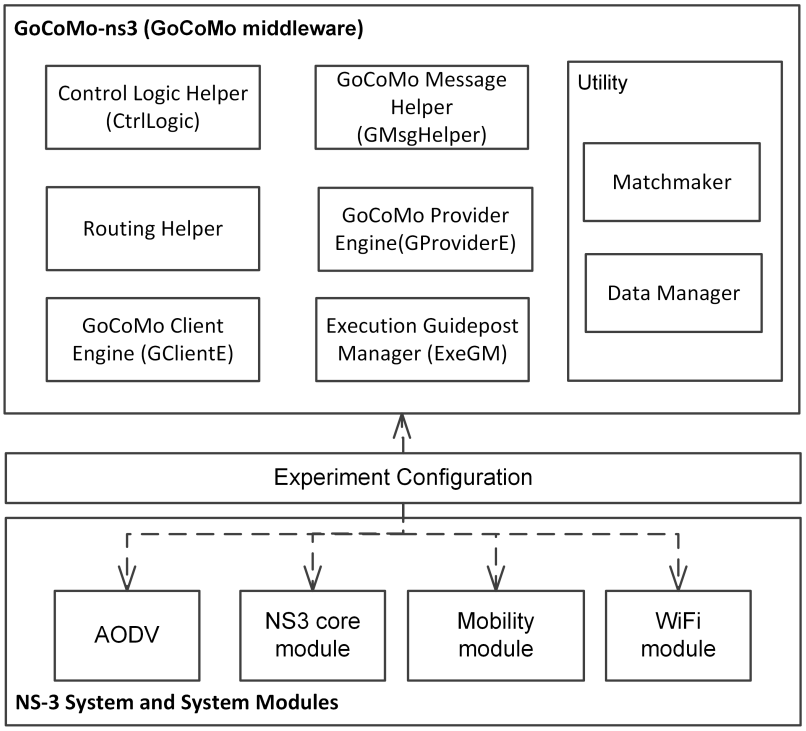


FIGURE 4.10: GoCoMo Prototype on NS-3: *GoCoMo-ns3*

[1]An Activity provides a screen with which users can get information or interact in order to modify the local service information or input a composition requirement.

Ns-3 is an open-source network simulator based on C++, providing an open, extensible network simulation platform [Ns-3, 2015]. A ns-3 extension model is a group of classes providing a specific sets of functionalities, which includes related classes, examples, as well as tests, and can be used with existing ns-3 modules/models [Ns-3, 2015]. This thesis describes a ns-3 extension model, named GoCoMo-ns3, build on top of NS-3 platforms, which realizes most of the GoCoMo middleware modules. In particular, GoCoMo-ns3 includes all the major modules of the GoCoMo middleware, and a part of the utility modules. The GoCoMo UI module and the Service Manager module are excluded, because the prototype is designed for evaluating GoCoMo's performance and feasibility in a service network, and does not involve actual services' execution and user interactions. Figure 4.10 illustrates how the GoCoMo middleware is added to the ns-3 platform as an extension model and how other ns-3 modules/models work with GoCoMo-ns3 to support a simulation of the GoCoMo composition process. Further details in GoCoMo-ns3 are presented in Appendix A2, and the experiment configuration will be introduced in the next chapter.

4.7 Implementation Summary

This chapter introduces the GoCoMo middleware and two prototype implementations on the Android platform and the ns-3 platform. The GoCoMo middleware lies between local services and the network on each composition participant.

The GoCoMo composition is mainly realized by a number of major modules, which include GClienE and GProviderE to coordinate actions across the global composition process and the local composition process, respectively. ExeGM and CtrlLogic are used to generate a service execution workflow, which may include complex control logic if necessary. In addition, GMsgHelper manages all the messages that are used in interactions between composition participants to exchange information. RoutingHelper controls such messages' transmission in a network. Utility modules and UI modules underpin GoCoMo composition processes by assisting the major modules to operate matchmaking or obtain context data. Their implementation may differ in varying platforms.

This chapter also describes two prototype implementations for the GoCoMo middleware that are designed for evaluation. The next chapter, based on these prototypes,

describes the evaluation of GoCoMo through NS-3-based simulation and a case study in an Android device network, and presents the evaluation results.

Chapter 5

Evaluation

The previous chapter introduced the GoCoMo middleware and the details of its implementation, which underpins the GoCoMo composition process proposed in Chapter 3. This chapter evaluates how well the GoCoMo approach addresses the target environment by satisfying the required features specified in Section 3.1. Three evaluation objectives are realised in this chapter:

1 to determine whether the GoCoMo middleware can flexibly reason about a service composite at runtime, in a decentralised manner, and self-organizes the composite's execution and adaptation.

2 to quantify the success of GoCoMo's service planning, heuristic request routing and composition adaptation model, in terms of planning and execution failure probability.

3 to evaluate the performance of GoCoMo, determining its feasibility in a range of mobile and pervasive networks/scenarios.

This chapter first outlines the evaluation method and a list of evaluation criteria, and then introduces a case study that achieved *evaluation objective-1*. After that, a simulation is used to address *evaluation objective-2 and -3*. The GoCoMo composition process is examined under different scenarios in mobile and pervasive environments. The validity of the evaluation results is discussed in B.

5.1 Evaluation Methods and Criteria

Given the challenges outlined in this thesis (see Section 1.1), a usable service composition model for pervasive computing should be able to tackle infrastructure-less networks, providing a dynamic, decentralized service composition process. The service composition process itself should be sufficiently time-efficient and adaptable to reduce composition failures caused by changes to the network and the service topology. This thesis used a simulation study and a prototype case study to evaluate GoCoMo.

A case study is an observation-based method that investigates a single entity's activity within a specific time space [Flyvbjerg, 2006, Wohlin et al., 2003a,b], and can be used to evaluate the benefits of methods and tools [Kitchenham et al., 1995]. Prototype case studies relying on real-world pilot implementations can reflect the behaviour of a model in the real world scenario [Kiess and Mauve, 2007], based on which, the model's feasibility, reliability or flexibility can be investigated [Wohlin et al., 2003a]. Many service composition models and composite adaptation strategies proposed to support service provision have adopted prototype case studies [Bucchiarone et al., 2010, Mateescu et al., 2008, Mostarda et al., 2010, Newman and Kotonya, 2012, Poizat and Yan, 2010].

This thesis built a testbed that relies on Android platforms, and deployed a pilot implementation of GoCoMo on the testbed to conduct a prototype case study. Testbeds, sometimes called experimentation networks, are in-lab networks established and used by researchers [Hogie et al., 2006]. In general, testbeds have limited scalability in terms of network size because of the cost of hardware and the monitoring/deployment difficulties, but capture more aspects that influence the performance of algorithms and protocols comparing to software-based simulators [Kiess and Mauve, 2007]. The prototype case study focuses on the following evaluation metrics:

1. Support for pervasive environments where previously cached conceptual composites are impossible (Challenge 1).
2. Support for infrastructure-less networks and fully decentralized service discovery and execution (Challenge 2)
3. Support for service composite adaptation when the operating environment is dynamic and open (Challenge 3, 4 and 5).

In the domain of decentralized service composition that assumes multiple composition planners, the network will include a number of real devices that have enough resources to perform a local planning process. Using a number of real devices is always impractical in laboratory-scale evaluation studies, making it difficult to demonstrate a model's performance in a scalable network (i.e., where the number of nodes is equal to or above 20). Many researchers in the pervasive computing domain, especially those have the particular focus on mobile ad hoc networks, consider simulation studies as a helpful evaluation method. The majority of them used only simulations, and several others combined a small scale (using 5-25 nodes) prototype case study and a simulation study to evaluate their approach [Liu et al., 2015c].

Simulation studies that take a network simulation as a controlled experiment have been widely used in service-oriented computing research [Efstathiou et al., 2014a, Kim et al., 2006]. A simulation-based experiment models an algorithm with a high degree of abstraction and performs it in an artificial software environment, which makes the experiment effective, repeatable, controllable, and scalable [Breslau et al., 2000, Kiess and Mauve, 2007]. Such an experiment relies on the simplification of real-world scenarios, manipulating a series of environmental or systemic variables to get an algorithm's performance values in diverse scenarios or under different system configurations. [Wohlin et al., 2003a,b].

This thesis quantified GoCoMo's composition failures and performance, to demonstrate GoCoMo's advantages in a set of scenarios through a simulation-based experiment. This experiment used the following evaluation criteria:

4. Planning failure rate: GoCoMo'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.

5. Execution failure rate: GoCoMo'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 executions to the number

of all the successful planning processes, 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.

6. The failure rate for composing parallel service flows: GoCoMo's success at finding and invoking parallel solutions. Composition failure rates were calculated under different service availability configurations.

5.2 Prototype Case Study

The prototype case study deployed the GoCoMo middleware in a real world pervasive network, and discovered the feasibility of GoCoMo's *decentralised backward planning algorithm* and *adaptation mechanism*. The Android-based prototype implementation is named GoCoMo App and was introduced in Section 4.6.1 and Appendix A.1.

Testbeds have been used in the MANET domain, and usually realize a network containing less than 50 nodes [Hogie et al., 2006]. With the increasing number of advanced but expensive mobile devices to be used to support pervasive environments, many researchers have built testbeds on a network involving only a small number (< 10) of such devices [Hiyama et al., 2012, Kumar and Sam, 2015] to evaluate algorithms or protocols in MANETs.

This thesis developed a testbed with 8 mobile Android devices marked as Device 0 to 7. Table 5.1 presents the basic information about the devices and their configurations. The GoCoMo App was deployed on the Android OS with API level from 19 to 22, running across a range of mobile devices. The devices have diverse computing power ranging from a CPU speed of 2 GHz and 1GB RAM to a Quad-core 2.5GHz CPU and 3GB RAM, which covered most mainstream manufacturer brands and best-selling Android devices. The GoCoMo App was developed to visualise the GoCoMo composition process at runtime, and the dynamic system performance on each participated device was monitored by a Android Device Monitor[1] embedded in Android Studio IDE. Figure 5.1 illustrates all the devices used in the study and the testbed system in operation.

[1]http://developer.android.com/tools/help/monitor.html

TABLE 5.1: Devices used in case studies (M.M: Manufacturer and Model Number, OS: Android OS version)

Device	OS	API	M.M	CPU	RAM
0	5.1.1	22	Nexus 7 (2013)	Quad-core 1.5 GHz	2 GB
1	5.1.1	22	Nexus 7 (2013)	Quad-core 1.5 GHz	2 GB
2	4.4.2	19	SM-N9006 (Galaxy Note 3)	Quad-core 2.3 GHz	3 GB
3	5.0.2	21	Motorola G-2	Quad-core 1.2 GHz	1 GB
4	4.4.2	19	SM-N9005 (Galaxy Note3)	Quad-core 2.3 GHz	3 GB
5	5.1.1	22	OnePlus One	Quad-core 2.5 GHz	3 GB
6	4.1.2	16	Motorola RAZR i	2 GHz	1 GB
7	5.1.1	22	SM-G920F (Galaxy S6)	Quad-core 2.1GHz	3 GB

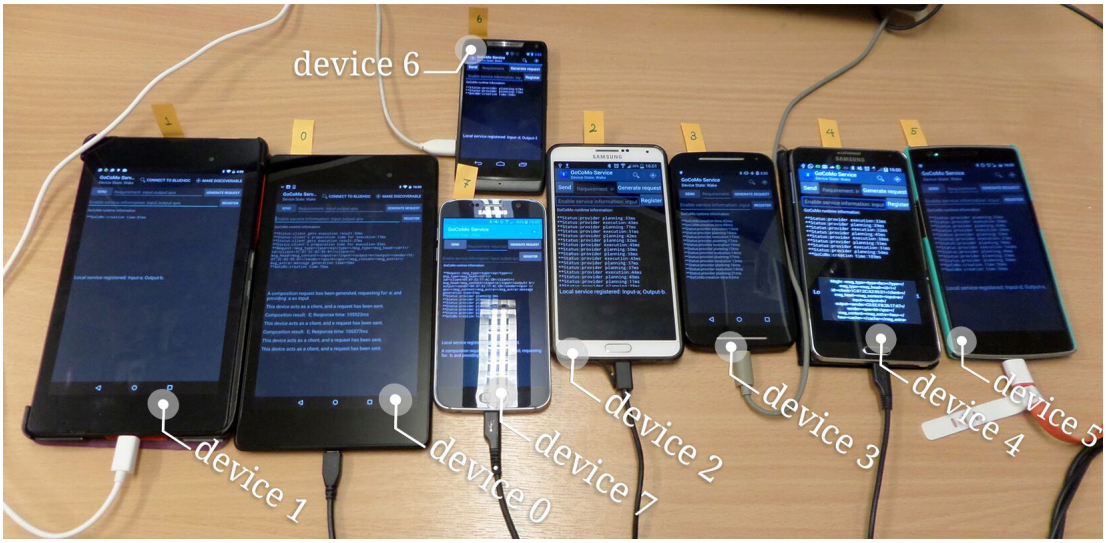


FIGURE 5.1: All the devices used in the study, and the testbed system in running

5.2.1 Case Study Configurations

This case study adopted 8 devices, 7 of them (device 1-7 in Table 5.1) were service providers and one (device 0 in Table 5.1) was the composition client. They all installed the same version of the GoCoMo App. These devices were connected to a desktop that ran the Android Device Monitor. To demonstrate the feasibility of GoCoMo, a set of scenarios were used, varying in composition complexity and service availability. The scenarios used in the study are described in Table 5.2.

The composition complexity was determined by services available in a network. Similar to those used in [Chen and Clarke, 2014, Kalasapur et al., 2007], the thesis adopted services that convert alphabets. More specifically, a service $s_{A \to B}$ can produce an output

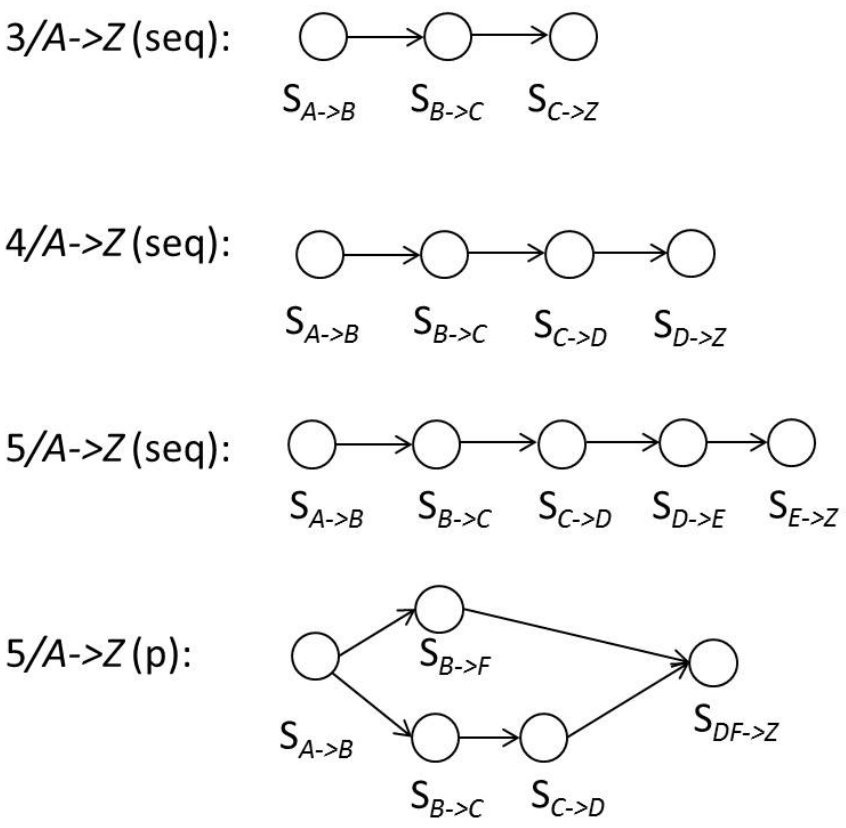


FIGURE 5.2: Service scenarios used in the case study

of type B if it receives an input of type A [Kalasapur et al., 2007], and is named *linear-service*. A service $s_{DF \to Z}$ can produce an output of type Z if it receives two inputs of type D and F [Chen and Clarke, 2014], and is named *join-service*. The thesis used 8 unique linear-services and 1 join-service in 8 different scenarios (see Table 5.2) in each of which a subset of these services were deployed on the 7 service providers. Every service subset supports a particular service flow to transfer data A to Z. The service flows for the 8 scenarios are shown in Figure 5.2. An individual service provider hosts only one service, and a copy of a service can be deployed on one or more providers. The case study achieved diverse composition complexities through different service flows for the same composition goal, which was for services that can support alphabet transformations from A to Z. This case study used 4 service flows varying in composition length (the number of service instances per request).

Service providers' mobility is one of the most important factor that causes changes in service availability. However, mobility models are difficult to apply in a real world implementation. The case study used a Wake/Sleep pattern for service providers to

TABLE 5.2: Scenarios for the case study, (seq) = sequential service flow, (p) = parallel service flow, P = probability of *Wake state*, D = duration of *Wake/Sleep state*

		Scenario							
Parameter	**Configuration**	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8
Services	$s_{A \to B}$	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
	$s_{B \to C}$	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
	$s_{C \to D}$		⋆	⋆	⋆		⋆	⋆	⋆
	$s_{D \to E}$			⋆				⋆	
	$s_{C \to Z}$	⋆				⋆			
	$s_{D \to Z}$		⋆				⋆		
	$s_{E \to Z}$			⋆				⋆	
	$s_{B \to F}$				⋆				⋆
	$s_{DF \to Z}$				⋆				⋆
Service flows	$3/A \to Z$ (seq)	⋆				⋆			
(service instances	$4/A \to Z$ (seq)		⋆				⋆		
per request	$5/A \to Z$ (seq)			⋆				⋆	
/composition goal)	$5/A \to Z$ (p)				⋆				⋆
Service availability	P=1 (static)	⋆	⋆	⋆	⋆				
	P=0.8, D=1s					⋆	⋆	⋆	⋆

manage their availability. Every service provider has two states: *Wake* and *Sleep*, and the default state is *Wake*. During the *Wake* state, the service provider can process GoCoMo messages and service requests. When the *Sleep* state gets activated, the service provider freezes any composition process it participates in, and stops its communication with any other entities in the network. In other words, during the *Sleep* state, the service provider is inaccessible, but still keeps the cached data of composition processes. Each state's activation and duration are determined only by the local device, according to a predefined probability and random variables. As presented in Table 5.2, this study used static a network that keeps all service provider wake and a dynamic network that allows service providers to periodically (period(D)=1s) "flip a coin" to decide if they remain to their current state or activate the other state. The possibility for a "Wake state" is 0.8 (P=0.8).

In the above scenarios, no centralized knowledge base or central composition controller was used. In particular, when a GoCoMo system gets initialsed, each service provider only has information about its own service, which is the only knowledge that can be used at the start of a local service composition process. Such knowledge is expanded through interactions with other participants during composition processes. For example, by joining a testbed network a participant can obtain a list of addresses of its direct

neighbours, and by receiving a GoCoMo discovery message, a participant can get the path length of discovered composites.

5.2.2 Samples and Results

The case study evaluation measured observed the performance of GoCoMo from each individual mobile device's point of view. These measurements contain observations of two service composition cases. The *composition planning case* adopted Scenario 1.1-1.4, using evaluation metric 1 and 2 to assess whether the proposed backward planning algorithm can support complex composition planning in infrastructure-less networks. The number of failed planning-based discoveries out of 50 composition attempts were counted in each experiment. The *adaptation case* applied evaluation metric 3 in Scenario 1.5-1.8, comparing against a service composition model with no adaptation support, to assess whether the proposed adaptation mechanism can reduce execution failure in a dynamic environment. The case study also assessed GoCoMo's maximal CPU usage and memory usage on different mobile devices.

5.2.2.1 Composition Planning Case

GoCoMo's feasibility in a network that is infrastructure-less and requires flexible composition planning were demonstrated from the perspectives of both composition clients and service providers. A composition client measured the overall discovery and execution failures as well as their response time, and the service providers that participated in a composition process measured the time spent on local composition planning and execution.

The composition client initialised a time $T_{discovery}$ for the composition discovery process after a composition request was sent, and the client detected and recorded a discovery failure if no composition complete token has been received when $T_{discovery}$ expired (see Section 3.4.2 on page 60). Execution failures were measured in a similar way and used an execution time T_{exe}. The response time for service discovery was the duration from the time a client issues a composition request to the first discovery result being returned to the client, and the response time for service execution was the duration from the time a composition invocation token is sent to receiving a composition result.

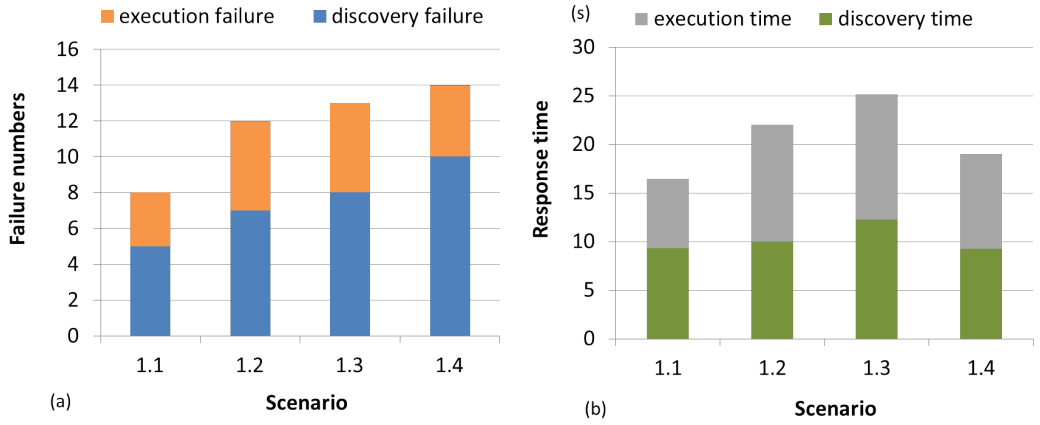


FIGURE 5.3: GoCoMo's feasibility on static networks: *(a) discovery and execution failures out of 50 attempts on a static network, (b) response time for the composition discovery process and the service execution process*

Figure 5.3(a) illustrates the number of GoCoMo's discovery and execution failures out of 50 attempts on a static network, and (b) shows the response time. The results show that GoCoMo was able to compose and execute different service flows depending on existing services using the same composition goal. It was difficult to avoid failures given the use of unreliable bluetooth-based communication channels and the limited number of service providers. The response time was decoupled, and represented by the discovery time (the green parts in Figure 5.3(b)) and the execution time (the grey parts). The discovery time was in range of 9.241-12.248 seconds. It was much beyond the tolerable value for a simple information query of 2 seconds [2]. However, this range is around the upper limit of mobile users' acceptable waiting time for applications, approximately $7 - 12$ seconds [Niida et al., 2010]. Similarly, the length of execution was in range of 7.142-11.992 seconds, and did not exceed the tolerable value for a task operation, 15 seconds [3].

Table 5.3 illustrates the performance of GoCoMo on each individual service provider. The build time was the time spent on initialising the GoCoMo App, which varis depending on the deployment device's computation power. The planning time was the duration of a local composition planning process starting from receiving a composition request and ending when a fragment of an execution path (a direction in the guidepost,

[2] A tolerable waiting time for a simple information query is about 2 seconds [Nah, 2004].
[3] For operating tasks, the waiting time should be within 15 seconds [Miller, 1968].

TABLE 5.3: Service provider's time consumption (*ms*) on each step in the GoCoMo service composition process, the average number of sent messages and the average size (*byte*) of them.

Scenario	Device	Build	Planning	Execution	Msg	Msg size
1.1	2	124	42	34	2	227
	3	63	40	31	2	227
	4	130	51	39	2	227
1.2	2	118	45	34	2	227
	3	58	43	36	2	227
	4	136	52	41	2	227
	5	80	50	29	2	227
1.3	2	121	41	30	2	227
	3	48	42	39	2	227
	4	131	47	36	2	227
	5	42	70	45	2	227
	6	130	40	72	2	227
1.4	1	52	24	47	2	264
	2	128	45	29	2	227
	3	46	55	32	2	227
	4	125	40	23	2	264
	5	45	38	49	2	227

see Section 4.4.1 on page 92) is generated. A device started timing the execution process right after receiving a service invocation message, and stopped before sending out a message to invoke the next service. Msg represents the number of messages generated and transmitted from the device during composition, and Msg size represents the average size of these messages. The results show that the local composition process itself is efficient as the average performance time (time spent on planning and execution) for every device is small ($< 200ms$). As the evaluation network was established in ad hoc mode, establishing message transmission channel was slow, which causes the high global response time (Figure 5.3(b)). Service providers only sent 2 messages, and the messages' size were 227 (*byte*), on average. In scenario 1.4, device 1 and 4 generated bigger messages, 264 (*byte*) on average. They hosted service $s_{C \to D}$ and $s_{B \to F}$, respectively, and received discovery messages from the device who provided join-service $s_{DF \to Z}$. None of them can satisfy the goal of data D and F independently, and so device 1 and 4 cached information about the unfinished goal in their sending-out discovery messages. As a result, they sent bigger messages.

Figure 5.4 illustrates the overall performance of GoCoMo in different scenarios, where (a) Max. CPU usage represents the GoCoMo App's maximal CPU usage in composition processes, and (b) Max. memory usage represents the maximal memory usage. The

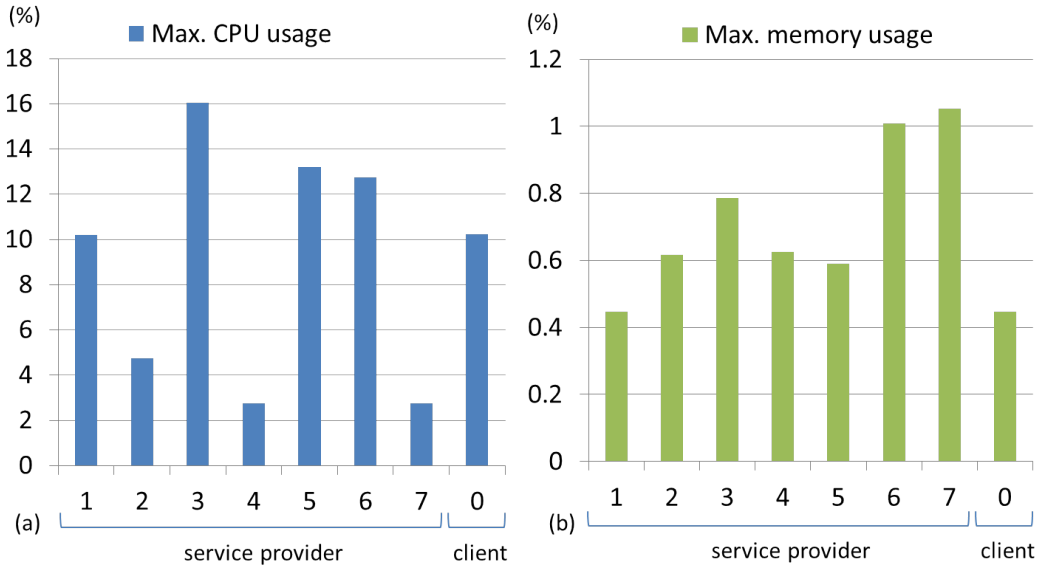


FIGURE 5.4: GoCoMo's maximal CPU usage and maximal allocated memory on different devices

results show that GoCoMo's CPU usage was below 16.04%, and was even smaller on devices that had more computing resources. The most RAM-costly process occurred on device 6 (Motorola RAZR i) and 7 (SM-G920F (Galaxy S6)). This is because that Motorola RAZR i has limited resources comparing to the rest of devices, and SM-G920F's processor (Galaxy S6) trades off CPU usage to RAM usage. Although they were less efficient, their memory usage were only very small portion of the total, 1.1%. On the other hand, the GoCoMo App implemented GoCoMo's service provider and client modules in a integrated architecture. They have the potential to be implemented as adjustable modules, by which a pure service provider does not have to load the modules that support a composition client's behaviour to reduces resource consumption on resource-constrained devices.

5.2.2.2 Adaptation Case

The adaptation case compared GoCoMo's adaptable composite against a service composition model using static composites, and measured composition response time and failure numbers. The results are shown in Figure 5.5. Overall, the failure rate for the two approaches increased with longer composites and more complex control logic. In addition, comparing to the results illustrated in Figure 5.3(a), GoCoMo produced more failures in scenarios (1.5-1.8) with dynamic service availability, but GoCoMo had less

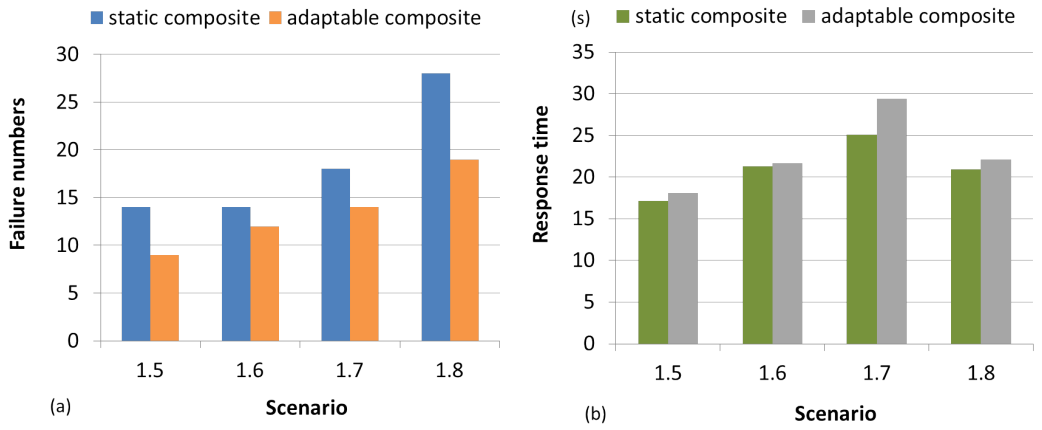


FIGURE 5.5: GoCoMo's feasibility on on dynamic networks: *(a) discovery and execution failures out of 50 attempts on a static network, (b) response time for the composition discovery process and the service execution process*

failures than the static composite approach. The static composite approach's response time in scenario 1.5-1.8 was similar to that of GoCoMo in scenario 1.1-1.4 (Figure 5.3(b)). However, GoCoMo, in scenario 1.5-1.8, took longer to return a composition result than the static composite approach. This is because GoCoMo adapts the execution path when a selected one becomes unavailable, which requires re-invoking services, taking extra time.

This case study has demonstrated how GoCoMo addresses this thesis's challenges by supporting dynamic planning and self-organised service composition. The composition planning case showed that GoCoMo can reason about different service flows according to services available in the network, and the composition response time was acceptable. Given a slow messaging channel was used in this case, the response time has the potential to be further reduced by employing an advanced wireless protocol with fast transmission like WiFi or UWB[4].

As the case study adopted a small scale scenario, and a short distance transmission technology, this study did not include an analysis on GoCoMo's scalability or an investigation into GoCoMo's heuristic service discovery model. Moreover, the ad hoc network used in this study is static, so only unanticipated service availability was modelled, leaving the issue of dynamic network/service topology to be explored [Chen and Clarke, 2014]. The global composition performance of GoCoMo in environments with dynamic

[4]A transmission on WiFi channel and on UWB channel can be about 75 times an 154 times faster than that on a Bluetooth channel, accordingly [Lee et al., 2007].

Table 5.4: Simulation Configuration: General

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) [Klusch, 2012]
Composition discovery	1-hop broadcast
Service routing	dynamic AODV 10-hops
Sample	300 runs
Random	
Node placement	Random
Service execution time	0.01-0.1 (s)
Node movement	random walk mobility model

network/service topology and the large scale scenarios are analysed using a simulation study, and presented in the next section.

5.3 Simulation Studies

Software-based simulations have been widely accepted in MANET research as an evaluation method [Hogie et al., 2006]. Such simulations are low-cost, scalable and capable of modelling devices' different mobility behaviours.

5.3.1 Environment Configurations

This thesis used ns-3 platform for its simulation, and deployed GoCoMo-ns3 (see Section 4.6.2 and Appendix A.2) in a system with Intel Core i7-2600, CPU 3.4GHz, 8G RAM, running the Ubuntu 12.04.5 Desktop(32-bit).

5.3.1.1 General Settings

The experiment setting for this simulation were chosen according to recommendations [Kotz et al., 2004] for MANETs. Specifically, the simulation used AODV[5] for service routing and, for simplicity, considered only 2-dimensional 1000*1000 (m^2) terrain. The other settings like the number of nodes and communication distance refer to state of the

[5]See Ad-hoc on-demand distance vector routing in [Perkins and Belding-Royer, 1999]

art research for MANETs [Groba and Clarke, 2014, Li et al., 2015], as shown in Table 5.4.

Mobility Model and Node Topology. In the simulation, a random position is assigned to each node when an experiment gets initialised. The movement of a node during the experiment is controlled by a 2D random walk mobility model. In this mobility model, a node's movement is determined by a previously assigned speed as well as a constant time interval or a constant travel distance. At the end of the time or the distance, the network simulator calculates a new direction and speed for the node. All the nodes created in the simulation communicate using WiFi 802.11b in ad hoc mode. The communication range(distance) used in the simulation refers to outdoor WiFi communication distance, up to 250(m).

Services. The simulation creates a number of nodes during initialisation. One of them is a service client, and the rest are service providers. The simulation assumed a network with a low composition demand (one composition request each run), since the current version of GoCoMo does not provide a mechanism that addresses invocation failures caused by multiple composition processes competing for one service provider. Each service provider hosts one service, which is assigned a random execution time ranging from 0.01(s) to 0.1 (s). The number of and the type of services are scenario-specific. Every individual evaluation scenario contains a number of different services and their duplicates, one per service provider (node). The simulation used alphabet transformation services that are similar to that were used in the case study introduced in Section 5.2.1. The simulation adopted a fixed semantic matchmaking delay [Klusch, 2012] to minimise the variance performance of the GoCoMo composition process caused by different delays, easing the measurement of response time.

5.3.1.2 Evaluation Scenarios

GoCoMo was simulated with configurations that differ in their service density, node mobility and the complexity of service composition. These configurations included a set of controlled variables that define 4 different scenarios. Table 5.5 illustrates these scenarios marked from 2.1 to 2.4. The two columns on the left represent configuration parameters and their values.

Environmental configuration. The simulation defined service density ranging from 20 to 50 nodes and mobility including speed intervals 0-2 (human walking), 2-8 (slow vehicles) and 8-13 (motor vehicles) m/s for random value taking. The type of service flows are the potential order and structure of discovered services that may be formed by a composition planning model according to these services' I/O dependency. Sequential service flows are the basic structure of a service composite, and contains only linear-services. Parallel and hybrid service flows include join-services, and are complex because of their control logic, but need to be flexible in some cases like when aggregation of data from different sources is needed. The simulation used different sets of services. For example, consider a scenario consisting of 5 different services, where a copy of each service can be deployed on one or more nodes in the network. In such a system, service composition could entail anything from 2-5 services involving multiple alternatives for each service.

The number of service instances per request refers to the complexity of a requested service composite, defined in composition requests. This simulation measured 5-service-instance composites and 10-service-instance composites in a scenario consisting of 10 different services. It also measured 5-service-instance composites in parallel execution flows in scenarios consisting of 5-15 different services. GoCoMo defines a composition request using input data and goals rather than conceptual composites (see Definition 2 on page 60) and assumes a client has no knowledge about the service availability in its local environment. So generally, a client cannot foresee the final composed service composite's complexity at the beginning (i.e, when issuing the request). The simulation controlled such complexity by initialising a service network containing a particular set of services, and measured network traffic and response time when resolving service composition in different complexities.

System configuration. To enable separate measurements on the performance of GoCoMo's composition planning model, execution model, and heuristic discovery model, the simulation also included configurations on GoCoMo itself. GoCoMo's heuristic discovery model (see Section 3.4.3) uses a value to represent the model's interference degree (see d in Equation 3.5 on page 73), which indicates the model's influence on request flooding, and is determined by local network properties (e.g., the number of direct neighbours). In general, all the service providers (nodes) cannot have the same local network properties, e.g., equal in the number of neighbours, since the nodes are owned

Table 5.5: Scenarios for the simulation

Parameter	**Configuration**	**Scenario** 2.1	2.2	2.3	2.4
Environmental configuration					
Service density	20 (sparse)	⋆	⋆	⋆	⋆
	30 (medium-dense)	⋆	⋆	⋆	⋆
	40 (dense)	⋆	⋆	⋆	⋆
	50 (extra-dense)	⋆	⋆		⋆
Mobility (m/s)	0-2 (slow)	⋆	⋆	⋆	
	2-8 (medium-fast)	⋆	⋆	⋆	⋆
	8-13 (fast)	⋆	⋆	⋆	
Type of service flows	sequential	⋆	⋆	⋆	⋆
	parallel				⋆
	hybrid				⋆
Type of services	5				⋆
	10	⋆	⋆	⋆	
	15				⋆
Service instance per request	5 (simple request)	⋆	⋆		⋆
	10 (complex request)	⋆	⋆	⋆	
System configuration					
Service composition	composition planning	⋆	⋆	⋆	⋆
	execution		⋆	⋆	⋆
Heuristic level (interference degree)	5 (strong interference)			⋆	
	4			⋆	
	3			⋆	
	2			⋆	
	1 (weak interference)			⋆	
	0 (no interference)	⋆	⋆	⋆	⋆

by third-parties, not distributed evenly, and dynamically change their positions. Service providers in the same network are likely to have different interference degrees, and the interference degree for each service provider changes over time. This simulation simplified this, and made all the service providers share the same interference degree to evaluate how varying interference degrees affect the GoCoMo composition model in different environments. This study used 6 degrees of interference signed from 0 to 5, and degree 0 means that GoCoMo has no interference on flooding, which means the request flooding is uncontrolled. Degree 5 represents the highest interference degree.

Scenario 2.1 Composition length, mobility and network density's influence on the flexibility of service planning

This scenario investigates how the composition length, mobility and network density impact the flexibility of service planning. This scenario used 10 different services, two

levels of request complexity: 5-service and 10-service composite, and sequential service flows. GoCoMo is configured to support composition planning and using uncontrolled request flooding. In this scenario, planning failure rate, the number of sending messages and response time were measured.

Scenario 2.2 Composition length, mobility and network density's influence on the flexibility of service execution

This scenario explores how the composition length, mobility and network density impact the flexibility of service execution. This scenario has the same environmental configuration as Scenario 2.1, and configures GoCoMo to support a full functionality of service composition including composition planning and execution. Similar to Scenario 2.1, its request flooding is uncontrolled.

Scenario 2.3 Impact of heuristic service discovery

This scenario investigates how GoCoMo's different interference degrees affect the flexibility of the composition discovery process. This scenario used complex requests since complex requests lead to more service providers participating in the request resolving and in turn are likely to generate more discovery messages. Controlled flooding is designed to reduce message transmission, which in turn reduces message loss caused by packet collision[6]. This scenario employed all 6 interference degrees to demonstrate how the heuristic service discovery affects the failure rate and the system traffic in a set of environments.

Scenario 2.4 Impact of environment including complex service flows

Composites containing different service flows may increase the chance of more service providers being used to solve a request. It may also improve the quality of the final result were used. This scenario explores the impact of environments including complex service flows.

[6]See Section 5.3.3.2 for packet collision

Table 5.6: Comparison of the baseline CoopC with proposed GoCoMo

	CoopC	GoCoMo
Composition planning	Traditional backward	Backward supports parallel
Service invocation	Decentralised	Decentralised
Request routing	Broadcast flood	Heuristic broadcast
Dynamic binding	QoS-driven	On demand
Fault tolerance	N/A	Dynamic recovery

5.3.2 Baseline approach

For the proposes of establishing a good baseline against which to compare GoCoMo, this study has combined state-of-the-art functionality. In particular, a decentralised co-operative discovery model [Furno and Zimeo, 2013] combined with a continuing message passing model [Yu, 2009] that enables decentralized service invocation. This baseline approach is referred to as CoopC in the following sections. Note that there are service composition approaches [Groba and Clarke, 2014, Kang et al., 2008, Pirrò et al., 2012, Prinz et al., 2008] related to this research, but the simulation did not consider them as baseline approaches because their composition discovery processes are workflow-driven, which is inflexible (see Section 2.3.3 on page 34).

CoopC's cooperative discovery employs a traditional 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 run-time service composite adaptation. Unlike the cooperative discovery model used as an input to CoopC [Furno and Zimeo, 2013, 2014], this study has 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. In addition, the latest version of CoopC assumes that a semantic service overlay network is previously cached to facilitate service discovery [Furno and Zimeo, 2014]. Given the infrastructure-less nature of our target network, this study went for an early version of CoopC [Furno and Zimeo, 2013] that uses request flooding for service discovery instead of the overlay infrastructure. CoopC's service execution was based on a decentralised message-passing model [Yu, 2009] that passes an invocation message from one service to its successive service. Yu [2009] recovers a failed execution process by giving the failed service a second attempt, which is controlled

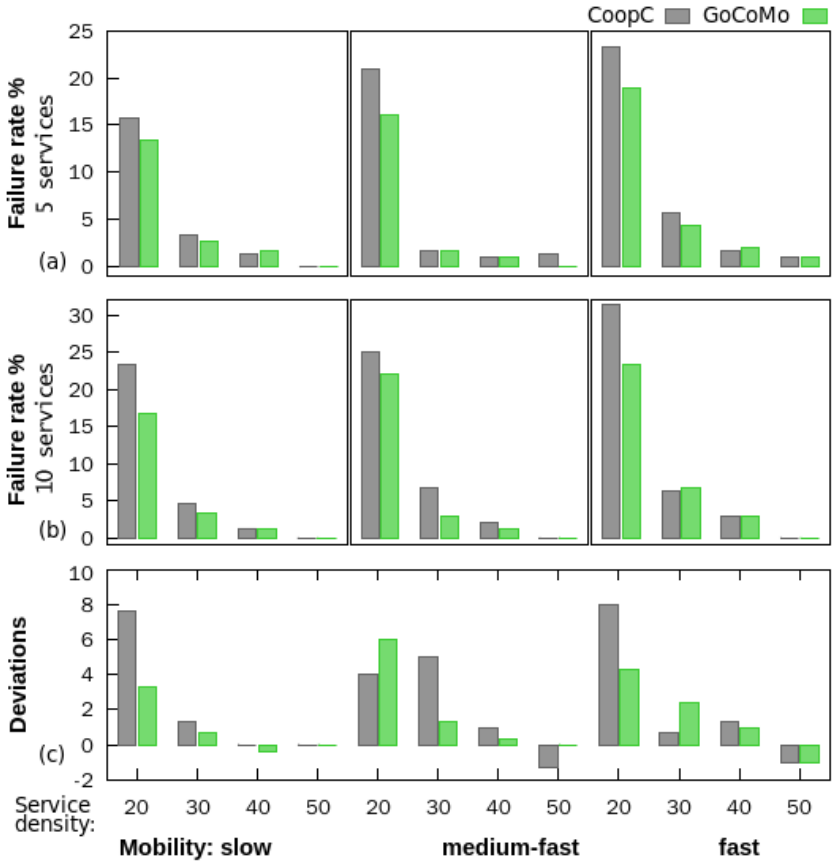


FIGURE 5.6: Planning failure rate in mobile networks: *(a) 5 service instance per request, (b) 10 service instance per request, and (c) the failure rate deviation between (a) and (b)*

by a scope manager that maintains run-time status information about the services in its responsibility scope. CoopC implemented this execution model [Yu, 2009] in an infrastructure-less network, so such a scope manager and the retry-based failure recovery was not adopted. Table 5.6 illustrates the difference and similarity between CoopC and GoCoMo.

5.3.3 Simulation Results and Analysis

5.3.3.1 Flexibility of Service Planning

This study applied scenario 2.1, and measured planning failure rate in the simulated target environment, to quantify the flexibility of service planning. The failure rate and the GoCoMo composition planning model's performance are illustrated in Figure 5.6, Figure 5.7 and Figure 5.8. On finding pre-execution plans, GoCoMo shows a higher

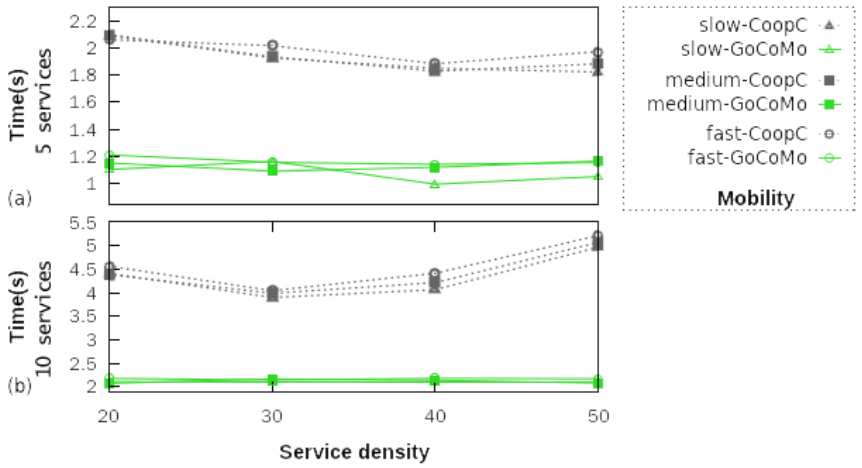
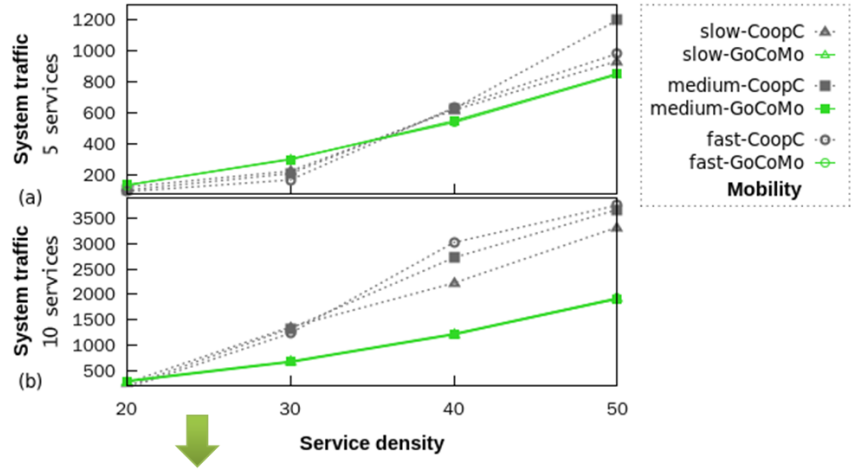


FIGURE 5.7: The mean and standard deviations for the discovery time in mobile networks: *(a) 5 service instance per request, (b) 10 service instance per request*



GoCoMo's system traffic:

	Slow mobility		Medium-fast mobility		fast mobility	
	5 services	10 services	5 services	10 services	5 services	10 services
20 providers	140	299	139	294	134	283
30 providers	324	701	323	701	321	704
40 providers	575	1268	586	1264	577	1272
50 providers	898	1996	902	1984	904	1993

FIGURE 5.8: The mean and standard deviations for the discovery traffic in mobile networks: *(a) 5 service instance per request, (b) 10 service instance per request*

possibility of returning a pre-execution plan than CoopC (Figure 5.6(a) and (b)). In particular, with the increasing complexity of service composition (Figure 5.6(c)), GoCoMo raised about $0-6\%$ failures while CoopC raised approximately $0-8\%$ failures in most of the scenarios. The results also show that GoCoMo discovery spent less time than CoopC to return the first pre-execution plan (Figure 5.7), but it sent slightly more messages than CoopC's discovery model (Figure 5.8(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 in some scenarios as it discovers more service links to find various possible execution paths for a single pre-execution plan.

5.3.3.2 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, scenario 2.2 was used, and 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 (Figure 5.9) 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 Figure 5.11) 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 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[7]. Frequent interactions in a network increases the possibility of high packet collision failures Lipman et al. [2009] Jun et al. [2010]. Therefore, CoopC

[7]Service execution time interval $[0.55, 5.3]s$ is calculated by removing the time spent on discovery (in Figure 5.7) from the response time (in Figure 5.10).

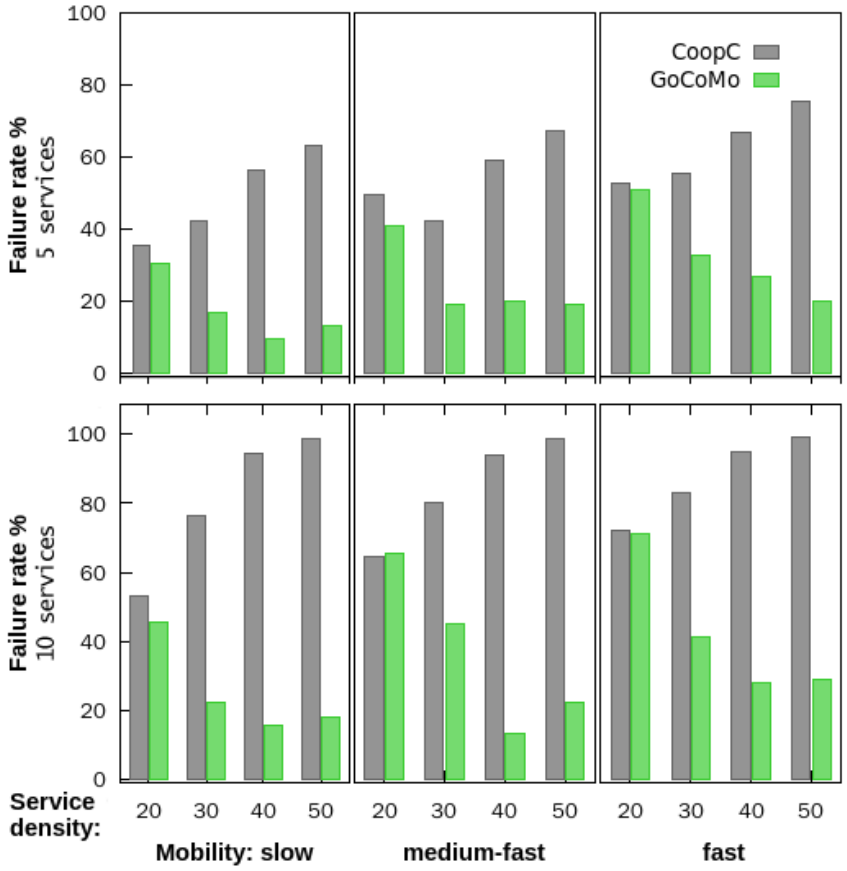


FIGURE 5.9: Execution failure rate in mobile networks: *(a) 5 service instance per request, (b) 10 service instance per request*

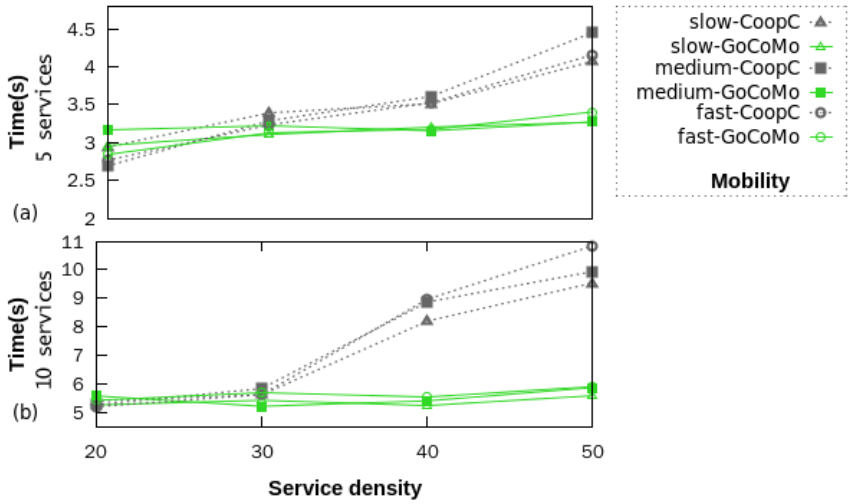
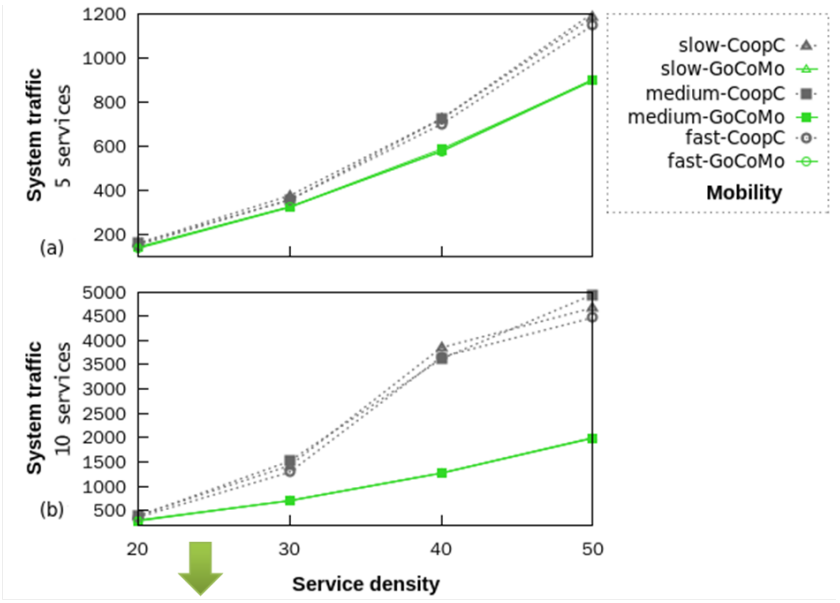


FIGURE 5.10: The mean and standard deviations for the response time in mobile networks: *(a) 5 service instance per request, (b) 10 service instance per request*



GoCoMo's system traffic:

	Slow mobility		Medium-fast mobility		fast mobility	
	5 services	10 services	5 services	10 services	5 services	10 services
20 providers	274	465	274	468	272	460
30 providers	497	808	523	802	520	808
40 providers	1112	1379	1137	1375	1121	1382
50 providers	1744	2213	1754	2201	1760	2224

FIGURE 5.11: The mean and standard deviations for the overall traffic in mobile networks: *(a) 5 service instance per request, (b) 10 service instance per request.*

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 Groba and Clarke [2014] Groba and Clarke [2011]. 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), 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

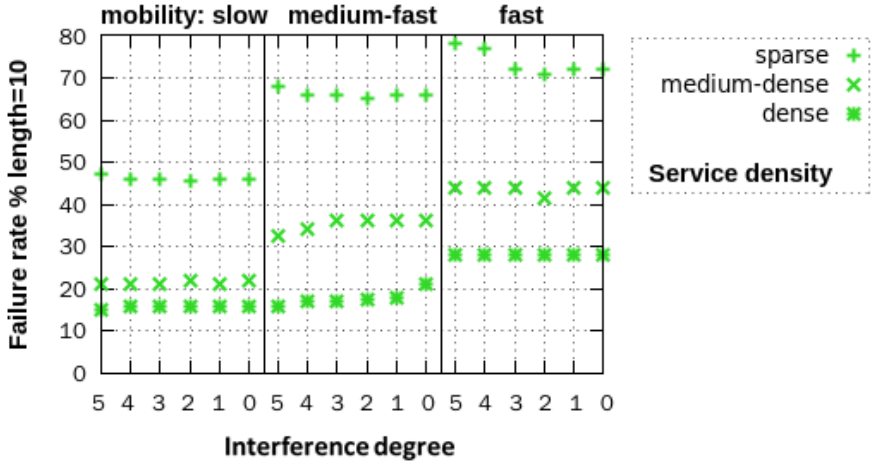


FIGURE 5.12: Interference degrees affect failure rates

traffic for a composition process was counted. The results (Figure 5.10 and Figure 5.11) show that GoCoMo processes service composition 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. Figure 5.11 shows that GoCoMo generates less traffic compared to CoopC, because CoopC merges all the partial plans during service discovery, which increases interactions among participants.

5.3.3.3 Impact of Heuristic Service Discovery

GoCoMo applied a heuristic service discovery mechanism. The selection of the interference degree may affect the results for returning a composition solution. Because a low interference degree can lead to more discovery messages that may increase the possibility of packet collisions and packet loss [Lipman et al., 2009], and in turn composition failures, while a high interference degree in some cases may result in a limited discovery scope, and are likely to increase discovery failures. A test used scenario 2.3 was run to measure this influence, which assumed all the nodes in the network use the same interference degree for simplicity. The test applied 6 levels of heuristic discovery from 0 to 5. Level 0 means there is no interference, and the level 5 process used the highest interference degree. Figure 5.12 shows that, in a medium-dense network (30 services) with medium-fast mobility, a high discovery interference (i.e., 5) will reduce composition failures. This is because, in such a dense network, even a small search scope can support

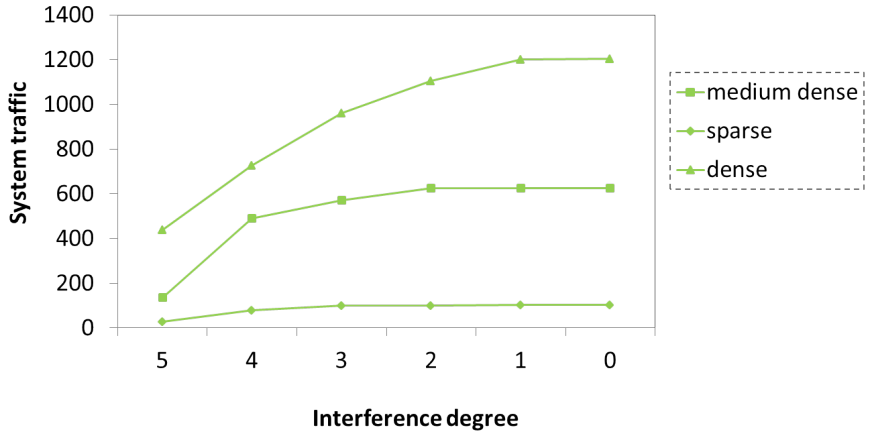


FIGURE 5.13: Interference degrees affect the system traffic

enough backup execution paths for failure recovery, while it sends less query messages than its high level counterparts, which reduces the potential for packet collision failures.

Figure 5.13 illustrates the system traffic in a medium-fast network with different interference degrees and service density. As shown in Figure 5.8 and 5.11, service providers' mobility has very limited impact on GoCoMo's system traffic, and so this simulation only presents GoCoMo's system traffic in a medium-fast network as a representative case. The result shows that a high discovery interference can reduce system traffic, especially in a dense service network. However, in a sparse service network, a high interference degree can increases composition failures (Figure 5.12), as it may imply a comparatively small search scope.

5.3.3.4 Planning Complex Service Flows

The evaluation scenario (scenario 2.4) 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 have multiple, differing in-degrees and out-degrees[8]. The potential service flows constructed by these atomic services for a composite service is illustrated in Figure 5.14. These service flows connect participating services relying on their data dependency, which include sequential workflow models (model a), parallel workflow models (model

[8]In-degree and out-degree indicate the number of connections entering and leaving a workflow node, respectively [Kiepuszewski et al., 2000].

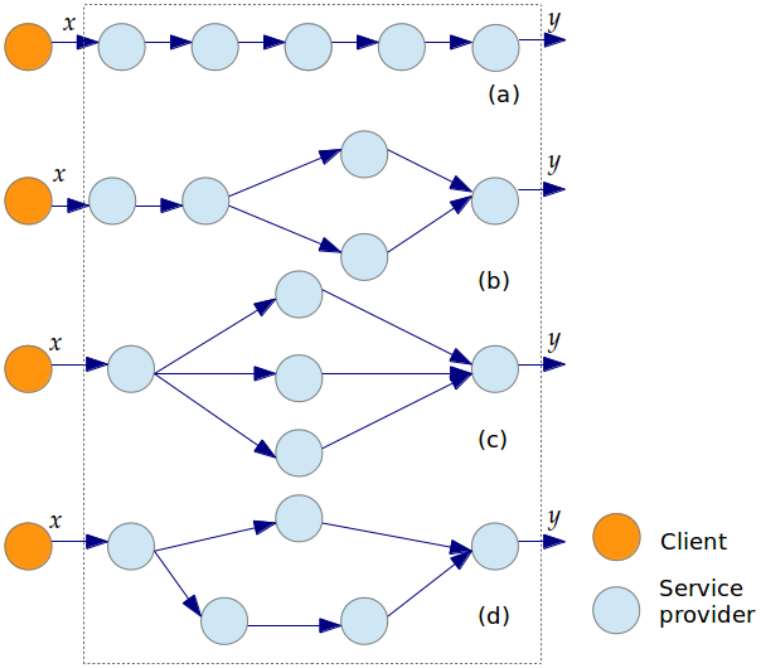


FIGURE 5.14: Potential service flow for a composite service that supports data transition: $X \rightarrow Y$

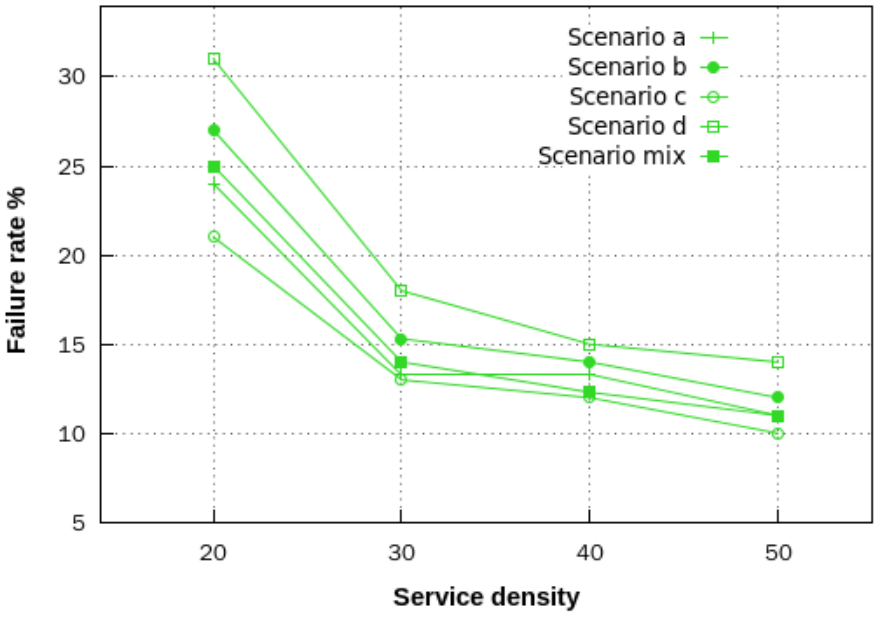


FIGURE 5.15: Failure rate for the data transition ($X \rightarrow Y$) request

c and d), and hybrid workflows (model b) [Kiepuszewski et al., 2000] [Liu and Kumar, 2005]. Each of them includes 5 service instances. All the service flows start and end with the client node that issues a service query, looking for a service composite that converts data X to Y. GoCoMo's failure rate on scenarios that contain the different kinds of flows are shown in Figure 5.15. It was simulated under medium-fast networks. On returning

solutions (Figure 5.14), GoCoMo supports scenario c the best, and has more failures in the scenario with type-d services. CoopC supports only type-a (sequential) services, and so only GoCoMo results are illustrated.

5.4 Evaluation Summary

This chapter outlined GoCoMo's evaluation, including a prototype case study and a simulation-based experiment. The evaluation used a set of criteria driven by this thesis's challenges. In the prototype case study, GoCoMo's feasibility in dynamic environments that have no conceptual composite or composition infrastructure was demonstrated. The simulation demonstrated GoCoMo's success in the target environment, indicated by planning and execution failure rate, response time and system traffic.

The case study in this chapter measured CPU consumption and memory usage, and took them as the main factors that affect a model's feasibility on mobile devices. It would also be interesting to measure service providers' energy consumption as the battery is a constrained resource to mobile devices. However, the case study used a Android Device Monitor running on a desktop, which requires a device to be plugged into the desktop's USB port. This means all the devices were charged during the experiment, and so battery consumption was not measured. On the other hand, the major energy consumer is wireless communications and screen. The GoCoMo App used in the case study keeps the screen active to display run-time information for evaluation, but it can be implemented as a background task. The power consumed when sending a message over a Bluetooth channel is about 0.145J/Kb [Lee et al., 2007]. The case study results have shown that the required message communication is 2 messages $*$ 227($byte$) on average, and so the GoCoMo composition process only requires a very small battery power.

In the simulation result, GoCoMo shows a reduction in communication over a selected related approach, and with an average 2 messages for each participant using controlled request flooding. In the worst case (i.e., uncontrolled request flooding in Figure 5.13), there are an average 10 messages per node in a sparse network, 20 messages in a medium dense network, and 32 messages in a dense network sent during a service composition process. The power consumed when sending a message over a WiFi network channel is

about 0.012J/Kb [Lee et al., 2007]. This thesis concludes that reasonably small battery power is required for the GoCoMo composition process.

Scenarios with a high service composition demand were not investigated. Such scenarios indicate multiple clients. GoCoMo uses on-demand binding and does not check a service's availability immediately before invoking the service. Service invocation is likely to fail when the required service is currently occupied by another composition process. As a service composition process locks a service provider's resource only when it is executing the required service, the length of a service's unavailability depends on the duration of the service's execution, which is normally predictable. Depending on a prediction of unavailable time, queuing up invocation messages when the time is short or otherwise re-selecting a service provider may solve the service invocation failure led by occupied service providers.

Chapter 6

Discussion and Conclusion

The research presented in this thesis has investigated a novel model for software service provision in mobile and pervasive computing environments. In particular, the research has focused on providing solutions to the challenges of composing services from multiple mobile devices, which behave in a decentralized fashion and adapt to network changes. A novel service composition model named GoCoMo that targets several service provision issues occurring in open and dynamic pervasive environments was proposed.

This chapter summaries the achievements of the research and its contributions to knowledge, discusses GoCoMo's trade-offs, and highlights potential areas for future work.

6.1 Overview of Thesis Achievements

Introduction Chapter 1 described the motivation of the thesis that arose from new challenges of service composition in mobile and pervasive computing environments. This thesis was especially concerned with the openness and dynamism features of the environments. The chapter argued that requests for complex functionality in a smart public space can be supported by the composition of services from a number of mobile devices located in the vicinity of the requester. However, such a dynamic, open computing environment can negatively impact the service composition process. Challenges include inadequate conceptual composites, limited system knowledge, unpredictable services availability, unreliable wireless communications, and dynamic composition links,

making the discovery scope limited or service execution unstable, and therefore, composition failures can occur. The chapter analyzed the most related solutions and claimed that the existing service composition model are not flexible enough to fully address these challenges. Based on the analysis on the state of the art solutions, a hypothesis was proposed, which assumed that a novel service composition model could reduce the composition failure probability, by dynamic composition planning that engages a big scope of potential service providers, through heuristic services discovery that reduces unnecessary system traffic, and using adaptable workflow-based composites that enable flexible execution and fast but low cost failure recovery.

State of the art Chapter 2 reviewed the state of art research in depth, and used a taxonomy that covered the most related aspects of a service composition process, containing when and how to locate a provider, what routes a request to a destined service provider or a composition manager, how to resolve a composition request, when a service will be bound for execution, how to execute services in decentralized ways, and when and how a failure can be recovered. As a result, the chapter claimed that the combination of dynamically controlled request flooding, goal-oriented planning, on-demand service binding, self-organized composition and dynamic failure recovery can provide a promising solution to the research presented in the thesis. However, existing approaches fail to bridge the gap between the solution and the target environment, as they are either inflexible about service discovery and planning or have limited adaptability.

Design Chapter 3 presented GoCoMo. To address the above issues, the thesis outlined a number of design requirements that GoCoMo must fulfill, and designed GoCoMo using a set of design concepts that entirely satisfy the requirements. More specifically, first, GoCoMo designed a planning-based composition announcement that allows service providers to update and pass a composition request to discover a service composite hop by hop, and only announces the finished composite to the requester. Second, GoCoMo dynamically controls request flooding according to the density of each service provider's neighbour services and the global service discovery and execution's time constraints, which trades off discovery scope with system traffic. Third, a flexible backward planning was designed to enable parallel and hybrid service flows when they are necessary. Fourth, GoCoMo selects service providers based on the estimated path reliability. Such path reliability values are calculated during service discovery and differ in every participant. Fifth, GoCoMo used dynamic service binding and only locks a service provider's resource

during the service's execution. Sixth, to adapt a composed service and engage the newly emerged service provider, GoCoMo allows service providers to proactively announce their available services. Composition participants receive such announcements and invite appropriate service providers to participate in a composition process. Last, service providers in GoCoMo only maintain execution paths linking to their subsequent services. The admission of service execution control is passed from one service provider to another.

Implementation The implementation of GoCoMo was presented in Chapter 4. GoCoMo was realised as an application layer middleware that connects services that are deployed in different devices by composing them according to their I/O dependency. The middleware's modules and the inner-module implementation were specified in the chapter. The chapter also included two prototypes of GoCoMo, GoCoMo App and GoCoMo-ns3, that implemented the modules in the GoCoMo middleware, and were used for evaluation.

Evaluation The evaluation of GoCoMo was described in Chapter 5, which showed GoCoMo's suitability and limitations in dynamic pervasive computing environments, and included two detailed evaluations, a prototype case study and a simulation. The prototype case study deployed GoCoMo App in a testbed composed by a set of real mobile devices, measured its performance on each individual devices, and demonstrated that GoCoMo can flexibly compose a number of diverse functionalities (services) deployed in diverse Android-based mobile devices to support a user's request. The simulation adopted 4 different scenarios and ran GoCoMo-ns3 in these scenarios. Results illustrated that GoCoMo is more flexible in terms of composition discovery/planning, and service execution, comparing to a service composition approach that also supports composition planning (baseline). In addition, GoCoMo produced less system traffic and spent less time on the service composition process. However, GoCoMo, similar to the baseline approach, does not solve the invocation collision problem that may occur when the demand for composition is high.

In summary, the thesis has presented a novel service composition model that does not rely on a conceptual composite, uses only local service knowledge, has low composition failure probabilities, and produces acceptable system traffic in environments where service providers are mobile, and service availability are unpredictable. The thesis's main contribution can be concluded as:

- A heuristic service discovery model that uses a composition's time-constraints defined by the composition requester and network properties, like node density, to prevent composition request flooding an infrastructure-less service provisioning network, while discovering sufficient potential service providers for the composition.

- A decentralized composition planning model for service composition that relies on an extension of goal-driven planning, supporting sequential, parallel and hybrid service flows.

- A composite adaptation and execution mechanism that is based on a novel notion for decentralized service execution: execution guideposts and directions, which offers more flexibility for composition participants when selecting a service provider to execution at runtime.

6.2 Discussion

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. This section discusses open issues that have not been targeted in the previous chapters but are essential in some cases.

6.2.1 Service Flows

GoCoMo is designed for mobile and pervasive computing applications that require multiple participants, and run in dynamic ad hoc environments. GoCoMo allows the generic processes of smart public spaces for pervasive computing, like multiple-source information querying, mobile-sensing, data aggregation, and in-network data processing to be planned and executed in a decentralized manner. It also raises the potential for more complex functionalities like the messaging-based concierge service [Shaffer and Keaveney, 2012] [Breau et al., 2013] to be applied in pervasive computing environments.

However, when executing a parallel service flow, GoCoMo selects a join-node before the parallel service flow gets invoked. This makes it costly to recover a parallel service flow comprising long branches when the selected join-node is missing, as re-selecting a join-node has to go back to the beginning of the branch (see results in Section 5.3.3.4). In addition, as service flows that include iterative logic have not been addressed, GoCoMo does not suit applications that imply iterative transactions, such as e-commerce and intelligent control.

6.2.2 Privacy and Security

The work in this thesis relies on services that are likely to be offered by third-party providers to resolve users' request and process their data. With regard to privacy and security concerns, a trustworthy service provider network is assumed in this research, or service selection should be based on services' trustworthiness levels, represented by data from off-device authorities like reputation ratings [Mallayya et al., 2015] or reviews. GoCoMo does go some way towards privacy and security issues because of its backward planning model. In general, hop-by-hop processing may expose the overall control and dataflow to assigned providers. GoCoMo's backward composition model partitions 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). In a network where individual service providers do not share any data except for those that are essential to finishing a composition process, the backward composition model can prevent an entire dataflow or service flow being exposed to any service provider.

6.2.3 Semantic Matchmaking

GoCoMo discovers services with a combination of multicast routing and constrained flooding. Semantic matchmaking is assumed for service matching, though no semantic matchmaking infrastructure is required for service discovery, e.g., pre-established semantic overlay [Pirrò et al., 2012]. This implies that dynamic matchmaking is used, which matches local services to a received composition request at runtime. GoCoMo searches for services by dynamically matching the composition goal provided by a client to the input/output (I/O) parameters of a service, but no specific matchmaker (e.g., iSeM,

OWLS-SLRlite, COV4SWS, etc.) or Semantic Overlay Networks (SON) is assumed in this work. The feasibility and performance of different semantic matchmakers has been studied by Klusch [2012], and are outside the scope of this thesis. A SON may introduce maintenance overhead to the system, however, semantic dependency overlays [Chen and Clarke, 2014] have the potential to be used to support the discovery model described in this thesis, and may achieve more efficient service discovery in a particular scenario where dynamic semantic matchmaking is time intensive. However, 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 required.

6.2.4 High Composition Demand

GoCoMo uses planning-based composition announcements, which means service providers decide whether or not to participate in a composition process to reason about a composite and provide services. GoCoMo's service selection is based on execution path's reliability. A popular service provider is likely to be subscribed frequently, and so may have to maintain many execution guideposts for different compositions at the same time, such a service has, as a result, a high probability of being unavailable. If it is possible, such kinds of services should be avoided as a primary service at service binding and invocation.

On the other hand, in a network with a high composition demand, selecting a popular service for execution is probable, as a lot of service providers may be over-subscribed. It is worth exploring how to optimise the schedule of service invocation requests for different composition processes, to minimise the length of a service's unavailability.

6.3 Future Work

The current version of GoCoMo suggests using existing semantic matchmaking models, but GoCoMo's prototypes only implemented a syntactic matchmaker. Current semantic matchmakers that have been employed by service composition approaches are generally based on centralized ontology bases [Hibner and Zielinski, 2007, Oh et al., 2008, Ren et al., 2011, Rodriguez-mier et al., 2012], rely on infrastructures which are

insufficiently flexible [Davidyuk and Georgantas, 2011, Fujii and Suda, 2005, Mokhtar et al., 2008, Pirrò et al., 2012] or allow for dynamic infrastructure-less matchmaking but resource-consuming [Klusch, 2012]. Further work can be done on supporting semantic matchmaking that are lightweight and flexible.

GoCoMo mainly considers service execution time as the QoS attribute and the length of the link as the factor that influences path reliability. More attributes like service reputation and bandwidth can be added to the context. Further research on modelling path reliability in the target environment with more QoS attributes will be required to support more exhaustive reliability estimation, resulting in possibly less composition disruption.

This thesis focuses on the application-level and addresses the combination of services instead of devices. Service composition can combine services from different devices or the same device. Existing research [Bianchini and Antonellis, 2008] distinguishes the link that connects services in the same device and that connects services in different devices as inter-peer links and intra-peer links, respectively. Inter-peer service connections and the management of them has not been studied in the thesis. Analysis of different inter-peer link estimation and management mechanisms [Prinz et al., 2008] will be needed to improve the flexibility of GoCoMo.

Appendix A

Further Implementation Detail: Prototypes

A.1 GoCoMo App

GoCoMo App is an Android application that supports service provision in bluetooth-based ad hoc networks. It includes a prototype implementation of the GoCoMo middleware and a visualization module, making service provisioning lightweight and flexible. GoCoMo App supports two groups of users: composition clients and service providers. The main features of GoCoMo App are illustrated below:

- Supports most of GoCoMo's functionalities and behaviours, including composition request generation and issue for composition clients, local composition planning, decentralised composition execution and adaptation, and service advertising/inviting.
- Compatible with Android devices with API level 16 and above.
- Visualises message transmission and the local composition process.

Figure A.1 illustrates the class diagram of GoCoMo App. The message builders were presented in Figure 4.8 on page 95 in Chapter 4. The orange line wraps the group of classes introduced in the GoCoMo middleware, and the rest of the classes and packages supports functions like wireless communication and process visualization.

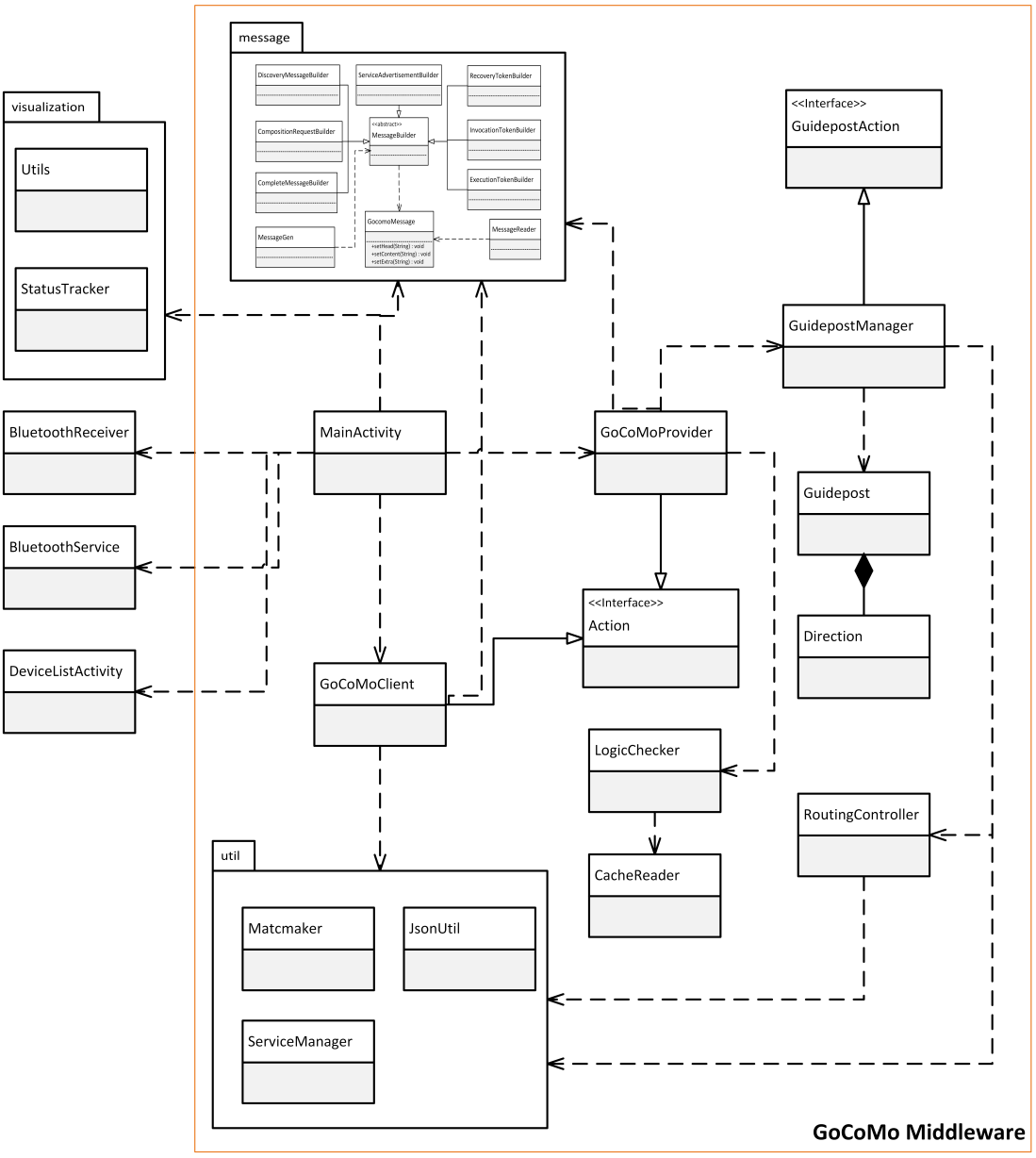


FIGURE A.1: GoCoMo App's class diagram

GoCoMo App did not implement GoCoMo's heuristic discovery model because BlueHoc, the ad hoc network used in GoCoMo App, does not support multi-hop transmission or dynamic routing. GoCoMo App can be extended to support the heuristic discovery model in Android 5.0 Lollipop (API 21) and above versions as these Android versions allow applications to directly control Bluetooth for broadcasting. GoCoMo App used JSON data to store the Guidepost object, and XML-based descriptions to specify GoCoMo messages. Figure A.2 and A.3 illustrate an example of a running service provider and a running composition client.

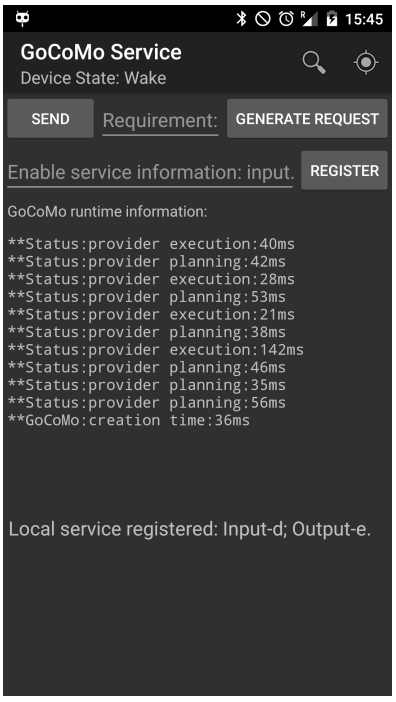


FIGURE A.2: GoCoMo App: an running device that acts as a service provider

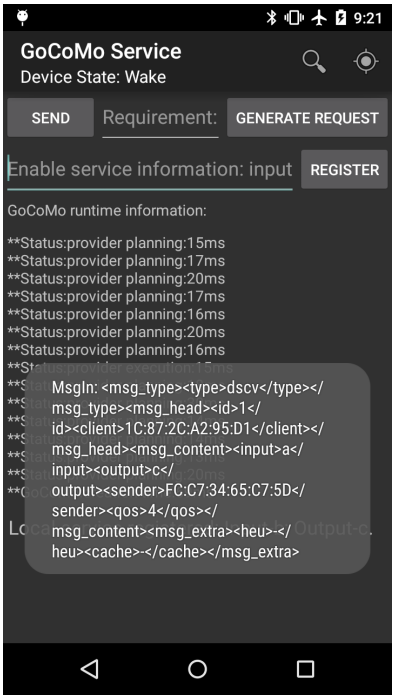


FIGURE A.3: GoCoMo App: An example of a GoCoMo message

A.2 GoCoMo-ns3

GoCoMo-ns3 works with ns-3 system modules, simulating the GoCoMo composition process in a WiFi ad hoc network. As illustrated in A.6, GoCoMo-ns3 is used by the *Experiment* class which initialises a number of wireless devices, configurates their mobility models, and establishes an ad hoc networks that controls message interaction between them. GoCoMo-ns3 implemented the *DataReader* class to load a global service topology, since the ns-3 platform does not manage each individual node's service information independently.

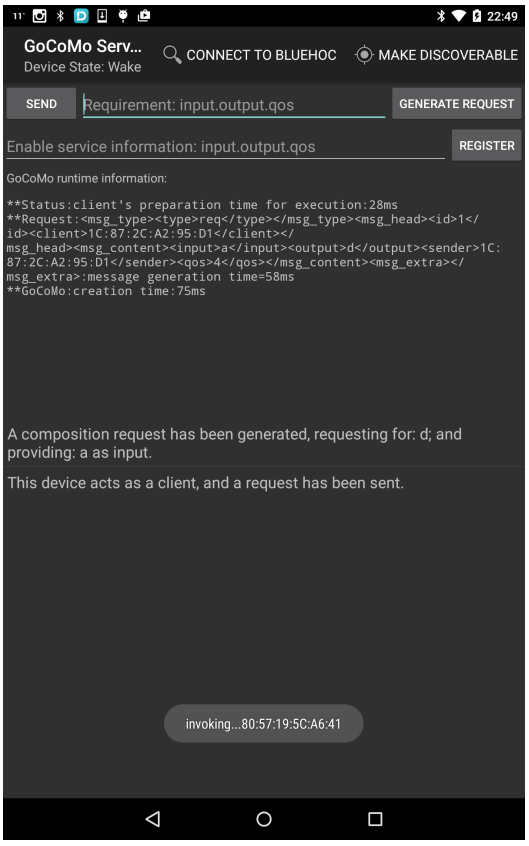


FIGURE A.4: GoCoMo App: an running device that acts as a composition client

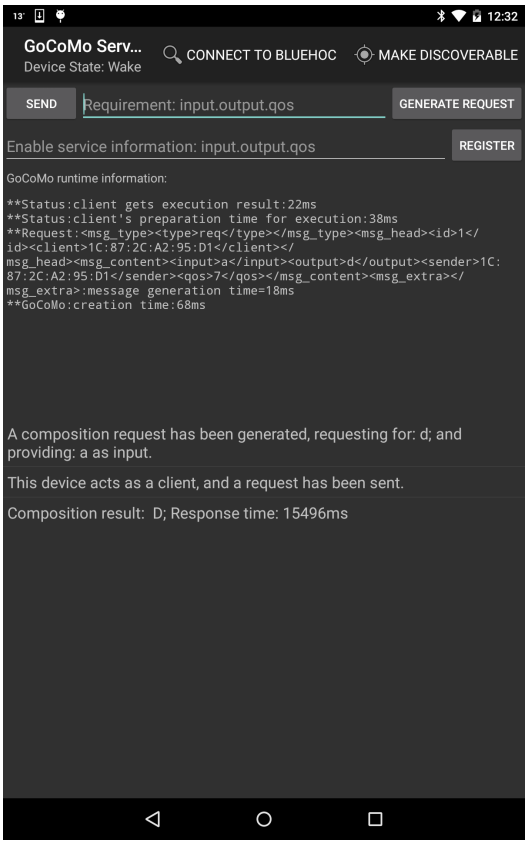


FIGURE A.5: GoCoMo App: a composition client gets composition result

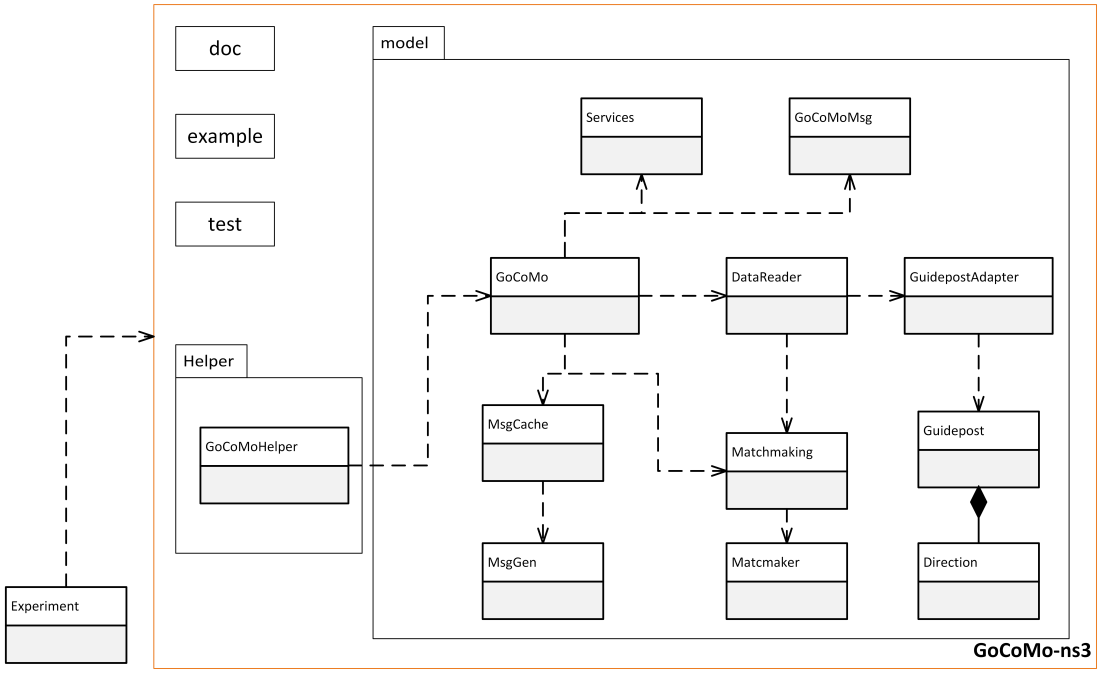


FIGURE A.6: GoCoMo-ns3's class diagram

Appendix B

Evaluation Results' Validity

B.1 Results' Validity Using 2-Sample Z-test

B.1.1 CoopC and GoCoMo's Service Discovery Delay

This section a. As 300 (> 30) samples were used for the simulation, this thesis uses 2-sample z-test. The testable statistical hypothesis for the simulation result are listed below: The null hypothesis (H_0): There is no difference between the population means of GoCoMo's discovery time and CoopC's discovery time on the test of discovery delay in a service composition process, that is $H_0 : \mu_1 - \mu_2 = 0$.

The alternative hypotheses (H_1 and H_2): There is a difference between the population means of GoCoMo's discovery time and CoopC's discovery time on the test of discovery delay in a service composition process, that is $H_1 : \mu_1 - \mu_2 < 0$ and $H_2 : \mu_1 - \mu_2 > 0$.

The results of the simulation found z values under the null hypothesis and the alternative hypothesis shows in Table B.1.

B.1.2 CoopC and GoCoMo's Service Discovery Traffic

The testable statistical hypothesis for the simulation result are listed below: The null hypothesis (H_0): There is no difference between the population means of GoCoMo's discovery traffic and CoopC's discovery traffic on the test of discovery traffic in a service composition process, that is $H_0 : \mu_1 - \mu_2 = 0$.

TABLE B.1: The difference between GoCoMo's discovery time and CoopC's discovery time

slow, 5 services	**20**	**30**	**40**	**50**
z value	-12.56106113	-16.79026592	-18.02333183	-10.69922846
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
mid-fast, 5 services	**20**	**30**	**40**	**50**
z value	-19.04224056	-14.76793744	-19.41144055	-12.27153227
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
fast, 5 services	**20**	**30**	**40**	**50**
z value	-11.54649281	-14.73933101	-17.47812923	-17.79875104
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
slow, 10 services	**20**	**30**	**40**	**50**
z value	-32.29453272	-37.9788997	-26.56567767	-24.23023048
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
mid-fast, 10 services	**20**	**30**	**40**	**50**
z value	-19.30967952	-41.52542078	-31.77308579	-25.29579807
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
fast, 10 services	**20**	**30**	**40**	**50**
z value	-19.64695977	-28.91184312	-33.77897592	-24.06613906
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject

The alternative hypotheses (H_1 and H_2): There is a difference between the population means of GoCoMo's discovery traffic and CoopC's discovery traffic on the test of discovery traffic in a service composition process, that is $H_1 : \mu_1 - \mu_2 < 0$ and $H_2 : \mu_1 - \mu_2 > 0$.

The results of the simulation found z values under the null hypothesis and the alternative hypothesis shows in Table B.2.

TABLE B.2: The difference between GoCoMo's discovery traffic and CoopC's discovery traffic

slow, 5 services	**20**	**30**	**40**	**50**
z value	-6.56873	8.410306	-9.09909	-2.9423
H_0	reject	reject	reject	reject
H_1	accept	reject	accept	accept
H_2	reject	accept	reject	reject
mid-fast, 5 services	**20**	**30**	**40**	**50**
z value	-6.76328	11.75466	-8.38718	-10.0033
H_0	reject	reject	reject	reject
H_1	accept	reject	accept	accept
H_2	reject	accept	reject	reject
fast, 5 services	**20**	**30**	**40**	**50**
z value	-5.6026	11.03799	-7.44844	-10.00331775
H_0	reject	reject	reject	reject
H_1	accept	reject	accept	accept
H_2	reject	accept	reject	reject
slow, 10 services	**20**	**30**	**40**	**50**
z value	-15.4047	-52.666	-53.7117	-39.9217
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
mid-fast, 10 services	**20**	**30**	**40**	**50**
z value	-13.5711	-59.179	-45.8739	-46.6946
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
fast, 10 services	**20**	**30**	**40**	**50**
z value	-11.2968	-42.1154	-46.6897	-35.1998
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject

B.1.3 CoopC and GoCoMo's Response Time

The testable statistical hypothesis for the simulation result are listed below: The null hypothesis (H_0): There is no difference between the population means of GoCoMo's response time and CoopC's response time on the test of response delay in a service composition process, that is $H_0 : \mu_1 - \mu_2 = 0$.

The alternative hypotheses (H_1 and H_2): There is a difference between the population means of GoCoMo's response time and CoopC's response time on the test of response delay in a service composition process, that is $H_1 : \mu_1 - \mu_2 < 0$ and $H_2 : \mu_1 - \mu_2 > 0$.

TABLE B.3: The difference between GoCoMo's response time and CoopC's response time

slow, 5 services	**20**	**30**	**40**	**50**
z value	-9.992451711	-8.489276933	-8.994816681	-6.704062264
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
mid-fast, 5 services	**20**	**30**	**40**	**50**
z value	-9.467088577	-8.814763295	-9.297349149	-6.695604027
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
fast, 5 services	**20**	**30**	**40**	**50**
z value	-7.663452997	-8.307561975	-7.978716457	-5.495297028
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
slow, 10 services	**20**	**30**	**40**	**50**
z value	-22.97000542	-17.12140163	-9.065221592	-3.549828118
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
mid-fast, 10 services	**20**	**30**	**40**	**50**
z value	-15.65747282	-16.64296405	-8.09600387	-3.557499945
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
fast, 10 services	**20**	**30**	**40**	**50**
z value	-13.60161003	-13.03524901	-9.110352989	-3.6688557
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject

The results of the simulation found z values under the null hypothesis and the alternative hypothesis shows in Table B.3.

B.1.4 CoopC and GoCoMo's Composition Traffic

The testable statistical hypothesis for the simulation result are listed below: The null hypothesis (H_0): There is no difference between the population means of GoCoMo's composition traffic and CoopC's composition traffic on the test of composition traffic in a service composition process, that is $H_0 : \mu_1 - \mu_2 = 0$.

TABLE B.4: The difference between GoCoMo's composition traffic and CoopC's composition traffic

slow, 5 services	**20**	**30**	**40**	**50**
z value	-1.633975083	-0.379535408	-17.01789151	-6.446042073
H_0	accept	accept	reject	reject
H_1	reject	reject	accept	accept
H_2	reject	reject	reject	reject
mid-fast, 5 services	**20**	**30**	**40**	**50**
z value	-16.25638314	1.10071635	-8.67860778	-25.19971832
H_0	reject	accept	reject	reject
H_1	accept	reject	accept	accept
H_2	reject	reject	reject	reject
fast, 5 services	**20**	**30**	**40**	**50**
z value	-12.76707972	-0.500782492	-7.814265259	-20.12883122
H_0	reject	accept	reject	reject
H_1	accept	reject	accept	accept
H_2	reject	reject	reject	reject
slow, 10 services	**20**	**30**	**40**	**50**
z value	-22.75254441	-88.24796791	-56.87641912	-59.74280959
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
mid-fast, 10 services	**20**	**30**	**40**	**50**
z value	-10.41214504	-81.90194881	-34.45794557	-55.12380986
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject
fast, 10 services	**20**	**30**	**40**	**50**
z value	-4.51012531	-64.84092515	-70.28014077	-21.9503516
H_0	reject	reject	reject	reject
H_1	accept	accept	accept	accept
H_2	reject	reject	reject	reject

The alternative hypotheses (H_1 and H_2): There is a difference between the population means of GoCoMo's composition traffic and CoopC's composition traffic on the test of composition traffic in a service composition process, that is $H_1 : \mu_1 - \mu_2 < 0$ and $H_2 : \mu_1 - \mu_2 > 0$.

The results of the simulation found z values under the null hypothesis and the alternative hypothesis shows in Table B.4.

Appendix C

Glossary

- **cache**

  A cached request sender is represented as a $C_i \in cache$ ($C_i = \langle S_{id}, G^{matched}, \rho \rangle$), 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 ρ ($\rho \in (0, 1)$) captures the progress of addressing the partially matched goal.

- **composition request**

  A composition request is represented by $R = \langle R_{id}, \mathcal{I}, \mathcal{O}, \mathcal{F}, \mathcal{C} \rangle$, 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) $\mathcal{O} = \{\langle \mathcal{O}^S, \mathcal{O}^D \rangle\}$.

- **discovery message**

  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 Definition 6 on page 69), and h is a criterion value for request forwarding and service allocation (see Section 3.4.4 and 3.4.3).

- **directions**

  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.

- **execution guidepost**

  An execution guidepost $G = \langle R_{id}, \mathcal{D} \rangle$ maintained by composite participant P includes a set of execution directions $\mathcal{D}$ and the id of its corresponding composite request. For each execution direction $d_j \in \mathcal{D}$, d_j is defined as $\langle d_j^{id}, \mathcal{S}^{post}, \omega, \mathcal{Q} \rangle$, where d_j^{id} is a unique id for d_j, and the set $\mathcal{S}^{post}$ stores P's post-condition services that can be chosen for next-hop execution. The set ω 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 robustness of this direction, e.g., the estimated execution path strength and the execution time (Section 3.4.4).

- **service**

  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 QoS criterion as delay in composition and execution can cause failures [Groba and Clarke, 2014]. A service composition model should select services with short execution time to reduce delay in execution.

- **service announcement message**

  A service announcement message is described as $SA = \langle P_{address},\ OUT_p \rangle$, where $P_{address}$ represents the unique address of the service provider, and the OUT_p is the output data can be provided by the service provider.

- **service provider**

  A service provider is a service deployment device that has a wrapped functionality exposed through a service interface and could therefore be used remotely as a service.

Bibliography

[Al-Oqily and Karmouch(2011)] Ibrahim Al-Oqily and Ahmed Karmouch. A Decentralized Self-Organizing Service Composition for Autonomic Entities. *ACM Transactions on Autonomous and Adaptive Systems*, 6(1):1–18, February 2011.

[Alomari and Sumari(2008)]Saleh Ali Alomari and Putra Sumari. Multimedia Applications for MANETs over Homogeneous and Heterogeneous Mobile Devices. In *Wireless Communications and Networks - Recent Advances*. 2008.

[AppBrain(2015)] AppBrain. Number of Android applications, 2015. URL `http://www.appbrain.com/stats/number-of-android-apps`.

[Ardagna and Pernici(2007)] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6):369–384, June 2007. ISSN 0098-5589. doi: 10.1109/TSE.2007.1011.

[Artail et al.(2008)] Hassan Artail, Khaleel Wafiq Mershad, and Hicham Hamze. DSDM: A distributed service discovery model for manets. *IEEE Transactions on Parallel and Distributed Systems*, 19(9):1224–1236, 2008.

[Aschoff and Zisman(2011)] Rafael R. Aschoff and Andrea Zisman. QoS-Driven proactive adaptation of service composition. *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 421–435, June 2011. doi: 10.1109/SEAMS.2012.6224385.

[Atluri et al.(2007)] Vijayalakshmi Atluri, Soon Ae Chun, Ravi Mukkamala, and Pietro Mazzoleni. A decentralized execution model for inter-organizational workflows. *Distributed and Parallel Databases*, 22(1):55–83, May 2007. ISSN 0926-8782. doi: 10.1007/s10619-007-7012-1.

[Bianchini and Antonellis(2008)] Devis Bianchini and Valeria De Antonellis. On-the-fly collaboration in distributed systems through service semantic overlay. *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 0–3, 2008.

[Bianchini et al.(2010)] Devis Bianchini, Valeria De Antonellis, and Melchiori Michele. P2P-SDSD: on-the-fly service- based collaboration in distributed systems. *IJMSO 5 (3)*, pages 0–3, 2010.

[Breau et al.(2013)] Jeremy R. Breau, Eric E. Miller, Se Y. Ng, and Carl J. Persson. Concierge for portable electronic device. *US Patent 8,489,080*, 1(12), 2013.

[Breslau et al.(2000)] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *Computer*, 33(5):59–67, May 2000. ISSN 0018-9162. doi: 10.1109/2.841785.

[Brønsted et al.(2010)] Jeppe Brønsted, Klaus Marius Hansen, and Mads Ingstrup. Service composition issues in pervasive computing. *IEEE Pervasive Computing*, 9(1): 62–70, January 2010. ISSN 1536-1268. doi: 10.1109/MPRV.2010.11.

[Bucchiarone et al.(2010)] Antonio Bucchiarone, Raman Kazhamiakin, Cinzia Cappiello, Elisabetta di Nitto, and Valentina Mazza. A context-driven adaptation process for service-based applications. In *Proceedings of the 2Nd International Workshop on Principles of Engineering Service-Oriented Systems*, PESOS '10, pages 50–56, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-963-3. doi: 10.1145/1808885.1808896.

[Burnette(2009)] Ed Burnette. *Hello, Android: Introducing Google's Mobile Development Platform.* Pragmatic Bookshelf, 2nd edition, 2009. ISBN 1934356492, 9781934356494.

[Butler(2011)] Margaret Butler. Android: Changing the mobile landscape. *IEEE Pervasive Computing*, 10(1):4–7, 2011. ISSN 15361268. doi: 10.1109/MPRV.2011.1.

[Chakraborty and Joshi(2002)] Dipanjan Chakraborty and Anupam Joshi. GSD: A novel group-based service discovery protocol for MANETS. *Mobile and Wireless Communications Network, International Workshop on*, pages 140–144, 2002.

[Chakraborty and Joshi(2006)] Dipanjan Chakraborty and Anupam Joshi. Toward distributed service discovery in pervasive computing environments. *Mobile Computing, IEEE Transaction on*, 5(2):97–112, 2006.

[Chakraborty et al.(2005)]Dipanjan Chakraborty, Anupam Joshi, Tim Finin, and Yelena Yesha. Service composition for mobile environments. *Mob. Netw. Appl.*, 10(4):435–451, August 2005. ISSN 1383-469X.

[Chen and Clarke(2014)] Nanxi Chen and Siobhán Clarke. A Dynamic Service Composition Model for Adaptive Systems in Mobile Computing Environments. *IEEE International Conference on Service-Oriented Computing -ICSOC*, 2014.

[Cheng et al.(2011)] Ding-Yuan Cheng, Kuo-Ming Chao, Chi-Chun Lo, and Chen-Fang Tsai. A user centric service-oriented modeling approach. *World Wide Web*, 14(4): 431–459, May 2011.

[Cho and Lee(2005)] Chunglae Cho and Duckki Lee. Survey of service discovery architectures for mobile ad hoc networks. *term paper, Mobile Computing, CEN*, 2005.

[Christopher N. Ververidis and George C. Polyzos(2008)] Christopher N. Ververidis and George C. Polyzos. Service Discovery for Mobile Ad hoc Networks: A Survey of Issues and Techniques. *IEEE Communications Surveys Tutorials*, 10(3):30–45, 2008. doi: 10.1109/COMST.2008.4625803.

[Conti and Giordano(2014)] Marco Conti and Silvia Giordano. Mobile ad hoc networking: Milestones, challenges, and new research directions. *IEEE Communications Magazine*, 52(1):85–96, 2014. ISSN 01636804. doi: 10.1109/MCOM.2014.6710069.

[Crespo and Garcia-Molina(2005)] Arturo Crespo and H Garcia-Molina. Semantic overlay networks for p2p systems. *Agents and Peer-to-Peer Computing*, 2005.

[Dai and Wu(2004)] Fei Dai and Jie Wu. Performance analysis of broadcast protocols in ad hoc networks based on self-pruning. *Parallel and Distributed Systems, IEEE Transactions on*, 15(11):1–13, 2004.

[Dai et al.(2015)] Huijun Dai, Hua Qu, and Jihong Zhao. A Knowledge-Based Service Composition Algorithm with Better QoS in Semantic Overlay. *Mathematical Problems in Engineering*, 2015.

[Davidyuk and Georgantas(2011)] Oleg Davidyuk and Nikolaos 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*, (2011):1–22, 2011.

[de Medeiros et al.(2015)] Robson W. A. de Medeiros, Nelson S. Rosa, and Luís Ferreira Pires. Predicting service composition costs with complex cost behavior. In *Services Computing (SCC), 2015 IEEE International Conference on*, pages 419–426, June 2015. doi: 10.1109/SCC.2015.64.

[del Val et al.(2014)] E. del Val, M. Rebollo, and V. Botti. Combination of self-organization mechanisms to enhance service discovery in open systems. *Information Sciences*, 279:138 – 162, 2014. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2014.03.109. URL `http://www.sciencedirect.com/science/article/pii/S002002551400406X`.

[Efstathiou et al.(2014a)] Dionysios Efstathiou, P McBurney, Steffen Zschaler, and Johann Bourcier. Efficiently Approximating the QoS of Composite Services in Mobile Ad-Hoc Networks. *Technical Report*, 2014a.

[Efstathiou et al.(2014b)] Dionysios Efstathiou, Peter McBurney, Steffen Zschaler, and Johann Bourcier. Efficient Multi-Objective Optimisation of Service Compositions in Mobile Ad hoc Networks Using Lightweight Surrogate Models. *Journal of Universal Computer Science*, 2014b.

[Fdhila et al.(2009)] Walid Fdhila, Ustun Yildiz, Claude Godart, Loria Inria, and Nancy Grand. A flexible approach for automatic process decentralization using dependency tables University of California. In *IEEE 7th International Conference on Web Services*, volume 2009, 2009.

[Flyvbjerg(2006)] Bent Flyvbjerg. Five misunderstandings about case-study research. *Qualitative inquiry*, pages 219–245, 2006.

[Fok et al.(2010)] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Servilla: A flexible service provisioning middleware for heterogeneous sensor networks. *Science of Computer Programming*, 77(6):663–684, June 2010. ISSN 01676423. doi: 10.1016/j.scico.2010.11.006.

[Friedman et al.(2007)] Roy Friedman, Daniela Gavidia, Luis Rodrigues, Aline Carneiro Viana, and Spyros Voulgaris. Gossiping on manets: The beauty and the beast. *SIGOPS Oper. Syst. Rev.*, 41(5):67–74, October 2007. ISSN 0163-5980. doi: 10.1145/1317379.1317390.

[Fujii and Suda(2005)] Keita Fujii and Tatsuya Suda. Semantics-based dynamic service composition. *IEEE Journal on Selected Areas in Communications*, (December):2361–2372, 2005.

[Furno and Zimeo(2013)] Angelo Furno and Eugenio Zimeo. Efficient Cooperative Discovery of Service Compositions in Unstructured P2P Networks. *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 58–67, February 2013. doi: 10.1109/PDP.2013.10.

[Furno and Zimeo(2014)] Angelo Furno and Eugenio Zimeo. Self-scaling cooperative discovery of service compositions in unstructured P2P networks. *Journal of Parallel and Distributed Computing*, 74(10):2994–3025, October 2014.

[Geyik et al.(2013)] Sahin Cem Geyik, Boleslaw K. Szymanski, and Petros Zerfos. Robust dynamic service composition in sensor networks. *IEEE Transactions on Services Computing*, 6(4):560–572, October 2013. doi: 10.1017/CBO9781107415324.004.

[Gharzouli and Boufaida(2011)] Mohamed Gharzouli and Mahmoud Boufaida. PM4SWS: A P2P Model for Semantic Web Services Discovery and Composition. *Journal of Advances in Information Technology*, 2(1):15–26, February 2011.

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

[Groba and Clarke(2012)] Christin Groba and Siobhán Clarke. Synchronising Service Compositions in Dynamic Ad Hoc Environments. *2012 IEEE First International Conference on Mobile Services*, pages 56–63, June 2012.

[Groba and Clarke(2014)] Christin Groba and Siobhán Clarke. Opportunistic service composition in dynamic ad hoc environments. *IEEE Transactions on Services Computing*, X(c):1–1, 2014.

[Gronli et al.(2014)] Tor-Morten Gronli, Jarle Hansen, Gheorghita Ghinea, and Muhammad Younas. Mobile Application Platform Heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS. *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, pages 635–641, 2014. ISSN 1550-445X. doi: 10.1109/AINA.2014.78.

[Gu and Nahrstedt(2006)] Xiaohui Gu and Klara Nahrstedt. Distributed multimedia service composition with statistical QoS assurances. *Multimedia, IEEE Transactions on*, 8(1):141–151, 2006.

[He et al.(2008)] Qiang He, Jun Yan, Hai Jin, and Yun Yang. Adaptation of web service composition based on workflow patterns. *Service-Oriented Computing–ICSOC 2008*, pages 22–37, 2008.

[He et al.(2013)] Qiang He, Jun Yan, Yun Yang, R. Kowalczyk, and Hai Jin. A decentralized service discovery approach on peer-to-peer networks. *Services Computing, IEEE Transactions on*, 6(1):64–75, First 2013. ISSN 1939-1374. doi: 10.1109/TSC.2011.31.

[Hibner and Zielinski(2007)] Artur Hibner and Krzysztof Zielinski. Semantic-based Dynamic Service Composition and Adaptation. In *Services, 2007 IEEE Congress on*, pages 213–220, 2007.

[Hinojos et al.(2014)] G Hinojos, C Tade, S Park, D Shires, and D Bruno. BlueHoc : Bluetooth Ad-Hoc Network Android Distributed Computing. *In Proceedings of The International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 1–6, 2014.

[Hiyama et al.(2012)] Masahiro Hiyama, Elis Kulla, Makoto Ikeda, Leonard Barolli, and Muhammad Younas. A Comparative Study of a MANET Testbed Performance in Indoor and Outdoor Stairs Environment. *2012 15th International Conference on Network-Based Information Systems*, pages 134–140, 2012. doi: 10.1109/NBiS.2012.32.

[Hogie et al.(2006)] Luc Hogie, Pascal Bouvry, and Frédéric Guinand. An Overview of MANETs Simulation. *Electronic Notes in Theoretical Computer Science*, 150(1): 81–101, 2006. ISSN 1571-0661. doi: 10.1016/j.entcs.2005.12.025.

[Hosseini Seno et al.(2007)] Seyed Amin Hosseini Seno, Rahmat Budiarto, and Tat-Chee Wan. Survey and new Approach in Service Discovery and Advertisement for Mobile

Ad hoc Networks. *International Journal of Computer Science and Network Security*, 7(2):275–284, 2007.

[Ibrahim and Mouel(2009)] Noha Ibrahim and FL Mouel. A survey on service composition middleware in pervasive environments. *International Journal of Computer Science Issues*, 1:1–12, 2009.

[Jiang et al.(2007)] Shanshan Jiang, Yuan Xue, and D.C. Schmidt. Minimum disruption service composition and recovery over mobile ad hoc networks. pages 1–8, Aug 2007. doi: 10.1109/MOBIQ.2007.4451003.

[Jiang et al.(2009)] Shanshan Jiang, Yuan Xue, and Douglas C. Schmidt. Minimum disruption service composition and recovery in mobile ad hoc networks. *Computer Networks*, 53(10):1649–1665, July 2009.

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

[Kalasapur et al.(2007)] S Kalasapur, M Kumar, and B A Shirazi. Dynamic Service Composition in Pervasive Computing. *Parallel and Distributed Systems, IEEE Transactions on*, 18(7):907–918, 2007.

[Kang et al.(2008)] Eunyoung Kang, M Kim, Eunju Lee, and Ungmo Kim. DHT-based mobile service discovery protocol for mobile ad hoc networks. *Advanced Intelligent Computing Theories and Applications. With Aspects of Theoretical and Methodological Issues Lecture Notes in Computer Science*, 5226:610–619, 2008.

[Karaoglu and Heinzelman(2010)] Bora Karaoglu and Wendi Heinzelman. Multicasting vs. broadcasting: What are the trade-offs? *Proceedings of the Global Communications Conference-GLOBECOM*, 2010.

[Khakhkhar et al.(2012)] Sandip Khakhkhar, Vikas Kumar, and Sanjay Chaudhary. Dynamic Service Composition. *International Journal of Computer Science and Artificial Intelligence*, 2:32–42, September 2012.

[Khan et al.(2013)] Wazir Zada Khan, Yang Xiang, Mohammed Y Aalsalem, and Quratulain Arshad. Mobile Phone Sensing Systems: A Survey. *IEEE Communications Surveys & Tutorials*, 15:402–427, 2013. ISSN 1553-877X. doi: 10.1109/SURV.2012.031412.00077.

[Kiepuszewski et al.(2000)] Bartek Kiepuszewski, Arthur Harry, and Christoph J Bussler. On Structured Workflow Modelling Structured Workflows. *Advanced Information Systems Engineering Lecture Notes in Computer Science*, 1789, 2000:431–445, 2000.

[Kiess and Mauve(2007)] Wolfgang Kiess and Martin Mauve. A survey on real-world implementations of mobile ad-hoc networks. *Ad Hoc Networks*, 5(3):324–339, 2007. ISSN 15708705. doi: 10.1016/j.adhoc.2005.12.003.

[Kim et al.(2006)] M.J. Kim, M. Kumar, and B.a. Shirazi. Service discovery using volunteer nodes in heterogeneous pervasive computing environments. *Pervasive and Mobile Computing*, 2(3):313–343, September 2006. ISSN 15741192. doi: 10.1016/j.pmcj.2006.04.002.

[Kitchenham et al.(1995)] Barbara Kitchenham, Lesley Pickard, and Shari Lawrence Pfleeger. Case studies for method and tool evaluation. *IEEE Softw.*, 12(4):52–62, July 1995. ISSN 0740-7459. doi: 10.1109/52.391832.

[Klusch(2012)] Matthias Klusch. Overview of the S3 contest Performance evaluation of semantic service matchmakers. *Semantic Web Services*, pages 1–18, 2012.

[Kotz et al.(2004)] David Kotz, Calvin Newport, Robert S. Gray, Jason Liu, Yougu Yuan, and Chip Elliott. Experimental evaluation of wireless simulation assumptions. In *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '04, pages 78–82, New York, NY, USA, 2004. ACM. ISBN 1-58113-953-5. doi: 10.1145/1023663.1023679.

[Kozat and Tassiulas(2004)] Ulaş C. Kozat and Leandros Tassiulas. Service discovery in mobile ad hoc networks: an overall perspective on architectural choices and network layer support issues. *Ad Hoc Networks*, 2(1):23–44, January 2004. ISSN 15708705. doi: 10.1016/S1570-8705(03)00044-1.

[Kumar and Sam(2015)] C Niranjan Kumar and R Praveen Sam. MANET Test Bed for Rescue Operations in Disaster Management. *International Journal of Computer Science and Mobile Computing(IJCSMC)*, (7):432–437, 2015.

[Lane et al.(2010) Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, Andrew T. Campbell, and CollegeDartmouth. Adhoc And Sensor Networks: A Survey of Mobile Phone Sensing, 2010.

[Lee et al.(2007)] Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen. A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In *IEEE Conf. Industrial Electronics Society*, 2007.

[Li et al.(2015)] Bo Li, Yijian Pei, Hao Wu, and Bin Shen. Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds. *The Journal of Supercomputing*, 71(8):3009–3036, 2015. ISSN 0920-8542. doi: 10.1007/s11227-015-1425-9.

[Li et al.(2007)] Yang Li, JinPeng Huai, Ting Deng, HaiLong Sun, HuiPeng Guo, and Zongxia Du. QoS-aware Service Composition in Service Overlay Networks. *IEEE International Conference on Web Services (ICWS 2007)*, (Icws):703–710, July 2007. doi: 10.1109/ICWS.2007.148.

[Lipman et al.(2009)] Justin Lipman, Hai Liu, and Ivan Stojmenovic. Broadcast in Ad Hoc Networks. In Sudip Misra, Isaac Woungang, and Subhas Chandra Misra, editors, *Guide to Wireless Ad Hoc Networks*, Computer Communications and Networks, pages 121–150. Springer London, London, 2009.

[Liu et al.(2015a)] Chenyang Liu, Jian Cao, and Jie Wang. A parallel approach for service composition with complex structures in pervasive environments. In *Web Services (ICWS), 2015 IEEE International Conference on*, pages 551–558, June 2015a. doi: 10.1109/ICWS.2015.79.

[Liu and Kumar(2005)] Rong Liu and Akhil Kumar. An analysis and taxonomy of unstructured workflows. *Business Process Management*, pages 268–284, 2005.

[Liu et al.(2015b)] Xuanzhe Liu, Yun Ma, Gang Huang, Junfeng Zhao, Hong Mei, and Yunxin Liu. Data-driven composition for service-oriented situational web applications. *Services Computing, IEEE Transactions on*, 8(1):2–16, Jan 2015b. ISSN 1939-1374. doi: 10.1109/TSC.2014.2304729.

[Liu et al.(2015c)] Yang Liu, Yanyan Han, Zhipeng Yang, and Hongyi Wu. Efficient Data Query in Intermittently-Connected Mobile Ad Hoc Social Networks. *IEEE Transactions on Parallel and Distributed Systems*, 9219(APRIL):1–1, 2015c. ISSN 1045-9219. doi: 10.1109/TPDS.2014.2320922.

[Mallayya et al.(2015)] Deivamani Mallayya, Baskaran Ramachandran, and Suganya Viswanathan. An Automatic Web Service Composition Framework Using QoS-Based Web Service Ranking Algorithm. *The Scientific World Journal*, 2015.

[Mateescu et al.(2008)] Radu Mateescu, Pascal Poizat, and G Salaün. Adaptation of service protocols using process algebra and on-the-fly reduction techniques. *Service-Oriented Computing–ICSOC*, pages 84–99, 2008.

[Meshkova et al.(2008)] Elena Meshkova, Janne Riihijärvi, Marina Petrova, and Petri Mähönen. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer Networks*, 52(11):2097–2128, 2008. ISSN 13891286. doi: 10.1016/j.comnet.2008.03.006.

[Mian et al.(2009)] Adnan Noor Mian, Roberto Baldoni, and Roberto Beraldi. A survey of service discovery protocols in multihop mobile Ad Hoc networks. *IEEE Pervasive Computing*, 8(1):66–74, 2009. ISSN 15361268.

[Miller(1968)] RB Miller. Response time in man-computer conversational transactions. *Proceedings of the December 9-11, 1968, AFIPS Fall Joint Computer Conference*, 33: 267–277, 1968.

[Miraoui et al.(2011)] Moeiz Miraoui, Chakib Tadj, Jaouhar Fattahi, and Chokri Ben Amar. Dynamic Context-Aware and Limited Resources-Aware Service Adaptation for Pervasive Computing. *Advances in Software Engineering*, 2011:1–11, 2011. ISSN 1687-8655. doi: 10.1155/2011/649563.

[Mokhtar and Liu(2005)] SB Mokhtar and Jinshan Liu. QoS-aware dynamic service composition in ambient intelligence environments. *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, 2005(2005):317–320, 2005.

[Mokhtar et al.(2008)] Sonia Ben Mokhtar, Davy Preuveneers, Nikolaos Georgantas, Valérie Issarny, and Yolande Berbers. EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. *Journal of Systems and Software*, 81(5):785–808, May 2008. ISSN 01641212. doi: 10.1016/j.jss.2007.07.030.

[Mostarda et al.(2010)] Leonardo Mostarda, Srdjan Marinovic, and Naranker Dulay. Distributed Orchestration of Pervasive Services. *24th IEEE International Conference*

on Advanced Information Networking and Applications, pages 166–173, 2010. doi: 10.1109/AINA.2010.100.

[Nah(2004)] FFH Nah. A study on tolerable waiting time how long are Web users willing to wait? *Behaviour & Information Technology*, pages 1–37, 2004.

[Newman and Kotonya(2012)] Peter Newman and Gerald Kotonya. A Runtime Resource-aware Architecture for Service-oriented Embedded Systems. *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pages 61–70, August 2012. doi: 10.1109/WICSA-ECSA.212.14.

[Ngan and Kanagasabai(2012)] Le Duy Ngan and Rajaraman Kanagasabai. Semantic Web service discovery: state-of-the-art and research challenges. *Personal and Ubiquitous Computing*, 17(8):1741–1752, September 2012. ISSN 1617-4909. doi: 10.1007/s00779-012-0609-z.

[Niida et al.(2010)] Sumaru Niida, Satoshi Uemura, and Hajime Nakamura. User Tolerance for Waiting Time. *Vehicular technology magazine*, (September):61–67, 2010.

[Ns-3(2015)] Ns-3. ns-3 Tutorial: A Discrete-Event Network Simulator, 2015. URL `https://www.nsnam.org/docs/release/3.23/tutorial/html/introduction.html#about-ns3`.

[Oh et al.(2008)] Seog-Chan Oh, D. Lee, and S.R.T. Kumara. Effective web service composition in diverse and large-scale service networks. *Services Computing, IEEE Transactions on*, 1(1):15–32, Jan 2008. ISSN 1939-1374. doi: 10.1109/TSC.2008.1.

[Okediran(2014)] O. O Okediran, O. T Arulogun , and R. A Ganiyu. Mobile Operating Systems and Application Development Platforms : A Survey. *Int. J. Advanced Networking and Applications*, 2201(2014):2195–2201, 2014.

[Pandey and Nakra(2014)] Mithilesh Pandey and Neelam Nakra. Consumer Preference Towards Smartphone Brands , with Special Reference to Android Operating System. *IUP Journal of Marketing Management*, 2014.

[Park and Shin(2006)] Eunjeong Park and Heonshik Shin. Multimedia service composition for context-aware mobile computing. *Advances in Multimedia Modeling*, 2006.

[Perera et al.(2015)] Charith Perera, Chi Harold Liu Member, Srimal Jayawardena, and Min Chen. Context-aware Computing in the Internet of Things: A Survey on Internet of Things From Industrial Market Perspective. pages 1–19, 2015. ISSN 2169-3536. doi: 10.1109/ACCESS.2015.2389854.

[Perkins and Belding-Royer(1999)] C. E. Perkins and E. M. Belding-Royer. Ad-hoc on-demand distance vector routing. *Proc. 2nd IEEE Workshop Mobile Comput. Syst. Appl.*, pp. 90-100, 1999

[Pigadas et al.(2011)] Vassilis Pigadas, Charalampos Doukas, Vassilis P. Plagianakos, and Ilias Maglogiannis. Enabling constant monitoring of chronic patient using Android smart phones. *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments - PETRA '11*, page 1, 2011. doi: 10.1145/2141622.2141697.

[Pirrò et al.(2012)] Giuseppe Pirrò, Domenico Talia, and Paolo Trunfio. A DHT-based semantic overlay network for service discovery. *Future Generation Computer Systems*, 28(4):689–707, April 2012. ISSN 0167739X. doi: 10.1016/j.future.2011.11.007.

[Poizat and Yan(2010)] Pascal Poizat and Yuhong Yan. Adaptive composition of conversational services through graph planning encoding. 6416:35–50, 2010. doi: 10.1007/978-3-642-16561-0_11.

[Postolache et al.(2011)] Octavian Postolache, Pedro S. Girao, Mario Ribeiro, Marco Guerra, Joao Pincho, Fernando Santiago, and Antonio Pena. Enabling telecare assessment with pervasive sensing and Android OS smartphone. *MeMeA 2011 - 2011 IEEE International Symposium on Medical Measurements and Applications, Proceedings*, 2011. doi: 10.1109/MeMeA.2011.5966761.

[Prinz et al.(2008)] Vivian Prinz, Florian Fuchs, Peter Ruppel, Christoph Gerdes, and Alan Southall. Adaptive and fault-tolerant service composition in peer-to-peer systems. 5053:30–43, 2008. doi: 10.1007/978-3-540-68642-2_3.

[Prochart et al.(2007)] Guenter Prochart, Reinhold Weiss, Reiner Schmid, and Gerald Kaefer. Fuzzy-based support for service composition in mobile ad hoc networks. pages 379–384, July 2007. doi: 10.1109/PERSER.2007.4283943.

[Rambold et al.(2009)] Michael Rambold, Holger Kasinger, Florian Lautenbacher, and Bernhard Bauer. Towards autonomic service discovery - A survey and comparison.

SCC 2009 - 2009 IEEE International Conference on Services Computing, (Section II): 192–201, 2009. doi: 10.1109/SCC.2009.59.

[Raychoudhury et al.(2013)] Vaskar Raychoudhury, Jiannong Cao, Mohan Kumar, and Daqiang Zhang. Middleware for pervasive computing: A survey. *Pervasive and Mobile Computing*, 9(2):177–200, April 2013.

[Ren et al.(2011)] Kaijun Ren, Nong Xiao, and Jinjun Chen. Building quick service query list using wordnet and multiple heterogeneous ontologies toward more realistic service composition. *Services Computing, IEEE Transactions on*, 4(3):216–229, July 2011. ISSN 1939-1374. doi: 10.1109/TSC.2010.24.

[Ridhawi and Karmouch(2015)] Yousif Al Ridhawi and Ahmed Karmouch. Decentralized plan-free semantic-based service composition in mobile networks. *Services Computing, IEEE Transactions on*, 8(1):17–31, Jan 2015. ISSN 1939-1374. doi: 10.1109/TSC.2013.2297114.

[Rodriguez-mier et al.(2012)] Pablo Rodriguez-mier, Manuel Mucientes, and Manuel Lama. A Dynamic QoS-Aware Semantic Web Service Composition Algorithm. *Service-Oriented Computing Lecture Notes in Computer Science*, 7636:623–630, 2012.

[Sadagopan and Bai(2003)] Narayanan Sadagopan and Fan Bai. PATHS: analysis of PATH duration statistics and their impact on reactive MANET routing protocols. *Proceedings of the 4th MobiHoc*, pages 245–256, 2003.

[Sadiq et al.(2014)] Umair Sadiq, Mohan Kumar, Andrea Passarella, and Marco Conti. Service Composition in Opportunistic Networks: A Load and Mobility Aware Solution. *IEEE Transactions on Computers*, 9340(SEPTEMBER 2014):1–1, 2014. ISSN 0018-9340. doi: 10.1109/TC.2014.2360544.

[Schuler et al.(2004)] Christoph Schuler, Roger Weber, Heiko Schuldt, and Hans-J. Schek. Scalable peer-to-peer process management-the OSIRIS approach. *Web Service*, 2004.

[Sen et al.(2008)] Rohan Sen, GC Roman, and Christopher Gill. Cian: A workflow engine for manets. *Coordination Models and Languages*, pages 280–295, 2008.

[Shaffer and Keaveney(2012)] JD Shaffer and JF Keaveney. Automated concierge system and method. *US Patent 8,160,614*, 2(12), 2012.

[Silas et al.(2012)] Salaja Silas, Kirubakaran Ezra, and Elijah Blessing Rajsingh. A novel fault tolerant service selection framework for pervasive computing. *Human-centric Computing and Information Sciences*, 2(1):5, 2012. ISSN 2192-1962. doi: 10.1186/2192-1962-2-5.

[Sousa et al.(2006)] João Pedro Sousa, Vahe Poladian, David Garlan, Bradley Schmerl, and Mary Shaw. Task-based adaptation for ubiquitous computing. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(3): 328–340, May 2006. ISSN 1094-6977. doi: 10.1109/TSMCC.2006.871588.

[Su and Guo(2008)] Jian Su and Wei Guo. A survey of service discovery protocols for mobile ad hoc networks. *2008 International Conference on Communications, Circuits and Systems*, pages 398–404, 2008. doi: 10.1109/ICCCAS.2008.4657801.

[Thomas et al.(2009)] Louis Thomas, Justin Wilson, Gruia-Catalin Roman, and Christopher Gill. Achieving coordination through dynamic construction of open workflows. pages 14:1–14:20, 2009.

[Tyan and Mahmoud(2005)] Jerry Tyan and QH Mahmoud. A comprehensive service discovery solution for mobile ad hoc networks. *Mobile Networks and Applications*, pages 423–434, 2005.

[Ukey et al.(2010)] Nilesh Ukey, Rajdeep Niyogi, Alfredo Milani, and Kuldip Singh. A bidirectional heuristic search technique for web service composition. pages 309–320, 2010. doi: 10.1007/978-3-642-12189-0_27.

[Vukovi and Robinson(2007)] Maja Vukovi and Peter Robinson. Application development powered by rapid , on-demand service composition. 2007.

[Wang et al.(2013)] Hongbing Wang, Haixia Sun, and Qi Yu. Reliable Service Composition via Automatic QoS Prediction. *2013 IEEE International Conference on Services Computing*, pages 200–207, June 2013. doi: 10.1109/SCC.2013.45.

[Weiser(1991)] M Weiser. The computer for the 21st century. *Scientific american*, 3(3), 1991.

[Wohlin et al.(2003a)] Claes Wohlin, Martin Höst, and Kennet Henningsson. Empirical research methods in software engineering. *Empirical Methods and Studies in Software Engineering*, 2765:7–23, 2003a.

[Wohlin et al.(2003b)] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. Experimentation in Software Engineering. *Springer Science and Business Media*, pages 7–23, 2003b.

[Wu and Huang(2015)] Huijun Wu and Dijiang Huang. Mosec: Mobile-cloud service composition. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on*, pages 177–182, March 2015. doi: 10.1109/MobileCloud.2015.29.

[Yu(2009)] Weihai Yu. Scalable Services Orchestration with Continuation-Passing Messaging. *2009 First International Conference on Intensive Applications and Services*, pages 59–64, April 2009.

[Zaplata and Hamann(2010)] Sonja Zaplata and Kristof Hamann. Flexible execution of distributed business processes based on process instance migration. *Journal of Systems Integration, Vol 1, No 3*, pages 3–16, 2010.

[Zhenghui et al.(2009)] Wang Zhenghui, Xu Tianyin, Qian Zhuzhong, Lu Sanglu, Zhenghui Wang, Tianyin Xu, Zhuzhong Qian, and Sanglu 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*, number 3, pages 543–548. Ieee, March 2009.

[Zhou et al.(2011)] Xi Zhou, Yifan Ge, Xuxu Chen, Yinan Jing, and Weiwei Sun. A Distributed Cache Based Reliable Service Execution and Recovery Approach in MANETs. *2011 IEEE Asia-Pacific Services Computing Conference*, pages 298–305, December 2011.

[Zhu et al.(2003)] Feng Zhu, Matt Mutka, and Lionel Ni. Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services. *PerCom-Proceedings of the IEEE International Conference on Pervasive Computing*, 2003.

[Zisman et al.(2013)] A. Zisman, G. Spanoudakis, J. Dooley, and I. Siveroni. Proactive and reactive runtime service discovery: A framework and its evaluation. *IEEE Transactions on Software Engineering*, 39(7):954–974, July 2013. ISSN 0098-5589. doi: 10.1109/TSE.2012.84.

[Zou et al.(2014)] Guobing Zou, Yanglan Gan, Yixin Chen, Bofeng Zhang, Ruoyun Huang, You Xu, and Yang Xiang. Towards automated choreography of Web services

using planning in large scale service repositories. *Applied Intelligence*, 41(2):383–404, March 2014.