Service composition in opportunistic networks

Fadhlallah Baklouti

THÈSE DE DOCTORAT DE

L'UNIVERSITE BRETAGNE SUD
COMUE **U**NIVERSITE **B**RETAGNE **L**OIRE

Ecole Doctorale N°601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *INFO : Informatique*
Par:

Fadhl Allah Saddam BAKLOUTI

Service composition in opportunistic networks

Composition de services dans les réseaux opportunistes

Thèse présentée et soutenue à VANNES , 1 mars 2019
(UMR 6074) IRISA (Institut de recherche en informatique et systèmes aléatoires)
Thèse N° : 523

Rapporteurs avant soutenance :
Chantal TACONET, Maître de Conférences HDR, Télécom SudParis, Institut Mines Télécom
Abderrahim BENSLIMANE, Professeur des Universités, Université d'Avignon

Composition du jury :

Président :	Pierre-François MARTEAU, Professeur des Universités,Université de Bretagne-Sud
Examinateurs :	Nathalie MITTON, Directrice de Recherche INRIA, INRIA Lille-Nord Europe
	Chantal TACONET, Maître de Conférences HDR, Télécom SudParis, Institut Mines Télécom
	Abderrahim BENSLIMANE, Professeur des Universités, Université d'Avignon

Dir. de thèse :	Yves MAHÉO, Maître de Conférences HDR, Université de Bretagne-Sud
Encadrant de thèse :	Nicolas LE SOMMER, Maître de Conférences, Université de Bretagne-Sud

Contents

List of Figures

List of Tables

Part I

Introduction and Related Works

Chapter 1

Introduction

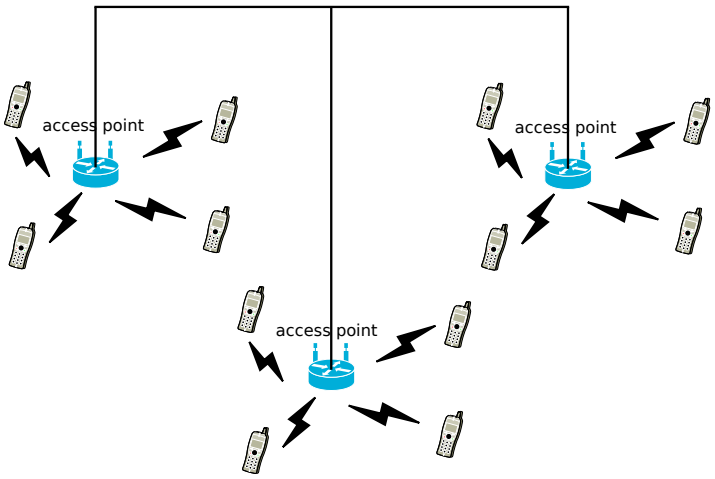
Contents

1.1 Background and motivation

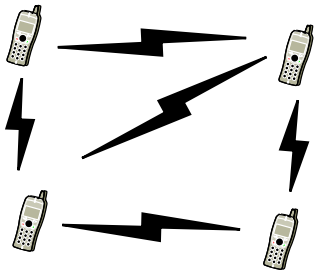
Nowadays, our environment is populated with heterogeneous electronic devices (PC, smart-phones, tablets, sensors, actuators, etc.) which are used for different kinds of purposes (*e.g.* environment sensing, smart home control, social communication).

Usually, these devices are connected to a network infrastructure using equipments such as Wi-Fi access points or base stations (BTS) as shown in Figure 1.1a. The cost of such infrastructures can be very expensive. This is the main reason why these infrastructures are not usually deployed in remote areas with small populations and in extremely poor countries. Furthermore, these infrastructures could cease to function in certain situations such as when a considerable damage is caused by catastrophes (Hurricanes, volcanoes, etc.). They can, as well, be subject to censorship for political reasons. In addition, such infrastructures are vulnerable to a single point of failure in case, an access point or a BTS breaks down.

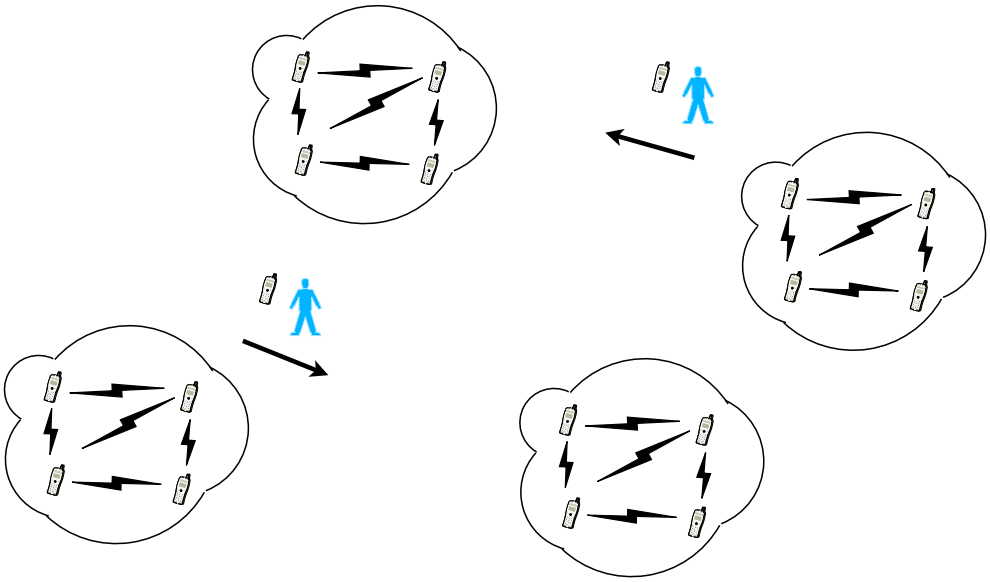
One solution to solve this problem is to enable device-to-device communication. In this communication mode, devices exchange data with each other directly without resorting to an infrastructure that plays the role of intermediate between them, as shown in Figure 1.1b. This communication mode is also referred to as ad hoc mode. Networks formed by mobile devices communicating in ad hoc mode are called *Mobile Ad Hoc Networks (MANETs) (32).* In these networks, devices play simultaneously the role of a host and a router to be able to cope with the fast changing topology of the network.



(a) Infrastructure-based communication.



(b) Ad hoc communication.



(c) Fragmented network.

Figure 1.1 – Wireless networks.

In MANETs, a dynamic routing protocol is needed to enable nodes to communicate with each other. Dynamic routing protocols determine the set of intermediate nodes that will transmit packets from a source to a destination. Routing protocols can be split into two categories: *table driven/proactive protocol*s and *reactive protocols*. The *proactive ones* maintain a table that contains routing informations about other devices in the network. This table is updated regularly. DSDV (89) is an example of such a type of proactive protocols. On the other hand, *reactive protocols* perform an on demand discovery and computation of routes. A node, that has packets to send to a certain destination, starts a route discovery process, and when a route is found the sending process can take place. AODV (98) is an example of a reactive routing protocol.

Unfortunately, MANET routing protocols are likely to prove inefficient in real conditions. Indeed, the objective of MANET routing protocols is to provide an end-to-end route between each pair of nodes in order that conventional applications can still run in these kind of environments. Thus, these protocols assume that the network is enough dense, stable and connected in order to guarantee the existence of these end-to-end routes. However, due to the mobility of devices, to the limited radio range of network technologies, and to the unpredictable failures, the network topology has probably a fragmented aspect, as shown in Figure 1.1c, where devices form a set of isolated connected islands.

Highly fragmented and disconnected MANETs constitute what we call *opportunistic networks (OppNet) (96).* OppNets are the target networks of this dissertation. Unlike MANETs, OppNets do not assume the existence of an end-to-end route between each and every pair of devices. Thus, OppNets do not rely on dynamic routing protocols. Instead, OppNets consider a contact established between two nodes as an opportunity to exchange data. OppNets rely on the "store, carry and forward" principle, which consists of storing messages in a cache memory and exchanging them with other devices whenever it is possible. OppNets also take advantage of device mobility in order to deliver data across the network.

OppNets are further extended with the concept of *Opportunistic computing (30),* where a user not only has access to the local resources of his device but also to resources offered by the devices of other users he encounters, and by the devices deployed in the network such as infostations. Doing so, a given user has more functionalities at his disposal and has the ability to combine them to create new ones.

Moreover, the techniques, provided by both opportunistic networking and opportunistic computing, have opened multiple new interesting perspectives for ubiquitous computing (122) and Internet of Things (IoT) (9). As an example, we can consider a mobile sensor network to monitor a wild animal population and to study their behaviours as in the project ZebraNet (62), where animals are equipped with wireless devices that embark GPS modules, flash memories, transceivers and small CPUs. Another example is the project ASAWoO (87) that uses the concept of Web of Things. In ASAWoO, physical objects are associated with avatars (Web-based software components) in order to be able to control and access these objects using Web standards.

One way to implement opportunistic computing is to adopt an approach based on

(a) Attraction park.

(b) Marathon event.

(c) Composition scenario in a marathon event.

Figure 1.2 – Scenarios suitable for opportunistic services.

service-oriented computing (SOC) (17; 40), which is the one chosen in this dissertation. In SOC, each resource (software or hardware) from a given device in the network can be wrapped in a form of a service, and made publicly accessible through a unique interface with a service description that provides service clients with the informations they need to invoke these services. SOC guarantees loose coupling and late binding between software components and allows them to evolve independently from each others. Moreover, one of the most important principle of SOC is service reusability. SOC requires services to be designed in way that supports reuse even if reuse opportunities are not immediately available. One example of reusability, that we focus on in this thesis, is service composition, where atomic services are aggregated into composite services in order to create new functionalities.

Opportunistic computing based on service-oriented computing can be applied in many applications and scenarios such as social networking, disaster relief and data collection. One example could be a marathon event (Figure 1.2b) where runners wear RFID tags on their bibs. Consequently, a service can be developed to read these RFID tags in order to add bib numbers on photos as metadata (S1 in Figure 1.2c). This service can be, then, composed with a service that applies filters to photos (S2 in Figure 1.2c). This service is possibly hosted by another node, due to the lack of resources. Another service could be added to indicate the location coordinates where these photos were taken (S3 in Figure 1.2c). This sequence of services, invoked one after the other, illustrates the principle of reusability based on composition. Likewise, users can share and comment photos of their favourite runners and follow them closely throughout the race.

Moreover, we can imagine a scenario where people, in an attraction park (Figure 1.2a), wish to share informations and feedbacks about their experiences. By relying

on smart-phones carried by people and on several other devices that the park administration may provide, services may be deployed to produce pictures, comments or measurements (temperature, queue length, waiting duration, etc).

1.2 Challenges

This thesis focuses on opportunistic networks and on the opportunistic computing paradigm (based on SOC). To devise solutions dedicated to this kind of opportunistic environments, many challenges and constraints should be taken into consideration. Some of these challenges would be considered trivial and would be simply ignored in other types of networks such as MANETs. In this section, we outline the main challenges and constraints that both opportunistic networking and computing techniques should cope with.

1.2.1 Opportunistic networking challenges

Throughout the literature, most of the solutions and studies on opportunistic networks are focused on forwarding algorithms. Generally the goal of a forwarding algorithm is to maximize message delivery while minimizing delivery delays. In this context, Thrasyvoulos et al. (116) identify 3 categories of constraints that a forwarding algorithm should take into consideration: (i) the uncertainty and the stochastic aspect of connectivity links, (ii) the randomness, the heterogeneity and the unpredictability of nodes mobility and (iii) the limited and heterogeneous resources of each node. Hereafter, we present these 3 categories in details.

Connectivity Yu et al. (124) provide two definitions for connectivity:(i) the probability that an end-to-end path exists between two random nodes or (ii) the percentage of nodes connected to the largest connected subgraph. Due to the mobility, to the short radio range and to the lack of resources, connectivity is rarely at a 100 % rate. Usually, it varies between 0 % (very sparse and fragmented) and 100 % (compact and condensed). In this same logic, the authors of (42) propose a classification of 3 types of networks based on their connectivity: *Sparse Networks, clusters networks* and *almost connected networks.*

Sparse Networks are the type of networks were nodes do not form any large cluster. Indeed, most of the nodes are isolated most of time and sometimes they come across other nodes. Nodes exploit their limited contact duration to exchange data. The probability of having an end-to-end path in this type of networks is close to 0, thus rendering conventional routing protocols inefficient.

The *clusters networks* can be considered as the most realistic ones. Indeed, in the real world, nodes tend to organize themselves in groups. Many examples can be considered such as a number of vehicles around a crossroad or in front of a traffic light (106), a group of students in a campus (54), or even a group of animals that move together (62) (Zebra herd or lion pride). Consequently, these nodes form a set of separated connected

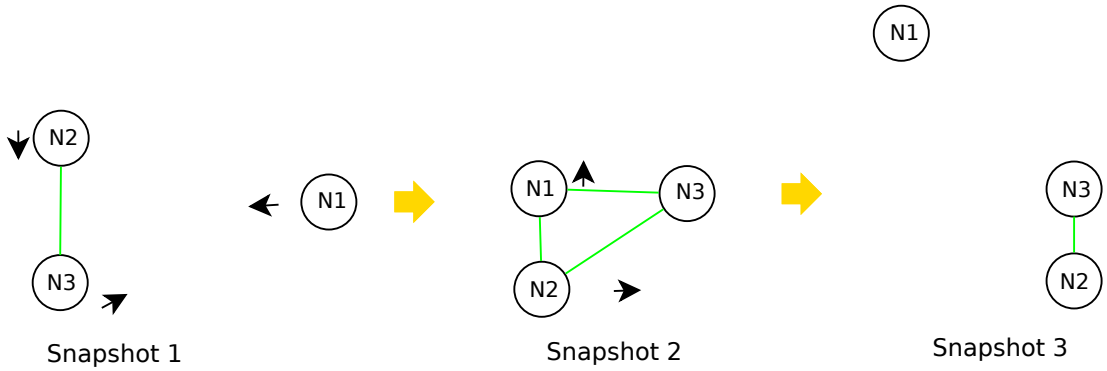


Figure 1.3 – Connectivity graph snapshots.

clusters. The nodes within the same cluster usually have a high connectivity. However connectivity between clusters is very low and data exchange between them relies usually on nodes leaving one cluster and joining another.

Sparse Networks and *clusters networks* represent the two types of networks where conventional routing protocols would fail and where opportunistic forwarding techniques should provide the alternative. Therefore, it is important for opportunistic forwarding techniques to figure out how to optimize data delivery between isolated nodes and how to efficiently relay data between clusters of nodes.

The third type is the *almost connected networks*. This type is characterized by a high node density with a few small isolated clusters or solitary nodes. An end-to-end path is likely to be found between a random pair of nodes, even though due to mobility and link quality fluctuations, this end-to-end path has in general a short lifetime. This type of networks is suitable for conventional ad hoc routing protocols (proactive (89) or reactive (98)). The only case, where opportunistic networking techniques can still be relevant in this type of networks, is when we want to connect isolated nodes and clusters to the rest of the network.

Mobility Mobility describes how nodes move around. It is the factor that determines how the topology of a wireless network evolves overtime. In other words, mobility defines the sequence of the connectivity graph snapshots of the network and how these snapshots are related to each others, as illustrated in Figure 1.3. Therefore, in the absence of any end-to-end routes that can be relied on, it is important to study mobility patterns and their stochastic properties in order to devise robust forwarding algorithms for opportunistic networks. In (116), the authors identify 5 mobility properties that might be relevant for forwarding algorithms, namely intensity, locality, regularity, heterogeneity and correlation.

Intensity is related to the speed, and to the frequency and duration of pauses. *Locality* is the set of locations that a given node prefers to visit. As shown in studies like (59), nodes prefer to visit a small number of locations over a large amount of time. *Regularity* expresses the order, in terms of importance, according to which a node visits its

most frequented locations. *Heterogeneity* underlines the fact that even though nodes have same qualitative properties (e.g. limited number of frequented locations), they often exhibit different choices (e.g. different frequented locations.). *Correlation* identifies nodes that have a correlated mobility pattern. For example classmates that are sharing the same classroom at a certain time of the day.

Resources Despite the important technological advancements, scarcity of resources, in wireless networks in general and in opportunistic networks in particular, remains an important challenge to reckon with. We identify two resources constraints: *network resources* and *local resources.*

As far as the *network resources* are concerned, opportunistic networks inherit the very same limitations from the traditional wireless networks. Opportunistic networks will always suffer from limited bandwidth and relatively high probability of link errors. Therefore forwarding solutions should be devised in a way that reduces signaling and control information exchanges. Furthermore, due to bandwidth scarcity, forwarding algorithms should be very prudent with their forwarding decisions.

Local resources depend on the type of nodes forming the networks. Networks, such as hybrid networks (12), can contain nodes that have relatively significant resources like infostations as well as nodes with very limited resources like sensors or mobile devices. Depending on the type of nodes taken into consideration, important resources like processing power, memory, storage capacity and battery lifetime (5) could vary dramatically. Therefore, opportunistic techniques should be aware of the amount of resources that are available in order to function properly, especially in the case of scarcity, where resources should be used carefully.

1.2.2 Opportunistic computing challenges

Developing applications for opportunistic networks can reveal to be a very complicated task. Indeed, opportunistic applications can not make the same flexible assumptions about their environment as conventional applications because of the volatility and the unpredictability of opportunistic networks. The main objective, when developing applications for opportunistic networks, is to maintain a certain quality of service while coping with the problems imposed by such difficult environments. Opportunistic applications inherit challenges from applications developed for MANETs and have to deal with new challenges imposed by the nature of opportunistic networks.

A common challenge, that opportunistic applications have with other wireless applications, as well as with opportunistic forwarding techniques, is the scarcity of resources. Most of mobile devices, composing opportunistic networks, have indeed a limited processing power, a small memory and storage space, and a relatively short battery lifetime. Besides, these devices suffer from link errors and limited bandwidth which can cause connection disruptions. Therefore opportunistic applications should be lightweight and economical with their local and network resource consumption.

Opportunistic applications should also take into consideration that their network exchanges could fail and that they need to implement recovery mechanisms.

Opportunistic applications should also take into consideration the scalability problem. As for any other type of distributed applications, the increase in the number of nodes should not affect the performance of the application overall.

Similar to opportunistic networking techniques, opportunistic applications have to cope with the fragmented aspect of the network. Actually, opportunistic networks are generally formed by isolated nodes or by isolated group of nodes. Nodes, within the same group, generally have no problem reaching each others. However, it can be difficult to reach an isolated node or a node in another group. Opportunistic applications should take into consideration that accessing the resources of a node in another group could introduce long waiting delays and sometimes could even fail.

Opportunistic applications based on SOC, in particular, face important challenges when performing service discovery, invocation and selection. Indeed, these applications should take into consideration, when selecting service providers (devices hosting services) to invoke, that these providers are only available for a short amount of time due to mobility and short radio range. Besides, communication delays are inherent to opportunistic networks, due to the adoption of the "store-carry-and-forward" principle in the absence of end-to-end routes. Therefore, service invocations should be asynchronous, and selection should favor providers that are likely to offer a minimum of invocation delay. Similar to other opportunistic applications, opportunistic SOC applications should take into account connection disruptions especially when performing an invocation process. Thus, opportunistic SOC applications should implement recovery mechanisms such as invoking an alternative provider or invoking a group of providers instead of a single one in order to minimize the probability of failure. Moreover, service discovery in opportunistic networks should focus on advertising services to clients that are likely to be able to invoke these services successfully while avoiding those that are not, in order to guarantee a certain service provision quality and to reduce bandwidth consumption.

1.3 Objectives and Contributions

In this thesis, we consider studying the service-oriented computing paradigm applied to opportunistic networks. We try to find a way to exploit this paradigm in order to create new functionalities, not provided by elementary services, using service composition. In this context, we suppose that there are enough service redundancy (instances per service) and enough service diversity (number of services available) in the network in order to be able to compose new services. In this thesis, we only consider sequential compositions. Other types of compositions can, indeed, be derived from the sequential ones. Therefore, our objective is to define a composition mechanism that execute composition requests with the least time possible while maximizing the composition success ratio. Our composition solution should be also efficient with resource consumption, given the scarcity of such resources in opportunistic networks.

Hereafter, we detail our principle contributions in this dissertation.

Service discovery and composition system We define a service discovery and composition system that has two main modules: a service discovery module and a service composition module. The service discovery module implements a discovery algorithm that follows a directory-less approach where each device implements it own service registry. This algorithm also uses a proactive discovery mode. The service composition module allows the use of both an orchestration strategy and a choreography strategy.

Utility function for provider selection We define a utility function that selects service providers to be enrolled in an invocation of a simple or a composite service. We also propose two implementations for this utility function: one is based on time and the other is based on location and distance.

Proactive service precomputing manager To reduce composition time and offer the user a better quality of service, we propose to compose services proactively without waiting for the user to decide to start composition requests explicitly. Since, it could be expensive to execute all possible compositions, we have devised a solution that automates the process of composition by identifying and executing composition requests that reflect the user preferences. We call this solution proactive service precomputing manager (PSP).

Distributed service composition caching Since we rely on opportunistic networking techniques, each node is supposed to have a cache memory in which it stores messages. Thus, we propose a solution that considers these cache memories as a distributed storage space that provides access to invocation and composition results from other nodes in the network instead of initiating new invocations of simple or composite services. Doing so, the workload and the resource consumption of nodes are reduced, and results can be returned to the clients more quickly. This solution is called distributed cache manager (DCM) and it represents an extension module to the service discovery and composition system.

Evaluation results We run several sets of real time emulations in order to compare the performances of both composition strategies (choreography and orchestration) mainly in terms of success ratio and composition time. We do the same, as well, to compare the two implementations of the utility function in order to study their effects on service composition. We also evaluate the performances of the DCM.

1.4 Outline of the thesis

The rest of this dissertation is organized as follows:

In Chapter 2, we introduce both the opportunistic networking paradigm and the opportunistic computing paradigm. We present the forwarding techniques dedicated to this kind of networks and some of the applications and middlewares that adopt an opportunistic computing approach.

In Chapter 3, we introduce the concept of service-oriented computing (SOC) and we discuss the main notions related to it such as service discovery, invocation, selection and composition. We also survey the works related to SOC both in MANET and in opportunistic networks.

In Chapter 4, we present the service discovery and composition system. We define the two implementations of the utility function (location-based and the time-based) that are used in order to rate service providers and to select them for invocations of either simple or composite services. We also present, in this chapter, the two composition strategies (orchestration, choreography) and we provide a mathematical formulation in order to estimate the composition time and the success ratio. In Chapter 5, we present the proactive service precomputing manager that automates service compositions based on the user preferences, and the distributed cache manager that shares and replicates composition results between devices.

In Chapter 6, we present an implementation of the service discovery and composition system as well as the software environment used in the development process. In Chapter 7 and 8, we compare the orchestration strategy and the choreography strategy using a set of emulations, and we study the impact of both implementations of the utility function on the invocation of composite services. In Chapter 9, we evaluate the performances of the distributed cache manager. Finally, we present our general conclusions as well as the perspectives for future works.

Chapter 2

Opportunistic networking and computing

Contents

2.1 Introduction

In (42), Kevin Fall states that conventional network protocols are based on the assumptions that there is always an end-to-end path between each and every pair of nodes in the networks, that the transmission delay between a sender and a receiver is relatively short, and that error rate and message loss are very low. He calls each network that fails to meet these characteristics a *challenged network.* Challenged networks have been widely studied over the years, especially Delay-Tolerant Networks (DTN) (117) and opportunistic networks (96) that are the closest to the topic of this dissertation. DTN is considered as an overlay architecture that allows DTN regions (heterogeneous networks) to communicate with each others using DTN gateways that deliver data from one region to another, while opportunistic networks are more considered as disconnected mobile ad hoc networks.

In this chapter, we focus on the topic of opportunistic networks and opportunistic computing. We organize the rest of the chapter as follows. In Section 2.2, we present the paradigm of the opportunistic networking in general and we discuss the forwarding techniques dedicated to this kind of networks. In Section 2.3, we introduce the paradigm of opportunistic computing and we discuss several examples of platforms dedicated to this paradigm. Finally, in Section 2.4, we provide a discussion and we give some conclusions.

2.2 Opportunistic networking

The concept of opportunistic networks has been introduced to address the short-comings of Mobile Ad hoc Networks' solutions. Most of the proposed solutions related to MANETs consider that the network is enough dense to overcome the limited radio range and the user mobility. Thus, the network guarantees the existence of an end-to-end path between each and every two nodes, which constitutes the main assumption upon which almost every MANET routing protocol is conceived. In real scenarios, such assumptions are not sustainable. Therefore, opportunistic networks rule out the existence of an end-to-end path between each pair of nodes. Instead, opportunistic networks rely on mobility that enables nodes to come at range of one another for relatively short contact durations. Nodes then exploit these contact durations in order to exchange data. To implement this approach, opportunistic networks rely on the principle of "store, carry and forward". Basically, when a given node fails to find a suitable destination to forward data to, it stores the data in a local cache and waits until it finds an opportunity to send it.

To summarize, a network is considered to be opportunistic if:

- most of the nodes are mobile and their mobility pattern is hard to predict,
- connection between nodes is in ad hoc mode with no infrastructure or access point to manage it,
- the absence of end-to-end path between nodes,
- relatively long transmission delays.

Most of the works done on opportunistic networks deal with the problem of data forwarding (84). Usually, in the literature, we use the term "forwarding algorithm" rather than "routing algorithm" in the context of opportunistic networks. The reason is the absence of a route between the sender and the receiver in most cases. Consequently, we focus on finding the best way to forward messages to their destinations instead of establishing a connected route between each pair of nodes. Forwarding algorithms can be classified in different ways. The authors of (84) propose to divide forwarding protocols into three categories based on whether or not these protocols make any use of context.

Context oblivious Context oblivious protocols do not exploit any kind of device-related or user-related information. They have no particular knowledge about the network or its topology, and they usually forward messages using a fixed mechanism that does not change its behaviour no matter what the context is.

An example of this category of protocols is the Epidemic protocol (119). This protocol forwards messages as a disease is spread among a population. Each node maintains a list of messages, that it carries in its cache, in a form of a hash table. The hash table is described using a summary vector. Upon contact with another node, the summary

vectors are exchanged between the pair. Each node determines the messages that it does not have and then requests them from the other node.

Flooding-based algorithms like Epidemic generally generate a huge overhead and are very bandwidth and energy consuming. This can cause a considerable problems for resource-limited nodes. In an attempt to remedy to these problems, the idea was to limit the number of copies of a given message that flow in the network. This is the underlying approach behind the spray and wait algorithm (114). Spray and wait consists of two main steps: (i) Spray phase: each source node creates L copies and forwards them. Each forwarding node, that has n>1 copies, will forward one or many copies to the next neighbours and so on. If it has only one copy, it can only perform a direct transmission to the destination. (ii) Wait phase: if the destination is not reached in the spraying phase, the nodes carrying the copies will perform a direct transmission when they come in contact with the destination.

One of the draw-backs of spray and wait is that it needs a highly moving nodes to guarantee a good performance. Due to the network being relatively sparse, spray and wait can face a serious problem of node locality which will inhibit the process of delivering the messages. To answer these shortcomings, a solution called spray and focus (113) was introduced. Spray and focus also has two phases: (i) Spraying phase: when a source generates a new message, it creates with it L tokens. A token means that the node, possessing it, can spawn and forward an additional copy of the message. A node can transmit a message, with n>1 tokens, to another node that does not have a copy of it. It is done by simply spawning the message and forwarding it with n/2 tokens. When n=1, we will use the focus phase. (ii) Unlike in Spray and wait, in this focus phase, when a node carries a message with one single token, it can forward it to another relay based on a specific criterion. A utility function is used to assess whether a given node is suitable as a relay or not. This utility function is based on a timer that calculates the elapsed time since the last encounter with a given node.

Partially context-aware Partially context-aware protocols collect informations about contacts between nodes and usually attempt to speculate or predict, in someway, their mobility in order to select the nodes, to which messages should be forwarded.

PROPHET (Probabilistic Routing Protocol using History of Encounters and Transitivity) (76) is an example of a partially context-aware protocol. This forwarding protocol relies on assessing the probability of a node to deliver messages. It uses a probabilistic metric called “delivery predictability” that indicates if a given node is likely to successfully deliver a message to a certain destination. PROPHET assumes that the nodes’ mobility is not random and that it is possible to predict it.

CAR (Context-Aware Routing) (88) tries to predict if the destination node is within the same cloud (cluster of nodes) as the sender. If it is the case, the message will be forwarded using the proactive routing protocol DSDV. Otherwise, CAR selects one or more carriers that are considered to have the highest chance of delivering the message according to a metric called *delivery probability* that is calculated using context informations. CAR uses as context properties the number of neighbours of a given node and

its current energy level. CAR is considered as a partially context-aware due to the fact that collecting informations about the network does not require an explicit exchange of messages.

Full context-aware Full context-aware protocols exploit informations about the user's environment like location, interest, social interaction and context, and do not restrain themselves to only informations about the network as the partially context-aware protocols do.

In this category, we find a protocol called HiBOp (20) proposed by Boldrini et al.. HiBOp relies on network topology, contact history and user context in order to establish similarity between nodes and to forward messages between them accordingly. HiBOp requires users to provide their personal data like addresses, hobbies, phone numbers, emails, etc. HiBOp makes the assumption that nodes with more similarities with the destination, have more chances to deliver messages to it.

PROPICMAN (90) is a probabilistic forwarding protocol that exploits nodes' profiles to calculate their delivery probability for a certain destination. A node profile consists of a set of evidences (attributes). Each evidence is associated with a weight. These weights are then used to calculate the delivery probability. When a given source wants to send a message to a certain destination, it starts a two-hop route probability selection. This selection process consists of choosing the route formed by the two consecutive neighbours (1-hop neighbour and 2-hop neighbour) that have the highest combined delivery probability to the destination. Furthermore, PROPICMAN provides some security features to protect the content of messages and the users' informations.

SimBet (34) is another full context-aware forwarding protocol. It uses two metrics to assess the probability of a node to reach a certain destination: centrality and social similarity. SimBet assumes that nodes with high centrality can play the role of a broker in order to relay messages between disconnected node communities (nodes clusters), that can not communicate directly with each others. Moreover, within the same node community, Simbet uses the similarity metric to choose which nodes to carry a message to its destination. Actually, it is supposed that nodes with close similarities are more likely to find each others.

Hui et al. (56) present a social-based forwarding protocol called BUBBLE Rap. BUBBLE Rap relies on a community-based view of the network combined with a node centrality assessment approach to forward messages. BUBBLE Rap, first, assumes that people have different roles and popularity and that should be reflected by the network. Therefore, BUBBLE Rap first forwards messages to the most popular nodes until we reach a node that is in the same community as the destination. This first step is justified by arguing that popular nodes have more chances to contact other nodes and that these popular nodes are included in many communities at the sametime. Second, BUBBLE Rap recognizes that people organize themselves in communities which also should be reflected by the network they form. Thus, the second step consists of identifying the nodes that are in the same community as the destination and use them to forward the messages.

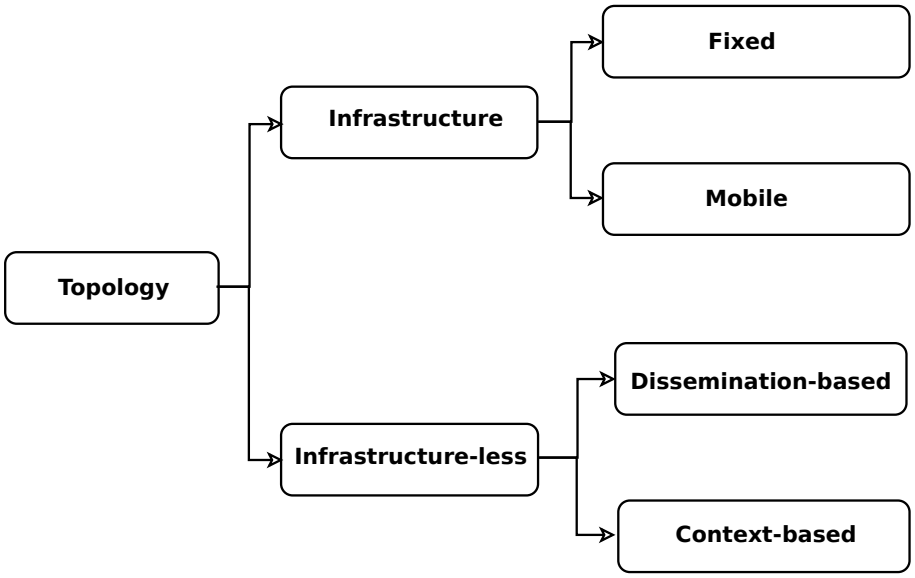


Figure 2.1 – Topology based classification

In (96), the authors propose another classification based on the topology assumed by the protocols, as shown in Figure 2.1:

- **With infrastructure:** The presence of an infrastructure can be exploited by forwarding protocols in order to forward the messages between nodes. Usually, the infrastructure, in this context, consists of a set of special nodes with more powerful resources than the normal ones. Generally, these special nodes collect the messages and try to deliver them. Furthermore, this class of protocol can be divided in two other subclasses:
  - **Fixed infrastructure:** Fixed infrastructure consists of a set of fixed base stations that collect messages and deliver them to more reliable networks (Internet, 3G,..). One example of protocols, that exploits this kind of infrastructure, is SWAM (112).
  - **Mobile infrastructure:** Mobile infrastructure consists of mobile data mules or ferries. Forwarding protocols use these data mules when they pass next to normal nodes to collect data. Data mules will then send this data to other networks via access points and gateways. One example of this kind of protocols is the data-Mule system (58).
- **Without infrastructure:** Protocols without infrastructure do not suppose the existence of any kind of special nodes in particular. This class of protocols can be further divided into two subclasses:
  - **Dissemination-based:** Dissemination-based protocols rely generally on flooding-based techniques. The most important thing is choosing the right heuri-

stics to decrease the considerable overhead due to flooding. It is important to mention that the overhead has a negative effect on the protocol performance since it consumes a lot of bandwidth, power and processing which are very scarce resources in opportunistic networks. Examples of protocols from this subclass: probabilistic routing (76), Spray and Wait (114), Spray and Focus (113) and epidemic routing (119).

– **Context-based:** Context-based protocols exploit informations about the node's environment such as location, user interest, history of pair-wise contact, etc. As examples, we can mention HiBOp (20) and MobiSpace (72).

2.3 Opportunistic computing

In (30), Conti et al. argue that two nodes, that come in contact within an opportunistic network, have the opportunity to exchange messages, and also to exploit each other resources and applications. This paradigm is called opportunistic computing by Conti et al..

Opportunistic computing is a relatively recent distributed computing para-digm that exploits unplanned and opportunistic interaction between mobile devices. Opportunistic computing is build upon opportunistic networks and tries to expand it by making it possible to leverage remote resources such as software applications, heterogeneous hardware, multimedia content and sensing capabilities. Likewise, a given device, that does not have enough resources to perform a certain task, can rely on remote resources provided by other devices to collaboratively perform this task using opportunistic computing techniques.

Several platforms have been developed specially for opportunistic computing. DoDWAN (52) (Document Dissemination in mobile Wireless Ad hoc Networks) is a Java-based middleware for opportunistic networks that adopts a content-based communication approach. Content-based communication focuses on delivering informations to any node interested in it rather than to a particular destination. DoDWAN is suitable to use with applications that require information exchange using a publish/subscribe programming model. Carzaniga et al. (24) and Costa et al. (33) provide similar protocols based on content-based communication.

Benchi et al. (15) present a middleware called JOMS which is a JMS implementation designed for opportunistic environments and based on DoDWAN. JOMS is formed by two layers: a lower layer that handles opportunistic communications based on a content-based communication approach, and an upper layer that implements a JMS provider and a local service directory compatible with JNDI API.

JION (14) is an implementation of JavaSpaces for opportunistic networks. JION is formed by two systems: a communication system and a JavaSpaces system. The communication system is based on DoDWAN. The JavaSpaces system, not only complies with the conventional JavaSpaces, but also adds to it asynchronous operations (work better in an opportunistic environments) like *readf()* and *takef()* that rely on the concept of future objects.

C3PO (67) is a framework that implements opportunistic networking techniques (store-carry-forward, forwarding algorithms). C3PO is intended to facilitate the development of spontaneous and ephemeral social networks used to cover events that are limited in space and time like sport events (marathons, rallies) or festivals. C3PO provides log messages that facilitate the evaluation procedure. C3PO also supports different communication technologies, that are adopted in the majority of smart-phones and wireless devices, such as Wi-Fi Direct and Bluetooth, without the need of rooting these devices. C3PO provides two communication modes: topics for content-based communication, and channels for destination-based communication. C3PO implements an optimized version of Epidemic routing to minimize the number of messages exchanged in the network. We provide more details on this framework in Chapter 6.

Haggle (108) is a network architecture for opportunistic networks that enables cross-layering communication. Unlike TCP/IP stack, Haggle is unlayered and it has 4 main modules: delivery, user data, network protocols and resource management. User data is not isolated from the network, which makes it shareable without the application level involvement. The application level also is free from protocol functionalities which simplify considerably its code. Haggle uses a user-level naming which provides compatibility with many network protocols. Finally, Haggle uses the resource management component to mediate between the three other components.

Boldrini et al. (19) present a middleware based on context and social awareness for opportunistic networks. This middleware exploits the Haggle architecture and adds to it a context manager for integration purposes. The authors focus on defining a context that suits opportunistic communication. They define three types of context: the *user context,* which describes the user's personal informations and his social interactions, the *service context,* which describes a given service provided by the middleware and *the device context*, which specifies the physical characteristics and limitations of the device. This middleware performs content sharing using an utility function. This utility function assesses the overall utility of the data carried by a given node based on how much this data is relevant to this node and to its entourage.

CAMEO (8) is also a context-aware middleware for opportunistic networks designed as a part of the project SCAMPI (29). CAMEO is intended to help develop applications for Mobile Social Networks (MSN). It provides them with a common application programming interface that enables them to exploit social-aware and context-aware functionalities. CAMEO uses a multidimensional context space that is based on data from the local device, from the local user, and from the interaction with the rest of the network.

Auzias et al. (10) propose a middleware for Internet of Things, that relies on delay-tolerant communication, called BOAP. BOAP consists of an integration layer between two protocols: COAP (111) and Bundle (109) protocol. COAP is a protocol that allows resource-limited machines to communicate asynchronously according to a RESTful style and using UDP protocol. Bundle protocol defines a format of messaging called Bundle dedicated to DTN networks.

	destination-based	content-based
forwarding algorithms (PROPHET, HiBOp, CAR, etc.)	x	
DoDWAN (52)		x
Haggle (108)		x
CAMEO (8)		x
Boldrini et al. (19)		x
C3PO (67)	x	x

Table 2.1 – Comparison between opportunistic solutions

2.4 Discussion and Conclusion

Based on the objectives of this thesis, forwarding protocols can provide some advantages but can also exhibit some drawbacks. Indeed, infrastructure-based forwarding protocols are not compatible with our objectives, since we make the assumption that each node can both provide and consume services. Infrastructure-based protocols are rather more adapted to a scenario where infostations provide services and the rest of the smaller nodes discover these services and invoke them.

Moreover, most of the forwarding protocols, especially the context-free and the partial context-aware, are destination-based, where the sender of the message explicitly specifies the address of the destination. The destination-based communication makes most of the forwarding protocols adequate for the process of service invocation (detailed in Chapter 3), where we send our invocation request to one or to a limited number of destinations. These forwarding protocols could prove to be inefficient in a service discovery or advertisement process (also detailed in Chapter 3), where the goal is to reach multiple nodes at the same time. Epidemic protocol constitutes an exception to this observation since it relies on disseminating messages to every node that does not have a copy of them. This protocol can be suitable for the discovery process, however it comes with a high cost of significant overload and resource consumption.

Full context algorithms, such as HiBOp, could be adapted for service advertisement and discovery purposes, since the context can be used to express the interest profile that indicates what kind of services a given node is interested in. Still, the best approach, to discover or advertise services, is the content-based. Solutions like DoDWAN, CAMEO and the one proposed by Boldrini et al. (19) do not target a single destination by using an address. Instead, they disseminate data to whichever node interested in it. Consequently, by using the content-based approach, services can be discovered by many interested nodes. Nonetheless, content-based communication can be inefficient with resource consumption when performing an invocation since we only target one or a limited number of destinations.

To summarize, we have come to the conclusion that destination-based communication are more suitable for invocations and that content-based communication are more suitable for discovery and advertisement. Both these types of communications are sup-

ported by the framework C3PO. Indeed, its communication mode using channel allows to send messages to a precise destination, and its communication mode using topics allows to disseminate messages to every node subscribed to the topics associated to these messages.

In this chapter, we presented the paradigm of opportunistic networking. We discussed several examples of forwarding protocols for opportunistic networks as well as some of the classifications from the literature. We, then, extended the discussion by presenting the concept of opportunistic computing as well as some of the works that adopt this concept. In the next chapter, we continue the discussion about opportunistic computing by presenting the service-oriented computing approach in details.

Chapter 3

Service-oriented computing

Contents

3.1 Introduction

Service-oriented computing (SOC) is a distributed computing paradigm that enables the design of distributed system with the objective of providing loose coupling between components and achieving a low-cost development process. In SOC, resources and functionalities are abstracted and exposed as services to be discovered and invoked remotely. Services are autonomous, platform-independent and also provide late binding. These characteristics make SOC suitable for dynamic and rapid changing environments characterized by volatility and heterogeneity. Thus, SOC could be considered as a reasonable approach for implementing opportunistic computing.

In this chapter, we explore different notions related to SOC, mainly discovery, invocation and composition. We present some of the research works and industry solutions that implement the concept of SOC for different types of environments, mainly Mobile ad hoc Networks (MANETs), and most importantly opportunistic networks that constitute our focus in this dissertation.

The rest of the chapter in organized as follows. In Section 3.2, we introduce the notion of service discovery and we discuss the solutions related to it. In Section 3.3, we discuss some of the works related to service selection and invocation in opportunistic networks. In Section 3.4, we present the notion of service composition. Finally, in Section 3.5, we provide a discussion and we draw several conclusions.

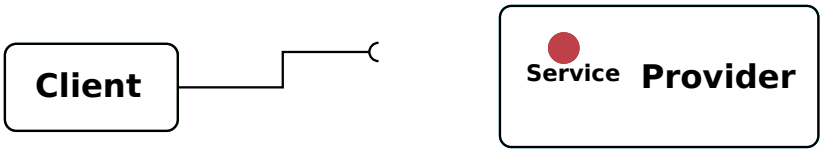


Figure 3.1 – Client and provider interaction.

3.2 Service discovery

In SOC, the elementary component is called "service". A Service is a software component that implements a set of functions. Services are hosted by devices called service providers and called upon by devices called service clients as shown in Figure 3.1. A service can be invoked to provide one of its functions using a public interface available to all potential clients. This interface is described using a document, called service description, written usually using a specific description language like WSDL (28). The service description details the functional properties of the service (functions, input and output parameters, datatypes, etc.) and also the non-functional properties (QoS, security, etc.).

So that the client can invoke one of the services hosted by the provider, the service description of that service should be made accessible to the client, in the first place. This is done by a process called service discovery. The goal of this process is to establish a loosely coupled relationship between the provider and the client. This will guarantee a small dependency, it will offer more flexibility, and it will allow both the client and the provider to evolve separately and asynchronously from each other.

Service discovery has been the topic for a lot of research works and several industry products that were developed in order to support applications relying on service-oriented computing (100). In these works, different types of networks have been considered so far, ranging from stable wired networks, where the network does not suffer from connection disruptions and user mobility, to wireless mobile networks, where nodes communicate directly with one another in an ad hoc mode without the need of a particular infrastructure. There are a lot of approaches, found through out the literature, describing how to implement a discovery process. Hereafter, we present some of them.

Directory-based *vs* directory-less architecture There are two types of architectures for service discovery: the *directory-based architecture* and the *directory-less architecture.*

The directory-based architecture relies on the existence of a directory that aggregates service descriptions. In this architecture as shown in Figure 3.2, service providers register the descriptions of services they host to the directory and service clients send requests to the directory in order to discover them. The directory will, then, provides the clients with the informations that are required to invoke service providers. This architecture can be implemented using a centralized approach, where the directory is hosted by a few fixed nodes in the network. A well-known example, that is centralized,

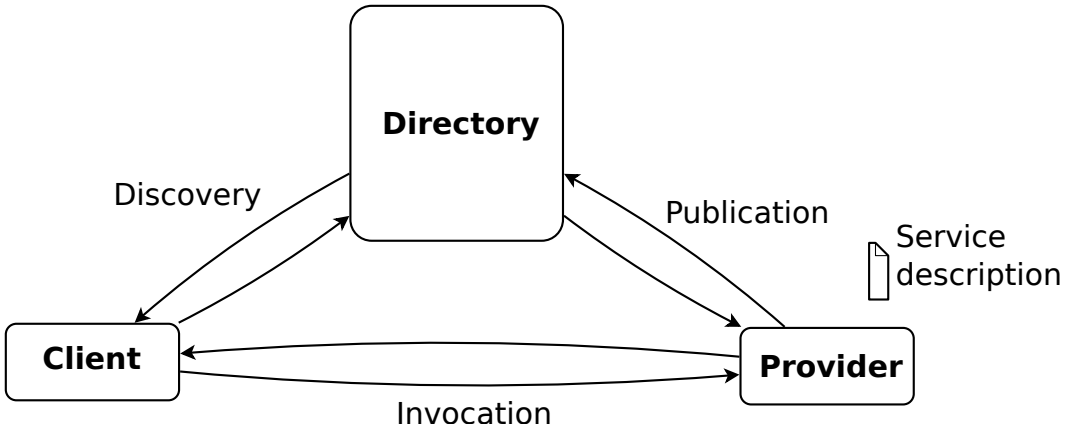


Figure 3.2 – Directory-based service discovery.

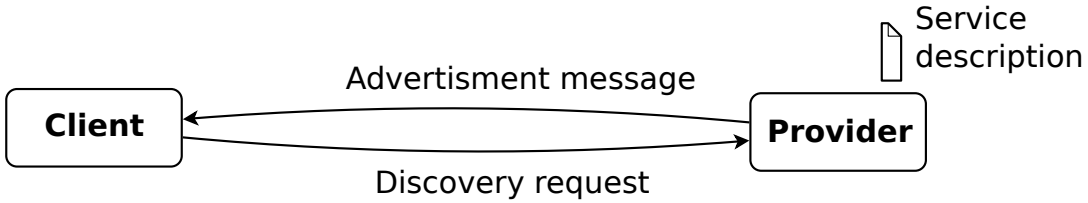


Figure 3.3 – Directory-less service discovery.

is UDDI (123). While this approach can be acceptable in a wired stable environment, it still can suffer from a single point of failure when only one node is used to host the directory. Besides, this approach is not scalable. To address these issues, a decentralized directory-based architecture can be implemented. The main challenge for this approach is to achieve a global discovery process, where any service client can discover any service provider. A well known example of a decentralized directory system is Jini (82), which is a service discovery architecture for Java-enabled devices. In Jini, some nodes play the role of lookup servers that act like directories. The downside of Jini is that there is no communication between lookup servers, which prevent global service discovery. Instead, it is the responsibility of the service provider to publish its services on all lookup servers in order to make these services discoverable in the entire network. The directory-based approach is, in general, not suitable for opportunistic environments, since there is no node, that can be considered stable and at all time reachable, to play the role of a directory due to mobility and to limited network and local resources.

As opposed to the *directory-based architecture*, in the *directory-less architecture*, there is no directory structure that plays the role of an intermediate between the service providers and service clients, as shown in Figure 3.3. Instead, service providers directly broadcast advertisement messages in the network to publish their services, and service clients listen to them. Clients can as well send a discovery request in order to find the services that answer their needs. One of the major challenges of such an approach is not

to overload the network by sending an excessive number of advertisement messages. Therefore, a service provider should determine the optimal periodicity of service advertisement to not overload the network, while maintaining an efficient advertisement process at the same time. DEAPspace (91) is a directory-less service discovery protocol based on a one-hop service advertisement broadcast to direct neighbours in a periodical manner. The advertisement contains a description of the local hosted services and the services that were discovered. DEAPspace relies on an exponential back-off mechanism that determines the periodicity of advertisement broadcasts based on the service provider priority and the changes that generally occur in the network. Campo et al. (22) propose another directory-less service discovery middleware for pervasive environments. Upon receiving a broadcasted discovery request, a given device checks in its local services and its cache. If it has a suitable response, this device proceeds by triggering an exponential back-off mechanism that favors availability time (the more available is the device, the shorter time it has to wait to respond to the discovery request). If the device detects a response for the same service discovery request while it is waiting for the back-off time to finish, it broadcasts the detected response rather than its own if the detected response comes from a provider with more availability time. Likewise, the middleware reduces the load on the network in term of exchanged messages, and favors providers that can be invoked for a long period of time thanks to their long availability time.

We can also combine both discussed architectures to create a hybrid one. In a hybrid architecture, providers and clients send their advertisement messages and discovery requests to a directory, if there is one available. Otherwise, they switch to a directory-less mode of operating. UPnP (31) is an example of a hybrid architecture. This protocol is intended toward discovering devices and the services they provide in small environments such as houses or offices. The basic entities in this protocol are control points that act like directories, and devices that provide services. When a device first appears in the network, it must multicast an advertisement of its services to the rest of the network. Similarly, when a control point joins the network, it must multicast a service discovery request to collect the descriptions of the available devices and services. In UPnP, the use of a control point is optional. Instead devices can multicast their advertisements directly to their clients. This protocol also faces important scalability issues due to the excessive use of multicasting.

Proactive *vs* reactive discovery We identify two principle modes of service discovery: *Proactive discovery mode* and *Reactive discovery mode.*

Proactive discovery mode consists of service providers or directories that advertise their services periodically while service client listening to these advertisements. This discovery mode should take into consideration not to overload the network by frequent advertisements and it has to limit the range of the propagation of the advertisement in order to maintain a decent quality of service provision. In this context, Li et al. (74) present an algorithm for proactive service discovery in pervasive environments based on context-awareness. In this paper, the context is represented using a multidimen-

sional space called *hyperspace analogue to context* (HAC), where each context property (location, temperature, pressure, etc..) is represented with a dimension and limited by a scope. The authors associate to each service an input context and an output context and to each user a "user context" and a "user preference". Upon an advertisement reception, the algorithm tries to match the user context with the service input context and the user preference with the service output context in order to determine the best services that suit the user's needs. The algorithm re-evaluates the services, if the user context or preferences change, or if some services update their context.

Reactive discovery mode consists of the service client soliciting either the directory or the service provider, by sending it a discovery request, in order to find a certain service. Similar to the proactive mode, issues related to propagation range should be taken into consideration in order to not to overload the network and to guarantee that only the most efficient services are discovered. Service Discovery Protocol (39) is a reactive protocol for Bluetooth-enabled devices. It is part of the Bluetooth official specification. The basic elements of the SDP architecture are SDP clients and SDP servers. A SDP client allows high level application to discover remote services by sending SDP requests to SDP servers. On the other hand, a SDP server allows high level applications to expose their services by storing descriptions in a form of a list of service records. SDP server then answers SDP requests by sending SDP responses based on its list of service records. It is important to mention that the SDP protocol does not provide any mechanism for service invocation. Moreover, a SDP client can not know if a SDP server has become available or if it has disappeared, which may affect negatively service invocations.

The Service Location Protocol (SLP) (99) is both proactive and reactive, and implements both the directory-based and the directory-less architecture. This protocol represents an IETF standard which was adopted in a lot of commercial solutions by several companies (IBM, Hewlett Packard etc.). SLP divides the networks into three categories: *Service Agent (SA),* which is responsible for advertising services to the rest of the network, *User agent (UA),* which carries out service discovery and *Directory Agent (DA),* which has the task of aggregating services descriptions. SLP has two modes of operation: *(i)* In the absence of a *DA, UAs* multicast their requests to the network, while *SAs* listen and answer with advertisements when they receive an *UA* request. Whereas *(ii)* in the presence of a *DA, SAs* send advertisements and *UAs* send requests to it. SLP provides also two methods of discovery: active and passive. With the active mode, *UAs* multicast their SLP requests to the network, and with the passive mode, *DAs* advertise their services periodically. In SLP, services are advertised using service URL and service Template (51), which can be described as a set of key-value pairs.

3.2.1 Service discovery in MANET

Kozat et al. (66) propose a distributed directory-based solution for service discovery. This solution is based on the creation of a backbone in a form of a mesh structure of the most dominant nodes in the network. This solution operates in two phases: first, the *backbone management* phase, that consists of forming the backbone by selecting the most

dominant nodes, using a threshold on a parameter called *NLFF (Normalized Link Failure Frequency).* NLFF represents the number of link loss in a fixed time window. If a given node has *an NLFF* that does not exceed the fixed threshold, this node will be selected to be part of the backbone. The second phase is called *Distributed Service Discovery,* where the nodes forming the backbone start to manage the service discovery process in the network.

Sailhan et al. (105) propose a similar approach for service discovery in MANETs using backbones. However unlike in (66), the construction of backbone relies on a directory election process to select nodes, that have the most resources and the most suitable context (number of nodes reachable at 1 hop, reachable other directories, stability level etc.), to play the role of directories. Directories then frequently exchange profiles constructed using bloom filters (18). This helps reduce the size of messages and the generated traffic, unlike in (66), where multicasting is heavily used when a discovery request is received in order for the backbone nodes to communicate with each other and be able to deliver a response.

Rendezvous Regions, presented in (110), proposes a distributed directory-based service discovery solution that relies on distributed hash tables based on location. In fact, *Rendezvous Regions* divides the network into several geographical regions. Each region is responsible for holding a set of keys. A key can represent either data, resource or service, and it is mapped to a region using a hash-table-like mapping function known to the entire network. This mapping function is used by nodes to either insert or look up for a key. Then, a packet, that contains the region identifier, will be generated in order to send the request (insertion/lookup) to the targeted region, where the packet will be processed by the servers managing this region. *Rendezvous Regions* also provides a simple election mechanism in order to elect a fixed number of servers per region.

Klein et al. (64) propose a service discovery framework based on clusters called service rings. Indeed, devices, initially, organize themselves in a form of closed rings called level 0 rings. Each ring has a service access point (SAP) that plays the role of a directory for the rest of the nodes in that ring. SAPs from level 0 also form rings called level 1 rings. The nodes from a level 1 ring choose a level 1 SAP that act as a directory for them. This process can be repeated until we reach level n rings, and thus, this framework establishes an architecture based on hierarchy. A given node has 4 main operations: join or leave a ring, advertise its services or search for a service.

SANDMAN, described in (107), is also a distributed directory-based service discovery frameworks that relies on clusters. The main goal of SANDMAN is to be energy efficient and to reduce latency in the discovery process. In SANDMAN, clusters are formed by *clustered nodes* (CN), that play the role of service providers, and by *clusterheads* (CH), that play the role of service directories to which CNs advertise their services, and from which clients retrieve descriptions of services they want to exploit. CNs periodically enter into a sleeping phase in order to save energy. Whereas, CHs stay always awake in order to avoid latency in discovery time. SANDMAN discovery approach unavoidably introduces latency in the service invocation process, especially, when at the time of invocation, the targeted provider is in sleep mode. SANDMAN also does not

provide any mechanism for cluster formation and leave the problem open for multiple approaches.

Forming structures, such as backbones, clusters or regions as the aforementioned works suggest, would be very difficult or even impossible to achieve in an opportunistic network. These kind of structures need the existence of a set of nodes that are stable and have low mobility, so they (the structures) can stay functional for a long enough time and also recover when needed. These requirements are obviously not available in opportunistic networks. Consequently, if these structures succeed to form in an opportunistic environment, they will have a very short life. Moreover, their recovery process will not be fast enough to react to the rapidly changing topology of the network.

Konark (53) is a directory-less service discovery protocol. Konark offers both proactive and reactive service discovery mode. In Konark, each node in the network, that runs a SDP (Service Discovery and Delivery Protocol) manager, plays the role of both the service client and the service provider. Each node in the network also implements a local service registry that is structured in a form of a tree to help classify services. Konark also proposes a service description language based on XML and considered by the authors as a simplification of the WSDL language, which is acceptable, in this context, due to the simplicity of the services.

The authors of (27) present a protocol for service discovery in MANETs called GSD (Group-based Service Discovery). GSD is also a directory-less discovery protocol that implements both the proactive and the reactive discovery mode. GSD aims to reduce the network load by classifying services in a hierarchy of service groups. Each service provider periodically emits an advertisement that contains a description of its local services and the groups they belong to, as well as the list of groups of remote services that the service provider knows about. Likewise, when a service client needs to discover a certain service, it will only send the discovery request to nodes that either heard about the group of the targeted service or actually host services that belong to that group. Likewise, this protocol avoids broadcasting and flooding the network. In GSD, service descriptions as well as service groups are implemented using DAML (1).

Nevertheless, these directory-less solutions still assume the existence of a end-to-end path between each pair of nodes. Therefore in order for them to function properly in opportunistic environments, a work of adaptation should be made.

In (86), Mrissa et al. introduce an approach for service discovery, using semantics, dedicated to the Web of Things (50). Each connected thing is extended by an avatar to expose its functionalities (services). In this paper, the authors define an ontology in order to be able to discover, compose and represent low-level capabilities in a form of functionalities available and exploitable on the Web.

3.2.2 Service discovery in opportunistic networks

Service discovery, in opportunistic environments, has been addressed in several previous works. In (79), the authors propose a set of protocols called TAO that operate over an intermittently connected hybrid network which is composed by simple nodes

and infostations. Infostations usually form clusters and are the main service providers. The service discovery is organized by TAO-DIS protocol. TAO-DIS piggybacks service guides(SG), which are the sets of service descriptors, in the beacon messages to reduce and optimize network traffic.

OLFServ presented in (69; 70) is a protocol for service discovery based on location-awareness and dedicated to opportunistic networking. Infostations advertise their services using a geographically controlled propagation. They include in every advertisement their location and the geographical area where they prefer to be discovered or invoked. OLFServ is equipped with self-pruning heuristics that determine if nodes should participate in the service delivery or not. These heuristics are the ones that prevent messages from being propagated outside their geographical areas.

Nevertheless, both OLFServ and TAO make the assumption that only fixed infostations provide and advertise services, and that mobile devices can only discover and consume them.

Other protocols adopt a social-aware strategy like (21; 81; 92). In (81), the authors model each node interests using a vector, in an m-dimensional space, noted IP (interest profile). Each generated message has its own relevance vector (R) in the same space. Every time a pair-wise contact takes place, both nodes exchange their IP vectors, and search in their buffers for the messages that represent a similarity with the received IP. The similarity is calculated using the cosine similarity metric.

Groba et al. (48) present a discovery and composition algorithm for opportunistic environments. The discovery process is based on a directory-less reactive approach. The downside of reactive discovery is that it introduces delays in the execution of invocation and composition requests. Indeed, everytime there is a service to invoke, the service client does not start the invocation process immediately. Instead, a discovery request is emitted to search for providers. Once the providers respond to this request, the client can then start the invocation process. Consequently, the reactive approach could undermine the performance of services that should provide a quick response time such as a service for finding parking places.

The authors of (65) define two location-based service discovery (LADS) and selection (LASS) protocols for wireless networks. Both these protocols rely on the maximum speed of the provider v_{max} and the maximum response time t_{max}. The discovery process is done reactively using a discovery message mechanism. In (121) , Wang et al. present a service discovery protocol for delay tolerant networks. Its approach is diretory-less and proactive. Every period of time an advertisement message is emitted. The protocol uses a Bloom filter in order to fix the size of service description, and the service query is done by emitting a limited number L of invocation messages.

Pitkanen et al. (101) introduce a service platform called SCAMPI (Service platform for social-aware mobile and pervasive computing). SCAMPI architecture provides distributed task executions for opportunistic environments by abstracting resources, scattered across the network, as services. SCAMPI implements its service-oriented model based on social and context awareness of the behaviour of users, present in the network, to achieve an efficient opportunistic interaction between sensors, resources stationed in

	reactive	proactive	directory-based	directory-less
TAO (79)		x		x
OLFServ (69; 70)		x		x
Groba et al. (48)	x			x
LADS (65)	x			x
Mahéo et al. (77)		x		x
Kozat et al. (66)	x		x	
Sailhan et al. (105)	x		x	
Rendezvous Regions (110)	x		x	
Klein et al. (64)	x		x	
SANDMAN (107)	x		x	
Konark (53)	x	x		x
GSD (27)	x	x		x

Figure 3.4 – Comparison between discovery algorithms

the network, and devices carried by individuals.

The middleware, proposed by Mahéo et al. (77), uses a content-based communication approach to perform the service discovery process. This middleware adopts the publish/subscribe paradigm in order to disseminate advertisement messages across the network. Each node maintains an interest profile and periodically advertises a catalog of the message headers it stores in its cache. If a node finds message headers in the received catalogs that match its profile, it will request a copy of the corresponding messages. This approach allows to avoid flooding the network by only forwarding messages to nodes interested by those ones. The authors use this approach to carry out service discovery and advertisement in a proactive and directory-less fashion. Indeed, each node creates service patterns in a form of a profile interest, and every time it receives a catalog that contains service descriptions, a matching process is executed in order to identify the most relevant remote services.

3.3 Service selection and invocation

Service selection is the process in which a service client has to choose a single or a group of providers that are considered the most suitable to invoke. Selection is then followed by the invocation process. In the invocation phase, the service client starts actually to communicate with the service provider by sending an invocation request to it. Upon reception, the provider uses the informations from the request to perform the service execution. Finally, the provider returns the result to the service client. In the invocation phase, a service client invokes one or even a group of providers by explicitly indicating their addresses following a destination-based communication. Likewise, the client can send its request using a unicast, a multicast, or an anycast mode. Therefore, we can rely on the forwarding protocols for opportunistic networks, presented in Chapter 2, to

deliver data between the service client and the service providers. Several works studied specifically the topic of service selection and invocation in opportunistic networks.

Apart from its discovery approach presented above, OLFServ (69; 70) also provides an invocation algorithm. After discovering infostations, that provide services, a given client can thus invoke a needed service by embedding its own location, speed and direction in the service request message so the infostation, providing this service, can predict where to send the reply based on these informations. Both the invocation request and the invocation reply are forwarded by the same self-pruning heuristics used in the discovery process.

TAO-INV protocol (79), proposed by Makke et al. (part of TAO protocols), assesses the ability of intermediate nodes to deliver messages from the client to the provider. It evaluates the immediate neighbours of each node and classifies them as good or bad intermediate nodes based on the contact date with the infostation to invoke. The recent is the date, the better intermediate node is the neighbour.

Le Sommer et al. (71) introduce a proxy style algorithm for invocation. It enables nodes to invoke services without the need of reaching the provider itself. This is made possible by accessing another node that previously requested the same service from the original provider. This very node is called proxy for its role of caching the results and redistributing them on demand. The proxy approach is useful when the result of a service invocation does not change for a certain amount of time, such as with weather forecasting services. It is important to mention that not all services are proxible. That is why the article authors classify services into categories.

In addition to its discovery approach, the middleware, proposed by Mahéo et al. (77) also proposes a service invocation mechanism based on content-based communication. When a node wants to invoke a given service, it does not send the invocation request to a particular provider. Instead, it formalizes a reduced version of the service description and publishes it to the network. Providers, that have subscribed to this reduced description, will execute the service and return the results in messages that contain description attributes. Clients, that have subscribed to these attributes, will accept the results. While this approach can improve the invocation success ratio by soliciting many providers, it is nevertheless expensive in terms of resource consumption.

3.4 Service composition

One of the most important principles of service-oriented computing, is to encourage the reuse of services. Consequently, each and every service should be designed as an easy to reuse software component. This makes the service-oriented approach suitable for pervasive environments.

One aspect of service reuse can be found in the concept of service composition. It consists of aggregating services in order to automate some tasks or business processes, or to create new composite services, to provide new functionalities, that otherwise atomic simple services can not provide by themselves. A composition consists of chaining services one after another by matching service interfaces either syntactically

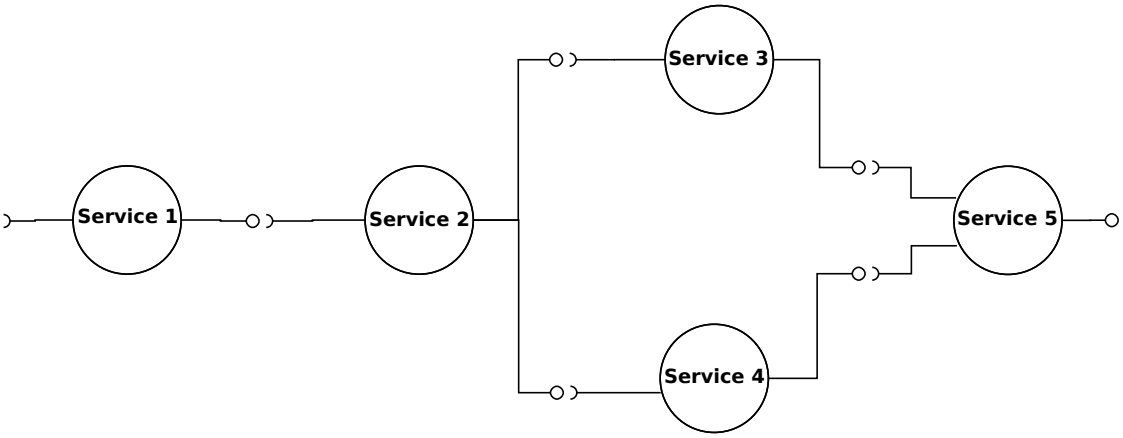


Figure 3.5 – Service composition.

or semantically using service graphs or specific languages like BPEL4WS (6) or OWL-S (80). Likewise, the outputs of one service are piped into the inputs of the next service in the composition, after eventually filtering content and changing formats, as shown in Figure 3.5.

Throughout the literature, we find many approaches to carry out a service composition. For example, a composition can be sequential or it may contain several parallel sub-processes. It can be dynamic or static, or it can be automated or manual. Furthermore, it can be orchestrated by a device responsible for supervising the execution of this composition, or it can be executed in a collaborative manner based on a choreographic approach. Hereafter, we discuss these approaches, as well as several works that adopt them.

Manual *vs* automatic composition Manual composition consists of putting the burden of generating the composition description on the user. This can be done using graphic tools or by literally coding the description using a specific language suitable for this kind of composition like BPEL4WS. The drawback of manual composition is that it requires knowledge and expertise from users in building, coding and maintaining a composition description. Whereas, the automatic composition intends to relieve users from the effort of taking care of the relatively complicated details of describing a composition. Automatic composition is mainly based on the usage of semantics and ontologies. In this context, we can mention the work done by Fileto et al. (45). The authors propose a framework called POESIA that helps developing applications on the basis of ontologies. POESIA is focused on domain-specific ontolgies and workflows, in order to carry out service compositions. Other efforts have been done in this direction. The authors of (85) present a framework called IRS-II that provides an infrastructure for semantic Web services. On the other hand, Tosic et al. (118) discuss the requirements that an ontology should answer to, especially in term of QoS, in order to optimize a dynamic and automated service composition.

A composition description, whether generated manually or automatically, can be either static or subject to changes dynamically throughout the invocation of the composite service.

Static *vs* dynamic composition Static composition consists of defining a composition description that stays unchangeable throughout the invocation of the composite service. The description is often defined off-line at design time before it is deployed and ready to be executed. Basically services are chosen, interfaces are chained and service providers are selected before the composite service is ready to be deployed on-line for execution. Several composition engines provides static composition like Microsoft Biztalk (41) and BEA WebLogic. While this static approach seems to be reliable, especially when dealing with a long composition that requires a complicated workflow, it can not cope with an environment where services could disappear due to issues like connection disruption or unpredicted technical problems affecting service providers. On the other hand, dynamic composition allows the composition description to be update at runtime. Thus, it provides more flexibility and it has more ability to adapt to services disappearing and other new services appearing. As examples of dynamic composition engines on the market, we can cite eFlow (25) and StarWSCoP (115). Moreover, the authors of (13) detail a solution to make service composition self-adaptive. Indeed, each composition is associated with a set of requirements called adaptive goals. The authors defines two types of goals: fuzzy goals where the satisfaction value is within the interval $[0..1]$, and binary goals that have a satisfaction value of either 0 or 1. Each goal has a set of membership functions that determine the satisfaction value. In the case of a satisfaction value not being met, adaptation actions can be triggered. Adaptation actions act on the set of goals of the composition, either by adding or removing them, by modifying them, by changing service providers or even by replacing several services in the composition.

Orchestration *vs* Choreography According to Peltz et al. (95), orchestration consists of the business process, that describes the interaction between services and the order in which these services should be invoked, from the perspective of an orchestrator that is responsible for executing the business process. In orchestration, services have no knowledge of whether or not they are being involved in a composition. Whereas, the choreography describes the messages exchanged between the services involved in the composition. It represents a more collaborative approach where services are aware of their involvement in the composition. Multiple works have addressed this topic. For example, we can mention BPEL4WS that provides both executable processes that model orchestration, and abstract processes that model choreography. Moreover, the WSCI (7) (Web Service Choreography Interface) specification provides message exchange and choreography mechanisms based on WSDL (28) (Web Services Description Language). In addition to that, the BPMI (Business Process Management Initiative) proposes an XML-based language called BPML (Business Process Management Languages) that models business process for orchestration.

3.4.1 Infrastructure-based and conventional composition

Conventional service composition solutions require stable and efficient means of communication that do not suffer from connection disruption or mobility. Almost all of them are based on the client/server architecture and therefore require the existence of a single or a group of nodes that have relatively large resources, and that are always accessible and available to be solicited by the network to play the role of servers. This renders these solutions inadequate for MANETs and opportunistic networks and need at least to be adapted to these environments by taking mobility and disconnection into consideration.

There have been a lot of efforts and solutions (4; 46; 73; 102) proposed to tackle the topic of service composition in an infrastructure-based and stable environments. In this part, we mainly focus on RESTful (44) services. RESTful services are lightweight, stateless and cacheable. They provide a uniform interface, and usually exploit HTTP (43) built-in methods (GET, POST, PUT, DELETE).

Several efforts focused on adapting legacy SOAP-based composition solutions to integrate RESTful services. Pautasso et al. (94) propose an extension to BPEL in order to integrate both SOAP and RESTful services in the same composition.

De Giorgio et al. (35) present a solution to re-describe both SOAP and RESTful services uniformly, in a composition, using a semantic annotation language called MicroWSMO. The authors then use LPML (Lightweight Process Modeling Language) to describe service compositions.

REST2SOAP framework (97) offers to wrap RESTful services in the form of SOAP services to be able to integrate them in a BPEL-based process. Other research works focus on devising solutions uniquely for RESTful service composition. Pautasso et al. (93) propose a composition language called JOpera. This language provides features like dynamic binding and dynamic typing, and respects the principle of a uniform interface. Li et al (75) present a lightweight solution for chaining services in a Unix pipeline fashion called “Hyperlink pipeline”.

The authors of (68) define a semantic description language called SEREDASj. This language permits to semantically describe RESTful services using JSON objects, as well as to automatically generate a human-readable documentation. SEREDASj allows also data integration through semantically describing them, which makes it possible to automatically generate composition descriptions.

Mrissa et al. (16) propose a RESTful service composition approach based on linked data principle. Each service contains a link to its descriptor in the headers. A descriptor has three main parts: an interaction model that describes operations provided by the service, links to other services related to the described service, and a link to an universal descriptor that contains meta-data that describes how to interpret a service descriptor. The paper does not detail any sort of semantics that can be used to implement this approach and leaves the problem open to be addressed in other future works.

Silva et al. (47) present a framework for automatic service composition at runtime called DynamiCoS. This framework is independent from description languages, and requires a specific language interpreter for each supported language. Likewise, Dy-

namiCoS can be more generic, and supports service publications in multiple languages. DynamiCoS organizes services in a matrix according to their inputs and outputs. For DynamiCoS, a composition request is expressed in the form of the set of initial inputs and final outputs required by the user. Upon receiving a composition request, DynamiCoS performs a backward graph building based on the matrix of published services, in order to determine all possible compositions that answer the user requirements.

3.4.2 Composition in pervasive and wireless environments

Service composition in dynamic pervasive environments, like mobile ad hoc networks, has also been considered (57) in the past years. However, the proposed solutions rarely tolerate the disruptions that can unpredictably occur between devices, and that can introduce additional delays and failures in the execution of invocation and composition requests.

Nevertheless, several research works attempt to address unpredictability, long delays and failures. For example, Deng et al. (37) propose a composition approach based on geographical proximity. The approach defines location-based communities called Mobile Service Sharing Communities (MSSCs), where users can join, leave, or move within a given community. The article also proposes a mobility model, that illustrates human behaviour within those communities, called CRWP. Based on these elements, the service composition is modeled as an optimization problem, and solved using the Krill-Herd algorithm.

In the same context, Capra et al. (36) present a composition framework for mobile environments that intends to increase composition reliability by considering mobility patterns of users and their colocation duration. The authors consider that the probability of accessing a service is related to the colocation duration. Wang et al. (120) also present a solution based on mobility prediction in order to improve dependability of service composition in wireless mobile ad hoc networks.

Other works address the problem by defining and maintaining a distributed graph of services, listing the providers that are likely to be found and called successively when a composite service is requested. The authors of (3) follow this idea in their solution for service composition in mobile ad hoc networks. This solution builds its service graph using a neighbour discovery mechanism based on a periodic emission of beacons. Service discovery is carried out by propagating service parameters using update messages, that are exchanged upon neighbour detection, service appearance or disappearance. Services are also described semantically, and the descriptions are contained within local service directories.

Zhou et al. (125) use a service graph for their energy-efficient composition framework dedicated to wireless sensor networks. This framework forms and maintains the service graph that connects services to each other on the basis of the similarities between their parameters and operations. Upon receiving a composition request that specifies the desired inputs and outputs, the framework will identify the potential sequential compositions that can fulfill the request based on the service graph. Then, it

tries to recommend the most optimal composition by solving an optimization problem, which has constraints such as energy efficiency, and spacial and temporal constraints, by using the particle swarm algorithm (PSO).

Relying on such a graph does not however guarantee that an invocation of a composite service will be completed successfully. Indeed, in a dynamic environment, services, providers and intermediate devices can appear and disappear at any time. Thus, it makes it difficult to perform an end-to-end invocation of a composite service. Furthermore, if a service provider is not available, the partially executed composition remains in an inconsistent state. Recovery strategies are therefore indispensable in such dynamic networks. SeSCo (63) attempts to find a new provider for the current service to invoke in a composition request every time a previously selected service provider become unreachable due to users' mobility. Likewise, SeSCo can perform partial compositions and then complete them whenever new service providers become available. SeSco is also based on a hierarchical architecture. It provides a composition mechanism capable of supporting locality, quality of services, semantics and mobility. Hierarchy in SeSCo is based on the processing power, memory and bandwidth capacities of nodes.

Chakraborty et al. (26) present a decentralized broker-based system for service composition in ad hoc environments. This system implements a fault tolerant checkpoint mechanism to allow resuming partial compositions interrupted due to users' mobility. The basic underlying idea is that the requesting node selects a broker and then delegates to it the orchestration of the service composition. The requester (RS) starts by looking in its vicinity for a suitable broker. It broadcasts a solicitation message that contains a list of the atomic services that form the composite service. Each candidate computes a potential value. This potential value is calculated based on the local resources and the number of cached service advertisements. The local resources consists of: the number of matching local atomic services, the battery life and the current number of requests that the candidate has to execute. The potential values will be sent back to the RS. It will then choose the broker that has the maximum value.

Jiang et al. (60) present a framework for service composition dedicated mobile ad hoc networks. Its goal is to guarantee a minimum disruption in the execution of the composition, as well as providing a recovery mechanism in case it is needed. According to the authors, the framework consists of two main tiers: *service routing*, which is responsible for selecting services involved in the composite service, and *network routing*, which is responsible for choosing the best path that connects these services. The authors also formalize the composition problem in a form of a dynamic programming problem, and analyze it to find a heuristic algorithm capable of approximating the best composition solution.

Nevertheless, these composition protocols suppose the existence of an end-to-end path between each and every pairs of nodes, and that ad hoc networks are dynamically routed using protocols like AODV (98), DSR (61) or DSDV (89). These protocols have been proven inefficient especially when the network suffers frequent and unpredictable connection interruptions, and when the network is fragmented into different communication islands. In reality most of the wireless networks supporting pervasive

environments are usually intermittently connected, which makes opportunistic and delay tolerant techniques the most suitable to improve communication in these types of networks. Even though, these techniques remedy to frequent and unpredictable connection disruptions, they can introduce considerable delays. Thus, composition and discovery techniques built upon opportunistic networks should take that into consideration and try to reduce communication time as much as possible.

Groba et al. (48) try to reduce the execution time of a service composition in opportunistic networks by integrating the execution phase into the composition phase, and allowing to perform a composition partially. The advantage of this solution is that it addresses each selected service provider only once and requires less data exchanges than a traditional consecutive approach that runs in two phases: a binding phase to select the providers, then a phase to invoke them. The downside is that the authors rely on a reactive discovery process, as discussed in Subsection 3.2.2, which introduces delays and consequently slows down the composition execution.

In (49), the authors propose an extension of the solution described in (48). This extension makes it possible to invoke several providers that offer the same service in parallel, thus creating several branches in the service composition graph. The branches can be split and merged, and the results of the composition can be combined using semantic features. Although leveraging the redundancy of service providers can be a solution to improve the success ratio and the execution time of a composition, it must be used with moderation. Indeed, this solution multiplies the number of copies of the service requests roaming in the network, requires several providers to perform a same computation in an undifferentiated and non optimal way, and implies more data exchanges between devices, thus abusively consuming the power budget of the devices and risks to overload the network.

In this same context, the solution proposed by Sadiq et al. (104) tries to reduce composition delays by making use of two metrics: *shortest temporal distance,* which is the minimum time needed to send data from one node to another one, and *service load,* which reflects the workload of a given service. The composition algorithm presented in the article uses these metrics in order to select and invoke service. This algorithm is designed to use a choreography-based strategy, where the current service provider is in charge of executing the current service in the composition and choosing the next service provider to invoke in order to execute the next service. These steps will be repeated until all services in the composition request are executed, and then the final result will be returned back to the composition initiator. The authors of this paper do not investigate the orchestration-based strategy, and do not compare it to the choreography-based strategy in the context of opportunistic networks. Moreover, the authors do not specify how informations about remote services are collected, and what service discovery approach they are using.

3.5 Discussion and Conclusion

In this chapter, we have introduced the service-oriented computing paradigm which represent our chosen approach to implement opportunistic computing. We have presented the general notions related to SOC, mainly service discovery, service selection, service invocation and service composition. We also surveyed some of the research works and industrial solutions that implement service computing or several aspect of it. This chapter allowed us to identify the techniques that can be adopted in our contributions as well as those that should be avoided. We summarize our analysis in the form of requirements that our solutions should meet.

Service discovery requirements Most of the service discovery systems rely on a directory-based approach where the directory can be either centralized or distributed. Service can be advertised proactively, by sending advertisement messages every period of time or they can be discovered on-demand reactively by querying registries or service providers directly using a discovery request. Reactive discovery, as in (48), guarantees an access to up-to-date informations about remote services. However, it introduces a relatively important delay when it comes to the execution of invocation or composition requests. The proactive discovery, nonetheless, does not have this limitation which reduce the invocation or the composition time drastically. Nevertheless, this approach introduces more failure in the composition or the invocation process, in case service informations are not regularly updated enough. Besides, from an opportunistic networking point of view, accessing and maintaining a directory, as in (66; 110), is a substantially difficult task due to frequent connection disruptions and mobility as underlined in many discovery protocols (55). Therefore most solution dedicated to such networks relies on a directory-less approach coupled with a proactive advertisement process. This will introduce a tremendous overhead because service providers should advertise their services regularly and periodically so the service client can have a relatively reliable and up-to-date information about the available remote services. One way to reduce this important overhead, is to piggypack service advertisement in beacon messages used by devices to announce presence to other surrounding nodes. Another solution is to find the optimal period of advertisement that allows the service informations to be up-to-date and avoid overloading the network.

Service selection requirements Service selection is an important step in the invocation of simple or composite services. Indeed, it aims at selecting, among the discovered service providers, the most reliable and efficient one to answer the request. In the type of networks we consider, carefully selecting intermediate nodes is an important phase for the execution of an invocation or a composition request. In fact, many solutions (84) have been devised to tackle the topic of routing. Usually, service selection is based on non-functional properties and informations (location, transmission delay, inter-contact, history, social informations, etc.), provided by service providers, that give an insight about the accessibility of these devices, as shown in (69; 70; 78). To be able to provide

these informations, cross-layering techniques must be implemented in the system in order to have a tight collaboration between the networking and the application levels.

Service composition requirements A service composition can be performed using either orchestration or choreography. It would be interesting to compare these two strategies in an opportunistic environment, given the fact that the orchestration was not taken into consideration by the closest works (49; 104) to this thesis. It would be also interesting to combine them (the two strategies) in a complementary manner in order to execute a composition request. In fact, depending on the network topology and how the services are distributed across the network, it would be better to choose orchestration over choreography or vice-versa. To achieve that, service providers enrolled in the invocation of the composite service should be able to dynamically switch between these two strategies in order to increase the likelihood of composition success in the shortest time possible. For that, the $i-th$ provider in the composition should be able to compare its capacity to invoke the $i+2$-th service provider with that of the $i+1$-th so as to decide which strategy to adopt.

In the next chapter, we present our approach for service discovery and composition based on the observations we made in this chapter.

Part II

Contributions

Chapter 4

Service discovery and composition system

Contents

4.1 Introduction

In this chapter, we present our service discovery and composition system, that complies with the requirements we determined in the end of the Chapter 3. We begin by detailing our service discovery approach, as well as two implementations of a utility function used in the selection process and also in rating discovered service providers. One implementation is location-based that relies on location and on average distance. The other one is time-based that relies on transmission delay and on inter-advertisement reception time (the difference between the reception times of two successive advertisements from a given provider). The discovery approach, we propose, is directory-less where each node holds its own local service registry rather than having special nodes that explicitly play the role of directories. This approach is also proactive where nodes periodically advertise there local services to the rest of the network. We also present our service composition approach that supports both the orchestration-based strategy and the choreography-based strategy. We only consider sequential compositions in this thesis.

The reminder of this chapter in organized as follows. In Section 4.2, we present our discovery approach and both implementations of our utility function. In Section 4.3, we explain both our composition strategies. We then finish the chapter by drawing conclusions (Section 4.4).

4.2 Service discovery and utility functions

4.2.1 Discovery

As previously mentioned in the introduction, our approach is directory-less (relies on local service registries rather than a dedicated directory) and proactive. Unlike in (48), we choose a proactive approach to eliminate the waiting delay between sending a discovery request to find service providers, and sending the invocation or the composition request to one or to some of the providers that answered the discovery request. Moreover, and unlike these works (70; 78) do, our discovery approach does not suppose that services are exclusively provided by infostations, that have relatively significant resources, and consumed by mobile nodes, that have limited resources. In our approach, each node can be both a client and a provider.

Every period of time T_{adv}, each service provider emits an advertisement message M_{adv}. M_{adv} is formally defined by the quadruplet $\{D, H, P, T\}$, where D is the description of the local services of the provider P, H is the number of hops M_{adv} is allowed to make. H decreases by -1 every time M_{adv} is received by a node. Initially H is initialized to H_{max}, the maximum number of hops that M_{adv}can make before it is deleted. H_{max} aims at limiting the propagation of advertisement messages. Doing so, we prevent advertisement messages from reaching far away service clients. Consequently, we shorten the traveling distances of invocation and composition requests, which help increase the chances of completing invocation and composition requests successfully and in a reasonable amount of time. Finally T is the emission time of the message M_{adv}.

Each node implements and maintains a local service registry that contains information about both local (provided by the node itself) and remote service instances recently discovered. We formally define the local service registry SR by the set $\{L, R\}$, where L is the set of the descriptions of local service instances and R is the set of the descriptions of remote service instances. L and R are formed by 6-uplets defined by $\{P, S, D, H', U, \mu(P)\}$, where P is the service provider, S is the service name, D is the description of the service, H' is number of hops to the provider P. H' can be calculated by the formula $H' = H_{max} - H$, U is the latest update time for the 6-uplet which corresponds to the reception time of the last $M_{adv}$from the provider P. $\mu(P)$ is the result returned by the utility function.

The subset of service instances in the service registry that belongs to the provider P is noted $SR[P]$. Every time an M_{adv} from P is received the subset $SR[P]$ gets updated. If $SR[P]$ stays with no update for a period of time greater than $T_{inactive}$ (remote service entry inactivity time threshold), service instances provided by P will be considered out of reach and will be excluded from any composition or invocation process.

4.2.2 Utility function

In this dissertation, a utility function has the role of rating remote services and selecting them for invocations of simple and composite services. To achieve its goal, a utility function should be able to collect and exploit the informations coming from the network. This requires the usage of cross-layering techniques. Our utility function follows two criteria of quality of service: the invocation success ratio and the invocation time. Consequently, service providers are selected based on an estimation of the time needed to reach them and on an estimation of the ratio of a successful transmission. While close works rely on estimations of time delay (temporal distance in the case of Sadiq et al. (104)), they do not take into consideration the success ratio of a composition or an invocation request. Hereafter, we present our provider selection method using two implementations of our utility function.

Service provider selection

Service providers are selected by the selection function φ, which is defined as follows:

$$\begin{cases} \varphi(S_i) = P_j, \mu(p_j) = max(\mu(P_l)), P_l \in SR[S_i] \\ \mu(x) = \frac{\alpha}{t(x)} + (1-\alpha) \times s(x) \end{cases}$$

In this formula, P_j is the provider having the highest value calculated by the function μ among the providers that offer service S_i (identified by $SR[S_i]$ in the above formula, where SR stands for the service registry). Function μ computes a value based on the estimation of the time needed to reach a provider and on the estimation to invoke that provider successfully. These estimations are respectively noted $t(x)$ and $s(x)$ for a provider x. Parameter α makes it possible to promote one quality of service parameter to the detriment of the other. $t(x)$ and $s(x)$ are computed either based on temporal or geographical information. Hereafter, we show how these values are calculated. Moreover, the selection function φ only chooses a unique provider to invoke, unlike in (77) where selection and invocation rely on a content-based scheme that invoke multiple providers. Likewise, we avoid overloading the network and we reduce resource consumption.

The function μ is used by our system to decide if a provider P and his remote services noted $SR[P]$ should be put in the service registry or removed from it. Indeed, in the service registry, we only take into consideration service instances that their providers are deemed relevant. Thus, if the computed value of μ is greater than a certain threshold μ_{min}, the entry is added into the registry (or only updated if it already exists). Whereas, if this value is less than the threshold, the entry is removed from the list.

Time-based implementation of the utility function

We estimate the time needed to send an invocation or composition request to a provider on the basis of the average of the transmission delays of the service advertisements that the provider periodically broadcast. Whereas, the success ratio is estimated by the number of the advertisements received by a local host among those emitted by the service provider. To calculate this ratio, we assume that all devices send service advertisements with the same period of time (noted δ in the formulas hereafter). Likewise, to estimate the number of advertisements, emitted by the provider, between two successive advertisement receptions from this one, we divide the corresponding inter-advertisement reception time (the time elapsed between these two receptions) by the period δ. The time-based utility function we have defined uses a sliding window of k values (i.e., we only consider the last k advertisements that are received by the local host). Therefore, we define the estimated time t and the success ratio s for provider P as follows:

$$t(P) = \frac{\sum_{i=1}^{k}(\beta_i^P - \alpha_i^P)}{k}$$

$$s(P) = \frac{k}{1 + \lfloor \sum_{i=2}^{k}((\beta_i^P - \beta_{i-1}^P)/\delta) \rfloor} = \frac{k}{1 + \lfloor ((\beta_k^P - \beta_1^P)/\delta) \rfloor}$$

where α_i^p and β_i^p are respectively the emission time and the reception time of i-*th* advertisement received from provider P. By adopting an average of the transmission delays instead of considering only the last one, we promote the most frequently reached providers with a minimum of delay, either directly or via intermediate devices, instead of providers that are met in fleeting way.

Location-based implementation of the utility function

In each advertisement message emitted, our service discovery and composition system includes the location coordinates of the local host. Based on these pieces of informations, receiving devices can compare their own location with that of the providers. Similarly to the time-based utility function implementation, the location based utility function operates on a sliding window that only takes into consideration the last k advertisements received from a given provider. This utility function calculates the estimation of the transmission delay on the basis of the average distance between the local host and a given provider. By relying on the average distance between the local host and a given provider instead of considering only the last advertisement, we favor providers that are the closest to the local host during a given period of time instead of providers that are only available for a brief moment. Therefore, we increase the probability of invoking a service provider either directly or via intermediate nodes. The estimation of the time needed to reach a service provider P is equal to the reception delay of the advertisement i, such that the distance d_i^P between the local host and the

provider is the closest to the average of the distances traveled by the k advertisements received by the local host:

$$t(P) = (\beta_i^P - \alpha_i^P), d_i^P \approx \frac{\sum_{l=1}^{k} d_l}{k}, i \in [1, k]$$

To estimate the success ratio of a transmission between a client and a provider, we consider the k last advertisements received by a (local) client from a provider P, and we compute the average of the distances between them d^P based on their respective location at the emission time and at the reception time. The probability of reaching a provider usually decreases when the distance between the local device and the provider increases. Indeed in this situation, the number of connection disruptions grows, and we must rely on the mobility of the client, of the intermediate devices and of the provider, and on their opportunities of contact, to forward a message successfully. That is why we have defined the success estimation function as a multiplicative inverse function, shifted to the left by -1, which takes d^P as a parameter. This function returns a value close to 1 when d^P is small and a value close to 0 when d^P is big. It is defined as follow:

$$s(p) = \frac{1}{1 + d^p}, d^P = \frac{\sum_{j=1}^{k} d_j^p}{k}$$

4.3 Orchestration *vs* choreography

A service composition request CR is defined by the 5-uplet $\{Id, SL, Rter, R, M\}$, where Id is the composition request identifier, SL is the list of services to compose, $Rter$ is the requester identifier, R is either the partial result in case the request is not finished yet or the final result in case the request is completed. M is the list of mapping rules between previous services outputs and the next services inputs.

Before explaining the differences between the two strategies, we should point out their common similarities. Both composition strategies can be used with both utility functions presented in the previous section, and both strategies support partial compositions which helps reduce failure and allow to resume unfinished composition requests, that have been aborted due to the disappearance of a provider, if new service instances are discovered.

4.3.1 Choreography-based strategy

The choreography-based service composition strategy consists of transmitting to the first selected service provider the composition request, and in delegating to it both the execution of the first service and the selection of the next provider as specified in the composition request. This first provider will then send the rest of the composition request and the result of its execution to the next provider which will carry out the same

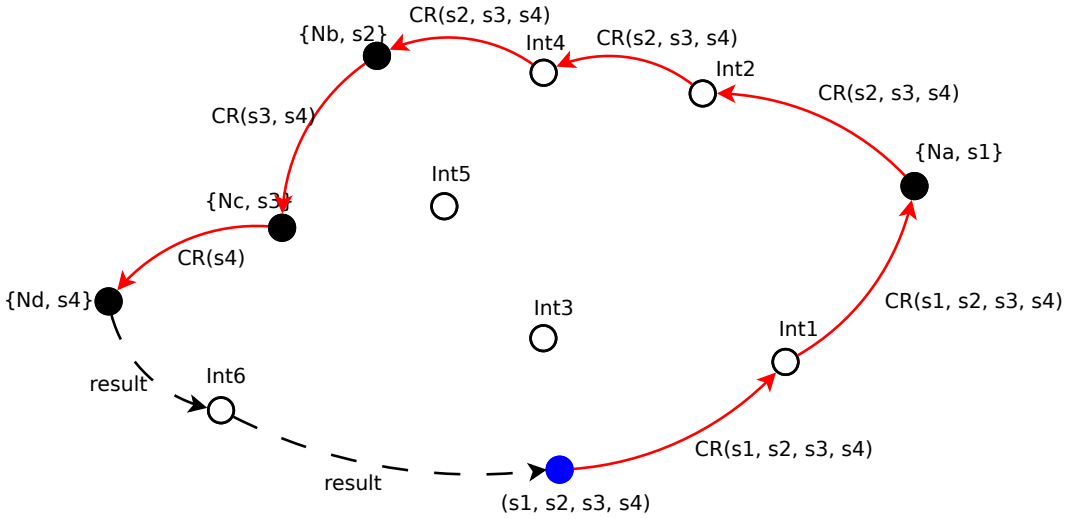


Figure 4.1 – Choreography example.

exact procedure as did the first one. This process will be repeated until the composition is completed. At this point, the result can be returned back to the requester $Rter$. Figure 4.1 depicts a scenario of a service composition using the choreography-based composition strategy. The requester (in blue) issues a CR composed of four services $(s1, s2, s3, s4)$. The requester begins by choosing the nearest service provider for the first service $s1$. The requester selects Na in our scenario. Na is two hops away from the requester so the CR has to be sent to $Int1$. After that, $Int1$will relay CR to Na. Na executes $s1$, updates CR and selects the nearest provider for $s2$ which happens to be Nb in this case. The CR will have to travel through $int2$ and $int3$ to reach Nb since Nb and Na are three hops away from each others. Nb will then pass CR to Nc. Finally CR reaches Nd which executes the last service $s4$ in CR and forwards the final results to the requester

4.3.2 Orchestration-based strategy

Unlike the choreography-based service composition, with the orchestration-based service composition, the composition request never leaves the requester. Furthermore, the tasks of selection and invocation is not delegated to service providers involved in the composite service invocation. Instead, it is the requester that takes in charge the tasks of selection and invocation of providers that should be involved in the composite service invocation. As opposed to the choreogra-phy-based strategy where CR is sent to service providers, the requester invokes services by sending an invocation request IVR to service providers. IVR is formally defined by the 4-uplet $\{Id, S, Rter, I\}$ where Id is the invocation identifier, S is the name of the service to invoke, $Rter$ is the requester identifier and I is the set of inputs required by S. The reply to IVR provided by the service provider is defined as $Rp = \{Id, R\}$, where Id is the same identifier of IVR and

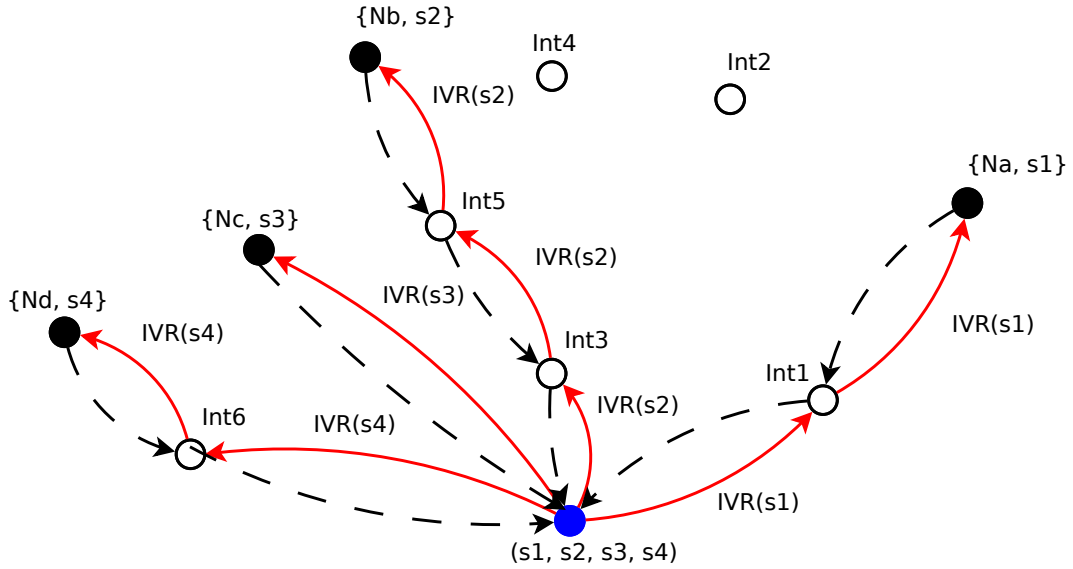


Figure 4.2 – Orchestration example.

R is the result of the service execution. Figure 4.2 provides an example of a composite service executed using the orchestration-based strategy. The requester initiates the same request CR in the choreography example. The requester selects Na to execute $s1$ and sends an IVR to it through $Int1$. After receiving the result from Na, the requester locally updates the CR, selects Nb to execute $s2$ and then send an IVR to it through $int3$ and $int5$. After the requester receives the result from Nb, it will repeat the same process to invoke $s3$ and $s4$ provided respectively by Nc and Nb.

We make sure in our contribution to implement both strategies to be able to compare their performances, and identify their advantages and drawbacks, especially with the orchestration strategy not being considered before in opportunistic networks. Besides, we look forward to be able to combine these two strategies and switch between them throughout the execution of a composition request in way that leverages the advantages of both of them and at same time, avoids their drawbacks.

4.3.3 Mathematical models for composition time estimation and success ratio estimation

Hereafter, we propose both an estimation of the execution time and an estimation of the success ratio of a composition for the two strategies implemented in our service discovery and composition system. Let us consider a composition request C that contains n services identified respectively by $S_i, i \in [1, n]$, and a set of m providers that offer one or several services. Let us also consider that composition C is emitted by a requester Λ.

Success ratio estimation In the orchestration-based strategy, the response of each service invocation is returned to the requester Λ. Thus, the estimation of the success of an

orchestration-based composition is defined as the product of the square of the transmission success estimation of a request from the requester Λ to the different providers enrolled in the invocation of the composite service. This estimation is defined by:

$$\gamma(C) = \prod_{i=1}^{n} s_{\Lambda}(P_l)^2, P_l = \varphi_{\Lambda}(S_i)$$

In the case of the choreography-based strategy, the estimation of the success of a composition is defined as the product of the transmission success estimation of a request from the requester Λ to the first provider (P_1) with the transmission success estimation of the composition result from the last provider (P_n) to the requester Λ, and with the intermediate transmission success estimations.

$$\begin{cases} \gamma(C) = s_{\Lambda}(P_1) \times s_{P_n}(\Lambda) \times \prod_{i=1}^{n-1} s_{P_i}(P_{i+1}) \\ P_1 = \varphi_{\Lambda}(S_1) \\ P_n = \varphi_{P_{n-1}}(S_n) \\ P_{i+1} = \varphi_{P_i}(S_{i+1}) \end{cases}$$

Composition time estimation The composition time of C is defined for the orchestration-based strategy by:

$$\tau(C) = \sum_{i=1}^{n} 2 * t_{\Lambda}(P_l), P_l = \varphi_{\Lambda}(S_i)$$

where P_l is the provider of the service S_i that has been selected by the utility function φ. As mentioned above when presenting the estimation of success ratio, in the orchestration-based strategy, the response must be returned to the device Λ that has initiated the composition request. Thus, the time $t_{\Lambda}(P_l)$ is multiplied by 2 to consider this round trip.

Concerning the choreography-based strategy, the estimation of the composition time is defined by:

$$\begin{cases} \tau(C) = t_{\Lambda}(P_1) + \sum_{i=1}^{n-1} t_{P_i}(P_{i+1}) + t_{P_n}(\Lambda) \\ P_1 = \varphi_{\Lambda}(S_1) \\ P_{i+1} = \varphi_{P_i}(S_{i+1}) \end{cases}$$

where $t_{\Lambda}(P_1)$ is the estimation of the time needed to send the composition request from the requester Λ to the provider of the first service, $t_{P_n}(\Lambda)$ is the estimation of the time needed to send the result of the composition from the provider P_n of the last service S_n to the composition requester Λ. The rest of the formula $\tau(C)$ is the sum of the estimations of intermediate composition times, where $t_{P_i}(P_{i+1})$ is the estimation of the time needed to send both the composition request and the intermediate result from the provider P_i to the next provider P_{i+1}.

4.4 Conclusion

In this chapter, we presented our service discovery and composition system. Its main goal is to carry out both the discovery and the composition processes while maximizing the composition success ratio and minimizing the composition time. We detailed our discovery approach that operates proactively to avoid time delays in the invocation of simple and composite services. Our discovery approach also makes use of local registries since deploying a dedicated directory in opportunistic environments is not reliable. We also presented two implementations of the utility function: one is based on location and distance, and the other is based on time. We then presented our service composition approach that supports both orchestration and choreography. We are interested in the orchestration strategy since it has not been considered yet in opportunistic networks. We also consider that switching between these two strategies, throughout the invocation of the composite service, could help optimizing this process. This switching could be based on which of the two strategies provides a better success ratio and a better composition time.

Chapter 5

Composition caching and precomputing

Contents

5.1 Introduction

In this chapter, we present a solution to optimize the invocation of composite services by using a distributed cache, and by triggering compositions based on the user interest profile. Indeed, a composite service can take a certain time to terminate and can be subject to failures due to the hard conditions of opportunistic networks. Since the same compositions are likely to be executed by different nodes, and their results to be stored in their caches, it could be interesting to retrieve the already existing results rather than starting new composition requests to obtain them, thus saving time and resources. Moreover, to avoid composition delays, it could be interesting to proactively trigger composition requests in advance to populate the cache with composition results before the user decide to execute these composition requests. Nonetheless, it would be unreasonable to execute all the possible composition requests based on the discovered services, due to the problem of limited resources such as battery lifetime, processing power and memory. Therefore, we propose to only compose services that match the user preferences.

The reminder of this chapter in organized as follows. In Section 5.2, we present our proactive composition precomputing manager, and we give a formal description of this one. In Section 5.3, we present our distributed cache manager, and how it is used to

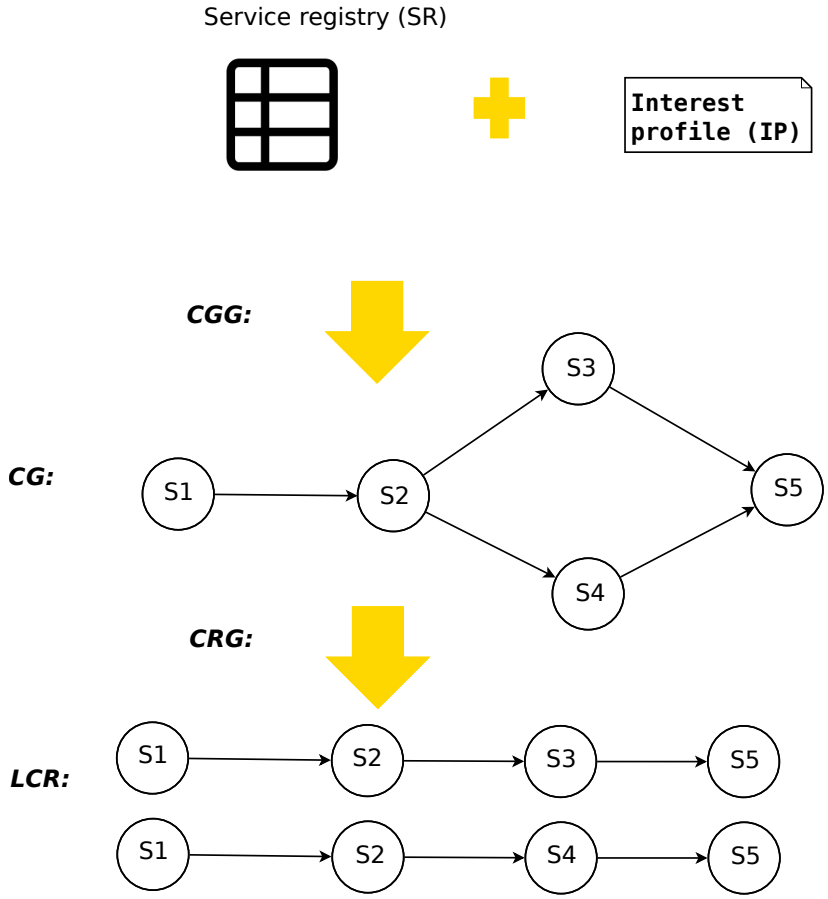


Figure 5.1 – Proactive service composition.

match composition requests to composition results. Finally, in Section 5.4, we present our conclusions.

5.2 Proactive service computing

5.2.1 General overview

To execute service compositions proactively based on the user interest profile, we propose a proactive service precomputing manager (PSP) that represents an extension module for our service discovery and composition system. Our PSP exploits the service descriptions from the service registry (SR) in order to generate compositions automatically. PSP identifies services that match the user interest profile (IP), and defines a composition graph (the graph CG in Figure 5.1), modeling the compositions that can be performed on the basis of the outputs and inputs of services, while respecting pre/post-conditions that should be met.

Afterward, PSP will proceed by identifying the compositions that can be found in the composition graph. PSP will actually perform a depth-first graph traversal in order to find these compositions. PSP will then return them in a form of a list of compositions (the list of composition requests LCR in Figure 5.1).

PSP can be useful for instance for a spectator during a marathon to follow a runner (or a group of runners). A composite service combining a photo delivering service, a bib number detection service and a photo tagging service, can be executed in advance by the PSP to provide the spectator with the photos of the runner(s) he follows with little delay.

5.2.2 Formal description

Service description and service composition graph Each node in the network computes a service composition graph CG that details the compositions possible to carry out according to the user interest profile. This graph is formally defined as $CG = (V, A)$, where V is the set of vertices that contain service descriptions and A is the set of arrows that indicate that two services can be composed in a certain order. Furthermore, a service description, at this level, represents the content of the element D of each service entry in the service registry SR, that we presented in details in Chapter 4. A service description is formally defined by the 7-uplet $D = \{S, I, O, PRE, POST, txtD, kw\}$, where S is the service name, I is the set of inputs, O is the set of outputs, PRE is the set of pre-conditions on the input set I, $POST$ is the set of post-conditions on the output set O, $txtD$ is a textual description and kw is a set of key words that help select services to be included in the composition graph CG based on the user interest profile as it will be explained later on.

I and O represent two sets formed by elements called $IOentry$s. An $IOentry$ (input/output entry) is defined by the couple $\{n, dt\}$ where n is the entry name, dt is the entry data type. A given service a is chained to a service b in the service graph CG, if and only if the outputs of a can be mapped to the inputs of b. An output o of service a is mapped to an input i of service b if only if o and i have compatible data types and the post-conditions on o are included in the pre-conditions on i, which means that the pre-conditions on i are less strict than or as strict as the post-conditions on o. Consequently any variable or parameter, that fulfills the post-conditions on o, automatically fulfills the pre-conditions on i. These conditions are formally expressed by: $Comp(i, o) \wedge (POST[o] \subset PRE[i])$, where $PRE[i]$ are the preconditions on i, $POST[o]$ are the post-conditions on o, and $Comp$ is a function that returns true, if i and o are of compatible data types.

Therefore, an arrow $a \rightarrow b$ is drawn from service a to b if and only if $(I_b \subset O_a) \wedge (POST_a \subset PRE_b)$. $(I_b \subset O_a)$ means that for each $i \in I_b$ there is an output $o \in O_a$, that is not mapped yet to any other input, and that to gather with i, it verifies the condition $Comp(i, o) = true$. $(POST_a \subset PRE_b)$ means that the outputs of a, that can be mapped to the inputs of b based on datatype compatibility, also fulfill the pre-conditions of the service b.

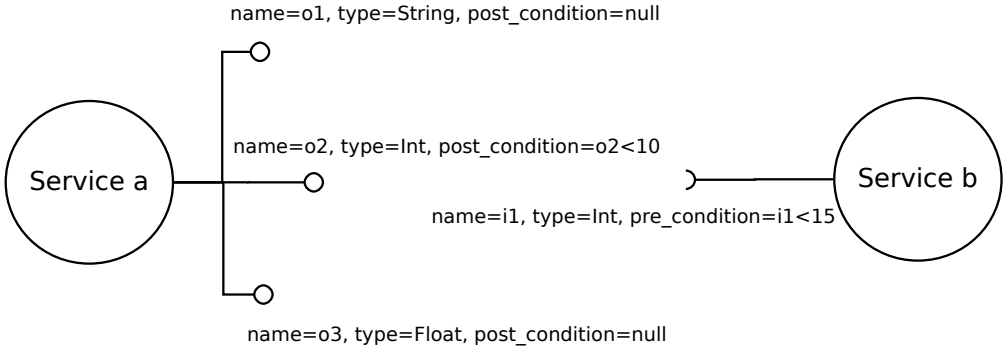


Figure 5.2 – Arrow between service a and b.

Figure 5.2 shows an example of how we establish an arrow between service a and b. Service a has three outputs of type String, integer and float, while service b has one input $i1$ compatible with $o2$ since both are integers. Consequently the condition $Comp(i1, o2) = true$, and since service b has only one input, we conclude that the inputs (I_b) of b are included in the outputs (O_a) of a, which means that the general condition on all inputs and outputs $(I_b \subset O_a) = true$. Moreover $o2 < i1 < 15$, which means that $i1$ can take the value of $o2$. Thus $POST[o2] \subset PRE[i1] = true$ and therefore the second general condition $POST_a \subset PRE_b = true$. Consequently, service a can be composed with b and an arrow $a \rightarrow b$ can be drawn between them.

Proactive composition manager PSP is initialized with a set of keywords $K = \{key_1..key_n\}$ from the user that represent his/her interest profile, where key_i is the i-th keyword. K will be fed to a composition graph generator CGG which is a function defined by: $CGG : K \times SR \times LXD \longmapsto CG$. CGG takes as parameters the keywords K provided by the user, the service registry SR, and the local lightweight lexical database LXD similar to the Princeton's WordNet (83). CGG tries then to find the composition graph CG that is compatible with the user interest profile. The function CGG compares K to kw provided by each service description D in the registry SR using the lexical equivalences provided by LXD in order to find the most compatible list of services that should be included in the composition graph CG.

Afterward, another function called composition requests generator, defined by $CRG : CG \longmapsto LCR$, takes as an input the composition graph CG generated by CGG and finds the list of composition requests LCR that meets the preferences of the user. To do so, CRG will perform a graph traversal using the depth first search algorithm (DFS) on the graph CG. CRG chooses a random node from CG as a root and starts the graph traversal from it. When the traversal is over, CRG selects another node as root and performs another graph traversal. This procedure will be repeated until all nodes were selected as root. The Algorithm 5.1 illustrates how CRG works.

After the LCR is determined, another function $Pr : CR \times Cache \longmapsto I$ is used to prepare the inputs I for each CR in LCR. Indeed, Pr explores the cache memory $Cache$

Algorithm 5.1 Composition request generator algorithm.

```
1:  CRG(CG) :
2:    all_nodes=CG.get_nodes()
3:    previous_roots=[ ]
4:    LCR=[ ]
5:    DFS(node,CR) :
6:      visited_nodes.add(node)
7:      for succ in CG.successors(node) :
8:        if succ not in visited_nodes :
9:          CR=CR + succ
10:         DFS(succ,CR)
11:         CR=CR - succ
12:       end if
13:     end for
14:     visited_nodes.remove(node)
15:     if {R ∈ LCR|CR ⊂ R} = ∅ :
16:       LCR.add(CR)
17:     end if
18:   end DFS
19:   while all_nodes.size() > previous_roots.size() :
20:     remaining_nodes=all_nodes - previous_roots
21:     root=remaining_nodes.pop()
22:     visited_nodes=[ ]
23:     previous_roots.add(root)
24:     DFS(root,[ root])
25:   end while
26:   return LCR
27: end CRG
```

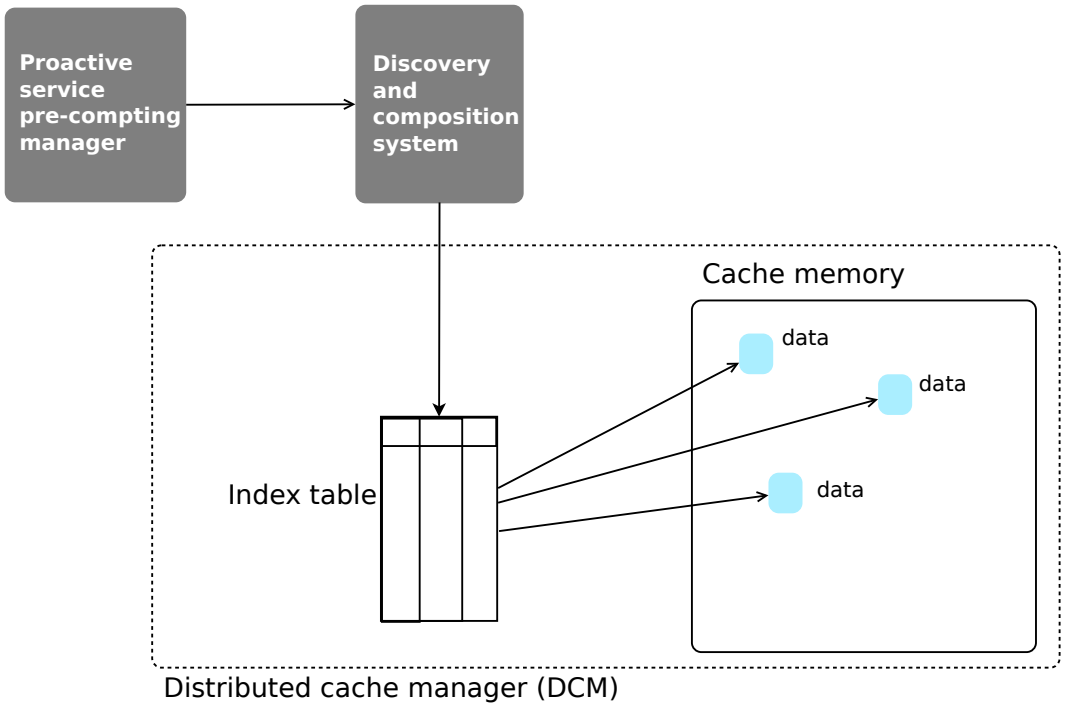


Figure 5.3 – Distributed cache system.

where the data is stored in order to find elements that are suitable to constitute the set of inputs I. Pr will then pass each CR with its inputs I to our service discovery and composition system in order to execute the composition request.

5.3 Distributed cache

5.3.1 General overview

To improve the invocation of composite services in terms of response time and success ratio, we define a distributed cache manager (DCM) that allows to share composition results between nodes in order to avoid repeating compositions, and instead directly retrieve these results from other nodes in the networks. To do so, we consider the set of caches available in the network as a distributed storage space, where composition results (complete or partial) can be shared and replicated between nodes that have interest in them. Therefore, we propose to extend the local cache of each node using an index table to identify composition results, produced both locally and by other nodes, based on a checksum of their inputs and the list of the services involved in the composition. DCM has been integrated in our service discovery and composition system.

Therefore, as shown in the example provided by Figure 5.3, when the PSP starts a composition request CR from LCR, our service discovery and composition system checks in the index table if there are partial or complete results that match CR by com-

paring the checksum of the CR inputs and the list of services to compose in CR, to those of the results listed by the index table. In case there are matches, the service discovery and composition system will return these results, and thus we avoid starting composition requests that are resource-hungry and time consuming.

5.3.2 Formal description

Our index table T is organized in a form of an AVL tree to optimize search operations. The table $T = \{itentry_1..itentry_n\}$ is formed by a set of $itentry$ elements. An $itentry$ (index table entry) is defined by the 5-uplets

$\{inputCS, sq, right, left, pointers\}$, where $inputCS$ is the checksum of the request inputs, and sq is the sequence of the service names $\{S_1, .., S_n\}$ according to the order of execution (if the composition is partial, sq corresponds to the services already executed). $pointers$ is the set of the references $\{p_1..p_n\}$ to the composition results $\{R_1..R_n\}$ in the local node cache that correspond to $inputCS$ and to sq. $right$ and $left$ are respectively the right child and the left child of the current $itentry$ in the tree. The table T is sorted first based on $inputCS$ and then on sq. To compare two service sequences $sq_a = \{S_1..S_n\}$ and $sq_b = \{S_1..S_m\}$, we concatenate the service names of both of them to obtain two strings $s^a = S_1^a|..|S_n^a$ and $s^b = S_1^b|..|S_m^b$. Then, we use the alphabetical order to compare these two strings. A given composition result R (partial or complete) is defined by the 6-uplet $\{id, inputCS, sq, ouputCS, expDate, P\}$, where id corresponds to the identifier of the composition request CR associated to R, P is the payload of the result, $expDate$ is the expiration date of the composition result after which this result will be removed from the cache, and $outputCS$ is the checksum of the payload P.

We use the function $search$ to look for existing results when we want to execute a certain composition CR. This function is defined by:

$$search(root, CRinputCS, CRsq) = \{itentry \in T | CRinputCS = itentry.inputCS \; and \; itentry.sq \subseteq CRsq \; and \; CRsq[0] = itentry.sq[0]\}$$

where $root$ is the root of the index table T, $CRinputCS$ is the checksum of the inputs of the request CR and $CRsq$ is the sequence of the names of the services to compose in CR. The function $search$ takes these two parameters and returns the list of $itentry$ that have an $inputCS$ equal to $CRinputCS$ and a service sequence sq included in $CRsq$ (for partial results) or equal to $CRsq$ (for complete results). The $search$ function performs a typical AVL tree search as detailed in Algorithm 5.2. This function starts by finding entries that have a $root.inputCS = CRinputCS$. The function $search$ will then compare the service sequences $CRsq$ and $root.sq$ (the service sequence of the current root), using the function $concat()$ to concatenate service names of both sequences, and compare them based on the alphabetical order.

If $concat(CRsq) < concat(root.sq)$, we obtain two cases:

1. $CRsq = \{S_a, S_b, S_c\} \subset root.sq = \{S_a, S_b, S_c, S_d\} \wedge CRsq[0] = root.sq[0] = S_a$. This shows that $CRsq$ is a partial sequence of $root.sq$ and that $root$ offers a longer composed sequence of services than our sequence.

2. $CRsq = \{S_a, S_b, S_c\} \not\subset root.sq = \{S_h, S_j, S_c, S_w\} \vee CRsq[0] \neq root.sq[0]$. In this case, we consider that $CRsq$ and $root.sq$ are completely different.

In both cases, we just continue the search without adding the current root to the list of results.

If $concat(CRsq) > concat(root.sq)$,we also obtain two cases:

1. $root.sq = \{S_a, S_b, S_c\} \subset CRsq = \{S_a, S_b, S_c, S_d\} \wedge CRsq[0] = root.sq[0] = S_a$. This means that $root.sq$ is a partial sequence of $CRsq$ and that $root$ points to partial composition results for our composition request CR. Consequently $root$ is added to the list of results. We then subtract $root.sq$ from $CRsq$ to obtain the rest of the sequence to compose $rest\_sq$. We call the function $search()$ recursively using $rest\_sq$ to look if there are results for the rest of the composition request.

2. $CRsq = \{S_a, S_b, S_c\} \not\subset root.sq = \{S_h, S_j, S_c, S_w\} \vee CRsq[0] \neq root.sq[0]$. $CRsq$ and $root.sq$ are considered, in this case, completely different. Therefore, we just continue the search without adding $root$ to the list of results.

Finally, if $concat(CRsq) = concat(root.sq)$, this means that $root$ offers a complete service sequence, and that it points to a set of complete composition results in the cache for our composition request CR. Thus we add $root$ to the list of results.

Moreover, to enable nodes to share and replicate composition results among each other, we can adopt a content-based approach based on a publish/subscribe communication mode, where a given node associates to each composition request CR, that it considers interesting, a topic T_{CR}. This will allow this node to provide and receive results related to CR everytime it comes in contact with another node.

5.4 Conclusion

In this chapter, we presented our proactive service precomputing manager (PSP) used to automate services compositions. It relies on a composition graph that provides the available and potential compositions to initiate based on the user preferences. PSP preemptively triggers composition requests, in order to populate the cache with results, before the user decide to launch these requests, in order to reduce composition delays. We also presented our distributed cache manager (DCM) that allows to share and replicate composition results between nodes to avoid starting unnecessary invocations of composite services. DCM searches for existing composition results, previously performed locally or by remote nodes, that match the composition request that we want to perform. By using DCM, we try to save time and resources that are already scarce in opportunistic networks.

Algorithm 5.2 The search function algorithm.

```
1:  search(root, CRinputCS, CRsq) :
2:    original_root=root
3:    results=[] //the list of results to return
4:    while root != null :
5:      if CRinputCS > root.inputCS :
6:        root=root.right
7:      else if CRinputCS < root.inputCS :
8:        root=root.left
9:      else :
10:       //concat(sq) concatenates service names from sq
11:       if concat(CRsq) < concat(root.sq) :
12:         //CRsq is either included in root.sq or
13:         //completely different from root.sq. Both cases are uninteresting
14:         root=root.left
15:       else if concat(CRsq) > concat(root.sq) :
16:         //In this case, root.seq is either included in CRseq,
17:         //or completely different
18:         if root.sq ⊂ CRsq and root.sq[0]==CRsq[0] :
19:           results.add(root)
20:           rest_sq=CRsq-root.sq
21:           for pt in root.pointers :
22:             comp_results=search(original_root, pt→outputCS, rest_sq)
23:             results.add(comp_results)
24:           end for
25:         end if
26:         root=root.right
27:       else :
28:         results.add(root)
29:         root=root.right
30:       end if
31:     end if
32:   end while
33:   return results
```

Part III

Implementation, Evaluations and Conclusion

Chapter 6

Implementation

Contents

6.1 Introduction

In this chapter, we start by introducing, in Section 6.2, the opportunistic communication framework C3PO (67) used in the development process. We then present, in Section 6.3, the implementation of our service discovery and composition system, as well as the implementation of our distributed cache manager (DCM) that constitutes an extension to our system. In Section 6.4, we present the design of our proactive service precomputing manager (PSP) that automates service compositions based on the user preferences. Each implementation, in this chapter, is presented using a general overview and a detailed conception using UML class diagrams. Finally, we draw our conclusions in Section 6.5.

6.2 C3PO

C3PO is a framework dedicated to opportunistic networking and developed under the project ANR-C3PO [1]. In this project, the C3PO framework is used to create and manage social networks that are limited in space and time. These networks are called *SESN (Spontaneous and Ephemeral Social Networks)* (67). C3PO provides two communication

[1] http://www.agence-nationale-recherche.fr/Projet-ANR-13-CORD-0005

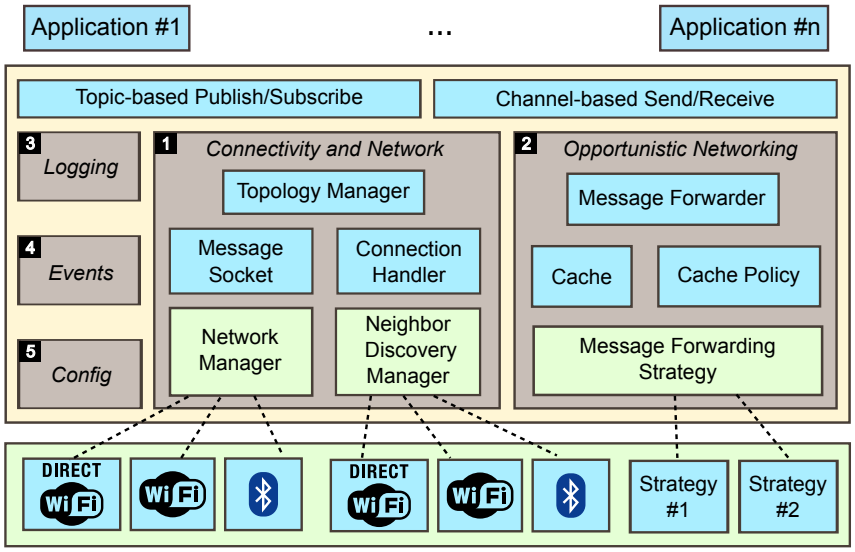


Figure 6.1 – C3PO framework.

modes: a point-to-point communication mode using channels, and a publish/subscribe communication mode using topics.

C3PO has a modular architecture. It is organized in five main modules as shown in Figure 6.1. The connectivity and network module (i.e. module 1) is responsible for managing wireless interfaces and organizing the topology of the network. This module implements a neighbour discovery manager, that is responsible for discovering other nodes and maintaining the topology. The discovery manager implements a beaconing process and uses the discovery mechanisms integrated in certain network communication technologies (e.g. Wi-Fi Direct, Bluetooth, etc.). The discovery manager also allows to perform a 2-hops discovery of nodes. Module 1 also implements a network manager, that creates and manages connections between the nodes that have been discovered. In addition, this module has a topology manager that manages the topology of network according to the constraints imposed by both the operating systems and the network technologies. The topology manager prevents the creation of redundant links between nodes, and also creates and manages micronets and macronets. A micronet is a subset of nodes connected using the same communication technology. Nodes in the same micronet are able to communicate directly with each other. A micronet serves essentially as an abstraction for a Bluetooth piconet, a Wi-Fi BSS or a Wi-Fi direct group. On the other hand, a macronet is a group of micronets interconnected through nodes that are at least members of two micronets.

The opportunistic networking module (module 2) implements a cache to store messages that are exchanged opportunistically by nodes. It also provides message forwarding strategies and a message forwarder in order to organize data dissemination

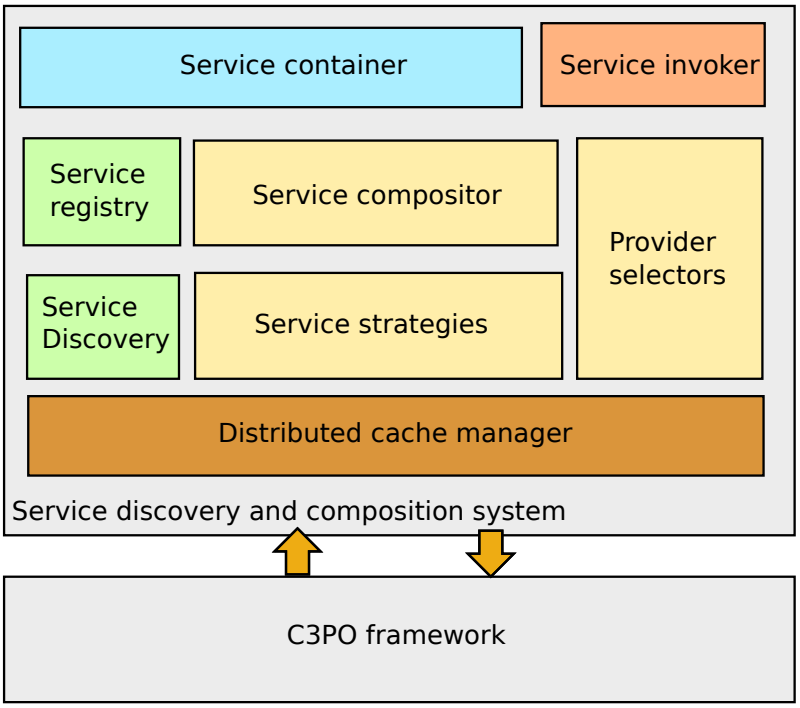


Figure 6.2 – General architecture of the service discovery and composition system.

within the network based on the store-carry-and-forward principle (detailed in Chapter 2). C3PO uses an optimized version of the Epidemic routing protocol to limit the number of messages exchanged in the network. The three other modules implement functionalities that include the management of events produced by the framework, a logging system for the traces generated by the framework and a configuration module.

6.3 Service discovery and composition system

6.3.1 Overview of the architecture

Our service discovery and composition system is formed by several components as shown in Figure 6.2. The *service container* hosts services that are provided locally. The *provider selectors* is the set of the implementations available for the utility-function used by the system to rate and to select providers. The *service registry* is the directory where advertisements from other providers as well as the descriptions of local services are stored. This directory is populated by the *service discovery* component that implements our discovery approach. The *service compositor* is the component responsible for executing a composition request. This component chooses the service providers that should be enrolled in the invocation of the composite service. It also updates the composition request by mapping the outputs of current service to the inputs of the next one, and delivers the final results at the end of the invocation of the composite service. To choose

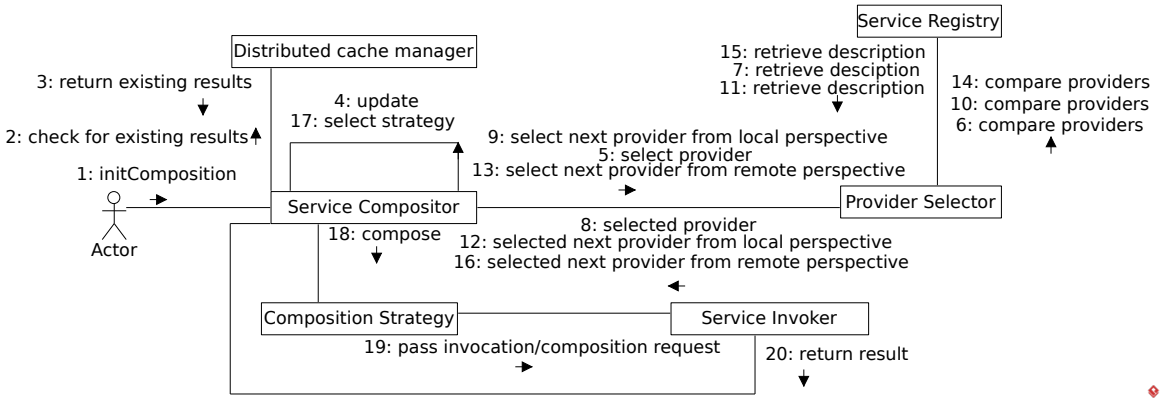


Figure 6.3 – Communication diagram for the service discovery and composition system.

which provider to invoke, the *service compositor* uses one of the available *provider selectors.* After the chosen provider ($i+1-th$ provider) is determined, the *service compositor* should be able to compare the best next service provider ($i+2-th$ provider) from the perspective of the local node ($i-th$ provider or the request initiator), with the best next provider from the perspective of the chosen provider ($i+1-th$ provider*).* Based on this comparison, the *service compositor* should be able to choose dynamically, in the middle of the invocation of the composite service, one of the *Service strategies* that dictates how the composition request should be executed. In the current implementation, the composition strategy is fixed from the beginning, and does not change throughout the execution of the composite service. The *service invoker* is then used to send invocation and composition request messages to service providers. The *service compositor* will also try to avoid this resource-consuming process by checking with the *Distributed cache manager,* if there are already existing composition results that answer the composition request. These interactions between the different system components are illustrated by the communication diagram in Figure 6.3.

6.3.2 Details

Class diagram of the service discovery and composition system and the DCM Figure 6.4 shows the class diagram of the service system and the distributed cache manager. In this figure, the *Compositor* has the responsibility of managing the invocations of composite services. The *start()* method of the class *Compositor* allows to configure and to start the activity of this class, whereas the *stop()* method implements the necessary steps to stop the *Compositor*. The method *startComposition()* allows the higher software layer to start a service composition. This method takes as parameters an instance of the class *CompositionRequest* and an instance of the class *CallBack.* The *Callback* instance processes the final result when this one is received by the requester. The attribute *compositionList* holds the list of the ongoing compositions. The *update()* method updates the composition request and maps service outputs to service inputs throughout the invocation of

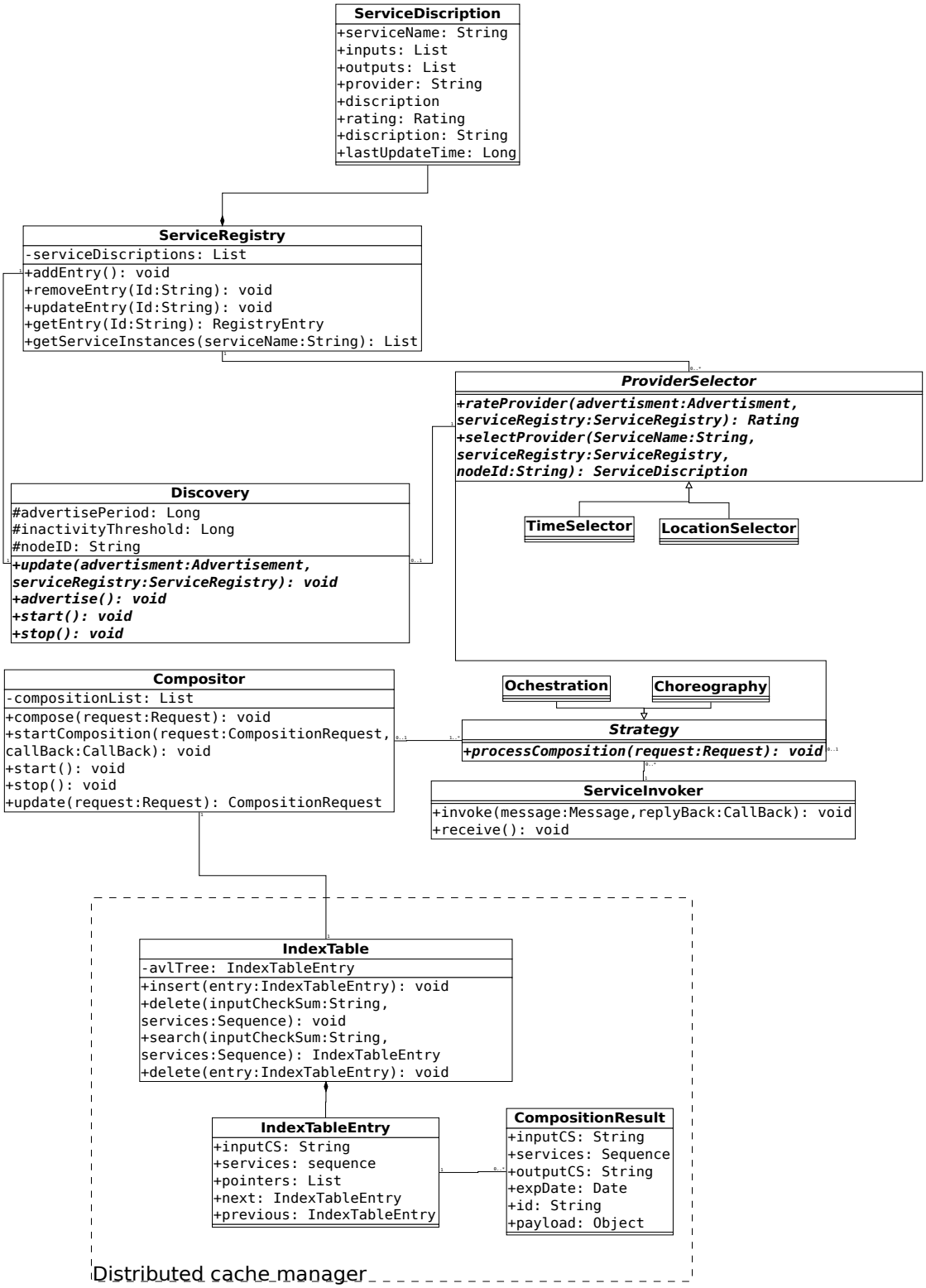


Figure 6.4 – Class diagram of the service discovery and composition system.

the composite service.

The *IndexTable* represents the main class of our DCM. The *IndexTable* class defines an attribute called *avlTree* which is an instance of the class *IndexTableEntry* and the root of an AVL Tree. The *IndexTable* class provides 4 methods: *search()* to look for an entry, *insert()* to add an entry, and two *delete()* methods to remove entries from the tree.

The class *IndexTableEntry* defines the attribute *pointers* that points to a set of composition results that have an equal checksum of inputs *inputCS* and the same sequence of *services.*

Composition results are represented by the class *CompositionResult,* where *payload* is the content of the result, the attribute *outputCS* is the checksum of the *payload*, *expDate* is the expiration date of the composition result, and *id* is the identifier of the composition request associated to the result.

The abstract class *Strategy* dictates how the composite service should be executed. The *Strategy* class defines the *processComposition()* method that invokes service providers. Currently, we implement two strategies: the *Orchestration* and the *Choreography*.

The abstract class *ProviderSelector* is used to select service providers using the method *selectProvider()* that returns a *ServiceDiscription* instance of the selected service. *ProviderSelector* has also the role of rating service providers using the method *rateProvider*(). A class, that extends the *ProviderSelector* class, should provide an implementation of our utility function. We currently propose two implementations of the class *ProviderSelector: LocationSelector* that relies on location and distance and *TimeSelector* that relies on time.

The class *ServiceInvoker* is used to invoke remote services. This class defines the method *invoke()* that takes as parameter an instance of the class *Message* that carries the request and an instance of the class *Callback* that processes the result upon receiving it.

The *Discovery* class is responsible for managing the service discovery process. It has two main methods: the *update()* method which updates the *ServiceRegistry* when an advertisement message is received, and the method *advertise()* that periodically sends an advertisement message of the local services based on the class *Advertisement*. The *Discovery* class also has a *start()* and *stop()* methods that have the same purpose as those of the *Compositor* class.

The local service directory of our service system is implemented by the class *ServiceRegistry.* This class has an attribute called *serviceDiscriptions* which is a list of instances of the class *ServiceDiscription* that describe both local and remote services. *ServiceRegistry* also provides several methods to manipulate this list. For instance, *addEntry()* and *removeEntry()* respectively allow to add and remove an instance of the class *ServiceDiscription* from the description list. The method *updateEntry()* is used to update an already existing instance of the class *ServiceDiscription* and finally both *getEntry()* and *getServiceInstances()* return instances of the class *ServiceDiscription* respectively based on identifiers and names.

Class diagram of the messages Figure 6.5 shows the UML class diagram of the different messages exchanged by nodes running our service system. In this figure, all

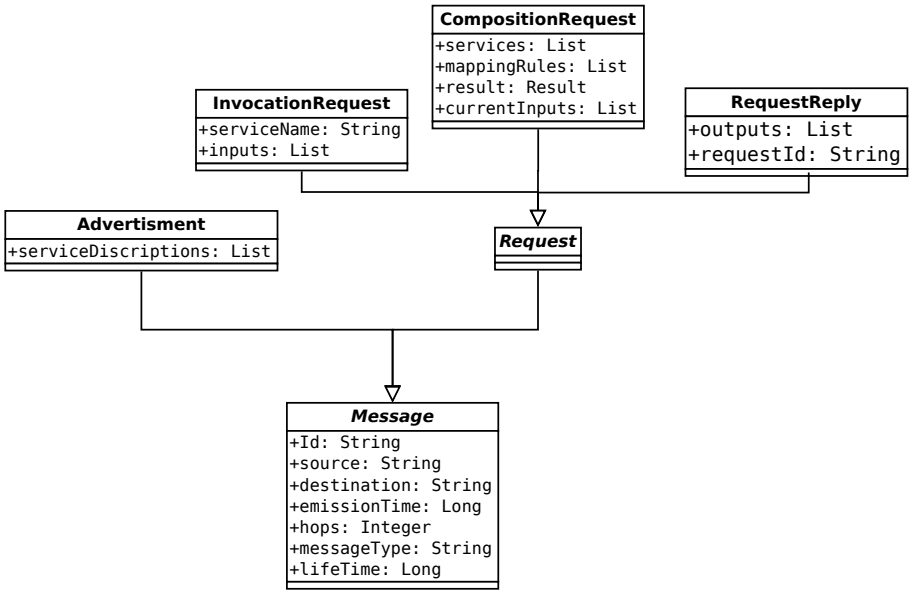


Figure 6.5 – Class diagram for messages exchanged by the service discovery and composition system.

classes extend the class *Message*. This class defines several attributes: *id* is the message identifier, *source* is the address of the sender, the attribute *destination* specifies the destination address. This attribute can be assigned a node address in case of a point-to-point communication or a wild-card expression if the message is meant to be broadcast. The *emissionTime* is the time on which the sender emitted the message. *hops* is the number of hops the message can still do. *hops* starts at the maximum number of hops a message is allowed to do and is decreased by -1 everytime the message is received by a node. The *lifetime* is the time after which the message is considered to be obsolete.

The class *CompositionRequest* specifies the attributes of a composition request. Indeed, the attribute *services* enlists the sequence of services to compose. The *mappingRules* are the set of rules that dictate how one service outputs should be mapped to another service inputs. The *result* attribute is assigned either the final composition result or the partial composition result. The *currentInputs* parameter holds the inputs for the next service to execute. *currentInputs* gets its value from the results of mapping performed using the *mappingRules*.

The *InvocationRequest* class is used to form the invocation request message for a single service invocation. This class specifies the service name *serviceName* and the *inputs* that the service requires.

The reply to an invocation request is modeled by the class *RequestReply*. This class specifies the *outputs* that the service returns after the execution, and the identifier *requestId* of the invocation request associated to the reply. The *InvocationRequest*, the *RequestReply* and the *CompositionRequest* extend the abstract class *Request* .

The *Advertisment* class is used in the discovery process by service providers to ad-

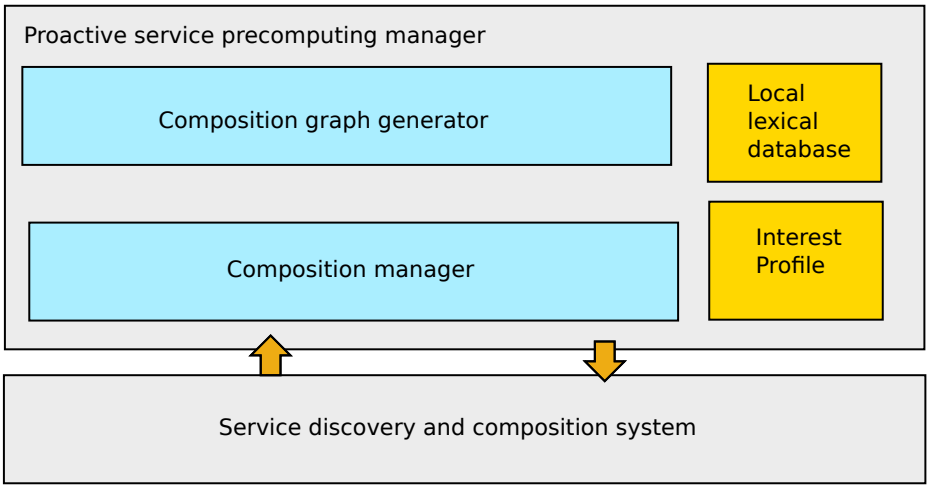


Figure 6.6 – Data sharing space general architecture.

vertise their services across the network. Like other classes, *Advertisment* extends the class *Message* and adds a list of service descriptors called *serviceDiscriptions.*

6.4 Proactive service precomputing manager

6.4.1 Overview of the architecture

Figure 6.6 shows the general architecture of our proactive service precomputing manager (PSP). Our PSP is formed by two major components: the *composition graph generator (CGG)* and the *composition manager (CM).*

The *CGG* main purpose is to generate the composition graph from the service descriptions stored in the service registry according to the user preferences. *CGG* tries to find services that match the user/developer *Interest profile* by using the *Local lexical database*. The *Local lexical database* matches the keywords from the user interest profile to those in the service descriptions to determine the most suitable services. Afterward, *CGG* proceeds by mapping the outputs of one service to the inputs of another. The mapping is based on the input/output datatype compatibility and the coherence between the inputs preconditions and the outputs post-conditions.

CM uses the graph provided by *CGG* to identify the compositions that suit the developer/user interest. As explained in Chapter 5, *CM* performs a depth first search (DFS) graph traversal on the composition graph and builds a composition request list. *CM* then prepares the inputs for every composition request in the list using the local node cache and passes these requests to our service discovery and composition system. *CM* will repeat this operation every time there are new inputs available in the cache.

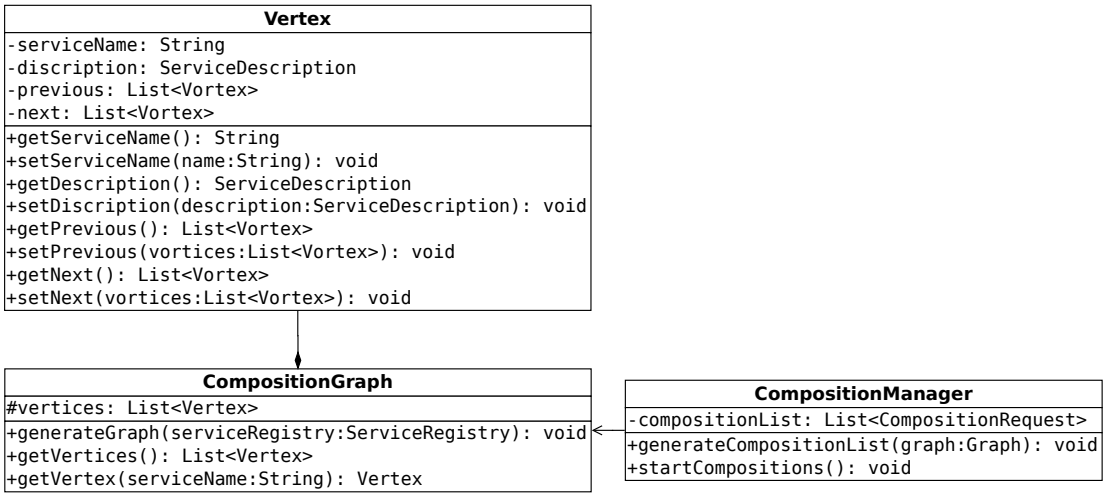


Figure 6.7 – Data sharing space class diagram.

6.4.2 Details

Figure 6.7 shows the class diagram for our PSP. In this figure, the class *CompositionGraph* represents the *Composition graph generator* component, and provides an implementation of a directed composition graph. The graph, provided by the *CompositionGraph* class, is formed by a list of instances from the class *Vertex.* The *Vertex* class contains the service name *serviceName* and its *description.* The *Vertex* class also specifies a list of vertices, called *previous*, that describes the services that can precede the service, described by the current instance of *Vertex*, in a service composition. The *Vertex* class also provides a list of vertices, called *next*, that describes the services that can follow the service, described by the current instance of *Vertex*, in a service composition. The *CompositionGraph* class builds the composition graph using the method *generateGraph()* that takes as a parameter the service registry *serviceRegistry.*

The class *CompositionManager* has an attribute called *compositionList* that contains the compositions that match the user interest profile. This class uses the method *generateCompositionList()* to generate the content of the attribute *compositionList* using the graph provided by the *CompositionGraph* class. The *CompositionManager* class also uses the method *startComposition()* to start these compositions every time there are new inputs in the cache.

6.5 Conclusion

In this chapter, we introduced the framework C3PO used in the development process. We presented the implementation of our service discovery and composition system and the implementation of the DCM that extends it. We explained the general architecture of this system and we presented its main components. We also illustrated the dynamic

behaviour of the system using an UML communication diagram. Then, we detailed its conception using an UML class diagram for the different messages exchanged by our system, as well as an UML class diagram that elaborates how we implemented the main components of the system. Finally, we discussed the implementation of our PSP.

Chapter 7

Comparison of orchestration-based and choreography-based strategies

Contents

7.1 Introduction

In this chapter, we compare the orchestration-based strategy and the choreography-based strategy. We identify the advantages and disadvantages of each strategy. We study the performances of these strategies, essentially, in terms of success ratio and composition time, by varying the maximum number of hops between the service provider and the service client, and by using the time-based implementation of our utility function presented in the Chapter 4.

The reminder of this chapter is organized as follows. In section 7.2, we present the lightweight emulator LEPTON. In Section 7.3, we detail the setup for our experiments. In Section 7.4, we present our results and we analyze them, and in Section 7.5, we draw our conclusions.

7.2 LEPTON

LEPTON *(Lightweight Emulation PlaTform for Opportunistic Networking)*[1] is an emulation platform dedicated to opportunistic communication support. The main objective of

[1] http://casa-irisa.univ-ubs.fr/lepton/

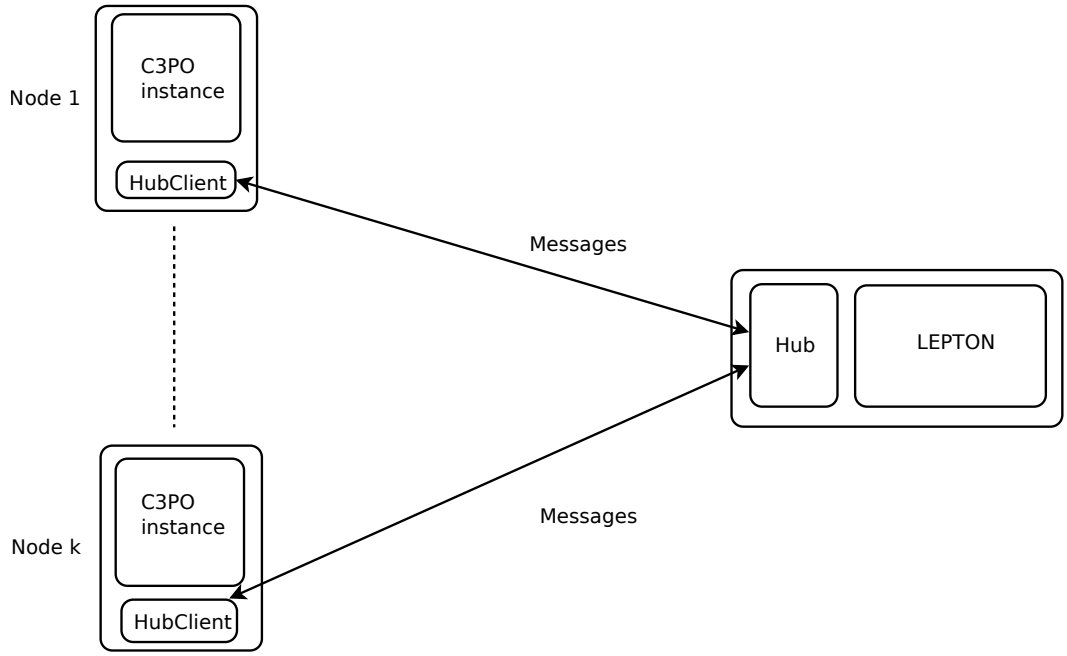


Figure 7.1 – C3PO/LEPTON integration.

LEPTON is to enable researchers and developers to carry out experiments using their own opportunistic systems while providing simulated mobility models for them.

Being a lightweight emulator, as it claims to be, LEPTON does not require any special or high performance equipment to be deployed. In fact, for experiments that include only a couple of hundred nodes, a typical workstation configuration can be enough to guarantee a proper execution of the emulation. LEPTON can also be deployed on a cluster of machines in order to carry out an emulation that is high demanding in resources.

Moreover, as an emulator, one of its principle task is to drive the communications between the different instances of an opportunistic system. This is done by taking into account the mobility, the distance between nodes, the radio communication technologies involved, and several other factors (battery lifetime, load, etc.) to determine if a given pair of nodes can or can not exchange messages. Therefore, any opportunistic system, that is willing to use LEPTON, should implement a integration layer. Currently LEPTON support several opportunistic systems such as DoDWAN (52), IBRDTN (38) and aDTN[2].

Figure 7.1 shows how we integrated C3PO with LEPTON. This integration is based on a client/server model. In fact, we add a software layer, that we call Hub. This layer plays the role of the server for the emulator LEPTON. The Hub is responsible for relaying messages between nodes. It notifies them when they are in radio range of each other, and instructs them to end the connection when they are out of range, all based

[2]https://github.com/SeNDA-UAB/aDTN-platform

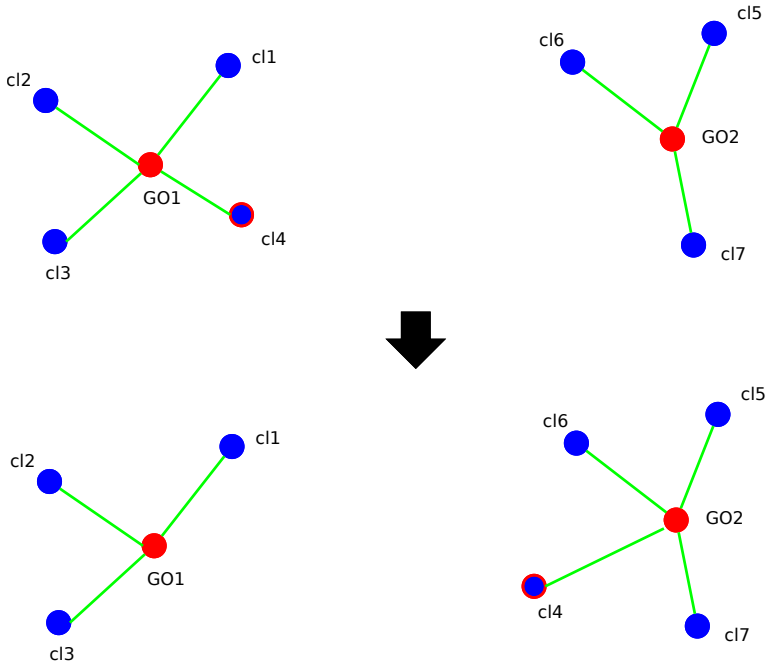


Figure 7.2 – Wi-Fi Direct scenario.

on the simulated mobility provided by LEPTON. On the other hand, C3PO instances are provided with another software layer called HubClient. The HubClients allow the C3PO instances to send and receive messages to and from other nodes through the Hub, that plays the role of an intermediate.

7.3 Evaluation setup

In order to compare the orchestration-based strategy and the choreography-based strategy, we perform a series of experiments in which we emulate our system using our emulator LEPTON. For our evaluations, we consider two scenarios: the first scenario is an open area of $500 \times 500\,m^2$ where people can roam around freely following the Levy walk mobility model (103). The second one is a sport event that took place in the city of Vannes in France (Figure 7.3) where attendees move along a predetermined running path. Coupled with the two strategies presented in Chapter 4, we obtain 4 configurations that are identified respectively as: open area/orchestration, open area/choreography, Vannes city/orchestration, and Vannes city/choreography. Pedestrians walk at a speed between 0.5 m/s and 2 m/s, while using their smart-phones to start compositions. The number of pedestrians varies between 50 and 250. The communication technology considered in the experiments is Wi-Fi Direct (23), using the Android implementation, with a maximum radio range of 80 m. Figure 7.2 illustrates how Wi-Fi Direct works. Wi-Fi Direct divides the network into communication groups. Each com-

Parameter	Value
Open area size	500 m x 500 m
Composition request generation	between 2 and 5 min.
Evaluation Duration	1 hour
Advertisement period	10 seconds
Service registry entry inactivity threshold	20 seconds
Number of service per composition	between 3 and 6
Number of local services per node	5
Number of hops in advertisement messages	between 1 and 3
Number of node per experience	50, 100, 150, 200, 250
Speed range	between 0.5 and 2m/s

Table 7.1 – Evaluation parameters.

munication group is managed by a group owner GO that acts as a soft access point. The GO is chosen at the beginning, when the first two nodes of the group establish a connection. A given node can join possibly any group. In the Android implementation, the number of clients is limited to 7 per GO. A client can only be connected to one GO, and two GO can not establish a connection between themselves. As Figure 7.2 shows, if $cl4$ wants to connect to $GO2$, it must first end its connection with $GO1$. Likewise, Wi-Fi Direct causes the network to be highly fragmented and formed by isolated connected islands.

In our evaluation, each emulation lasts 1 hour and each node in the network provides 5 services that can be invoked. Moreover, we suppose that each node generates a composition request formed by 3 to 6 remote services (we do not include local services in composition requests) every x time, with x between 2 and 5 minutes. All the evaluation parameters are summarized in Table 7.1.

7.4 Results and analysis

The results presented hereafter, show the effect of the maximum number of hops between the service client and the service provider on the success ratio, on the composition time, on the number of nodes involved in a service composition and on the number of compositions executed by a node.

Composition time Figure 7.4a depicts the median of composition time against the maximum number of hops between the service provider and the service client. We notice that the median of composition time increases when the maximum number of

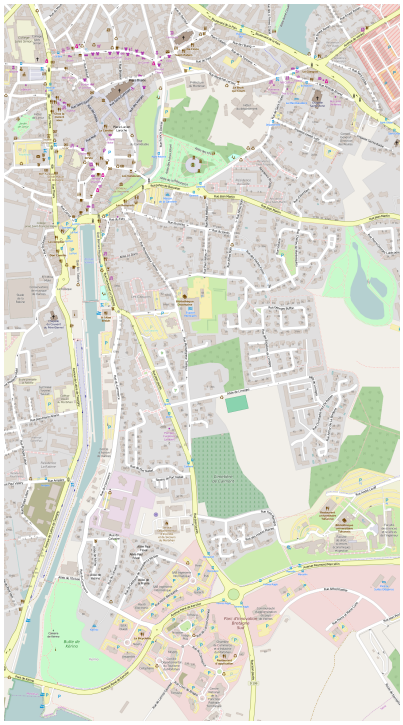
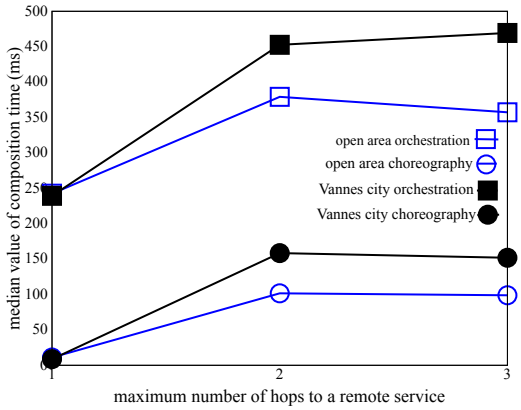


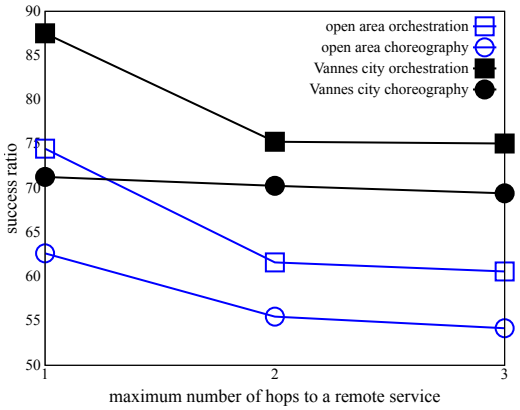
Figure 7.3 – Vannes city map.

hops increases in the four configurations. Clearly, the choreography-based composition strategy provides a shorter composition time than the orchestration-based strategy. Thus, it provides a better response time to the composition requester. For example, in the open area scenario, the median time for the choreography-based composition strategy to finish a service composition is between 11 ms and 101 ms, whereas the orchestration-based composition strategy takes between 242 ms and 379 ms in order to complete a composition. In the Vannes city scenario, the choreography-based strategy has a median of composition time between 8 ms and 158 ms while the orchestration-based composition strategy scores between 239 ms and 469 ms.

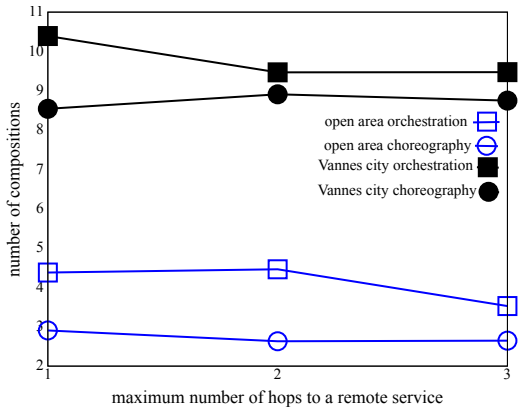
Figure 7.5 shows the composition time distribution for a maximum number of hops between the service client and the service provider equals to 2. We can observe on this figure that the orchestration-based composition strategy has almost all its composition times between 200 ms and 600 ms, with a maximum pick between 15 % and 25 % in the open area scenario, and between 10 % and 14 % in the Vannes city scenario. The orchestration strategy has, nonetheless, nearly 0% of compositions that have a composition time between 0 ms and 200 ms. However, for the choreography-based composition strategy, almost all composition times are between 10 ms and 200 ms with a maximum pick between 20 % and 25 % in the open area scenario, and between 10 % and 14 % in the Vannes city scenario both of them happens around 10 ms. This can be explained by the fact that with the orchestration-based composition strategy, the requester plays also



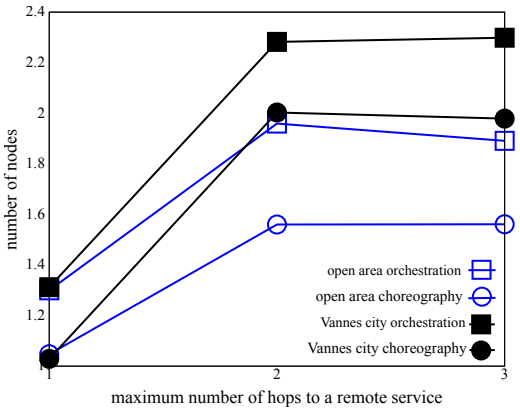
(a) Median composition time.



(b) Composition success ratio.



(c) Average number of compositions executed by a device.



(d) Average number of nodes involved in an invocation of a composite service.

Figure 7.4 – Impact of the number of hops on service compositions.

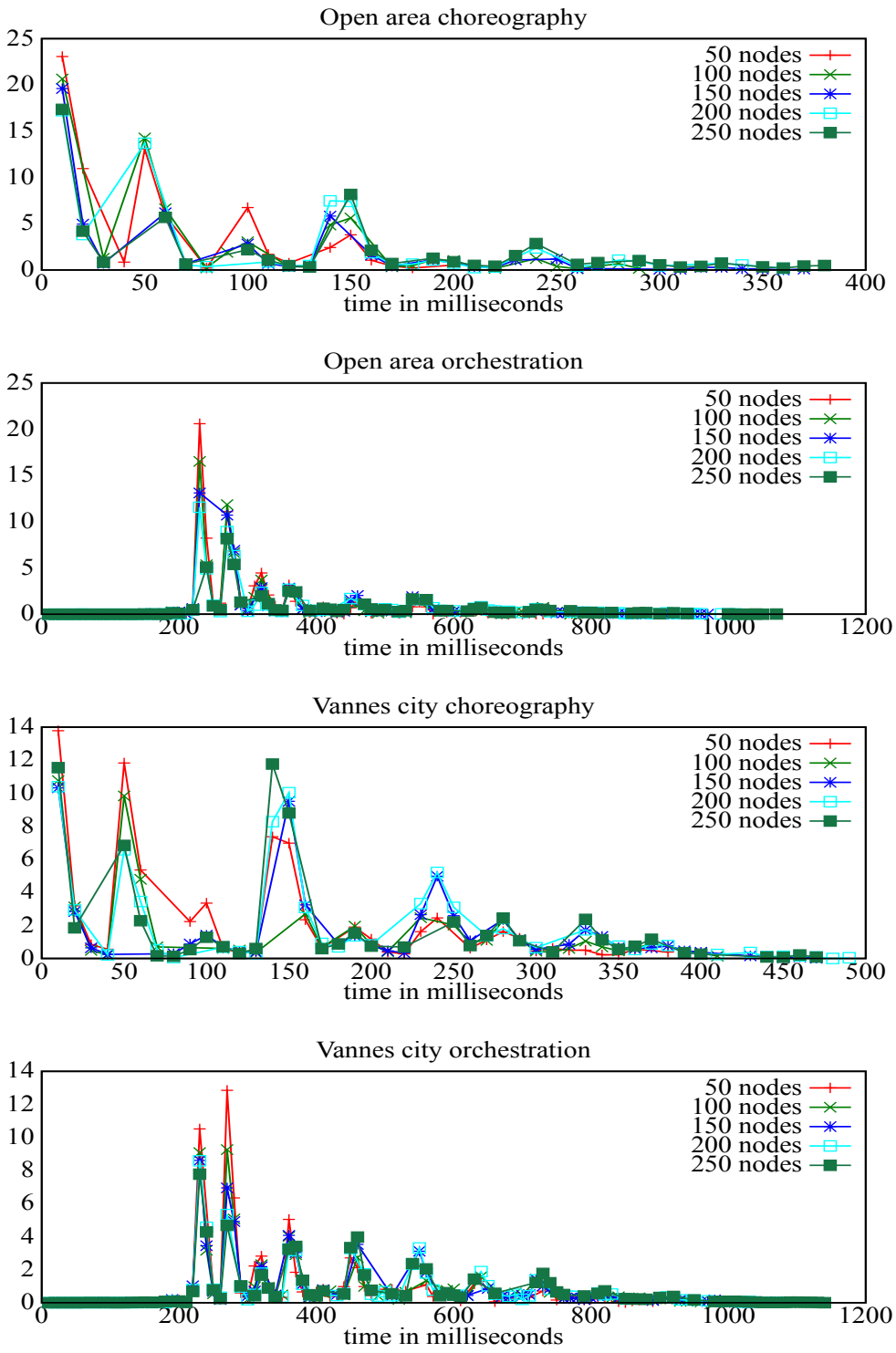


Figure 7.5 – Composition time distribution for 2 hops.

the role of the orchestrator. Thus, all service execution results should be returned to it in order to continue the invocation of the composite service. In addition, service requests and service results are likely to travel through intermediate nodes. Consequently, the orchestration-based composition strategy takes a long time to finish. Whereas, with the choreography-based composition strategy, it is only the final result that is returned to the requester. Intermediate results are indeed only sent by the current service providers to the next one and are never returned to the requester. Likewise, the choreography-based composition strategy generates less messages than the orchestration-based composition strategy, and thus reduces considerably the composition time.

Moreover this important time gap between the two strategies is due to the process of formation of groups in Wi-Fi Direct and of micronets in the C3PO communication middleware. As previously mentioned, with the orchestration strategy, the composition request stays with the requester. This one is in charge of selecting providers and invoking them throughout the execution of the composition request. This creates a difference in composition time between the two strategies especially when the maximum number of hops of messages is limited to one hop, because the requester has only access to its local services and the services of its group owner. Thus, a requester, using the orchestration strategy, should leave its group and move to another one in order to complete the composition, which slows down the execution of composition request. This is not the case of the choreography strategy that is not affected by this limitation. Indeed, the composition request and the intermediate results are transmitted from a provider to another provider until the composition is completed and the final result is then returned to the requester. For example, in a scenario where the maximum number of hops is equal one, a requester who is client of a group owner, can delegate the composition request to that group owner in case it provides the next service in the composition, then the group owner will delegate the request to one of its clients.

Success ratio

Figure 7.4b presents the success ratio against the maximum number of hops between the service provider and the service client. We notice that the success ratio is greater in the Vannes city scenario than in the open area scenario. This can be explained by the fact that the mobility of users, in the Vannes city scenario, is restricted by buildings and other obstacles and also by the fact that the sport event follows a predetermined running path. In open area scenario, we do not have such a limitation. People can roam freely in the whole area. The orchestration-based composition strategy provides a better success ratio than the choreography-based composition regardless of the scenarios. In the Vannes city scenario, the orchestration-based strategy achieves a success ratio between 75 % and 87.5 % while the choreography-based strategy only provides between 60.6 % to 74.4 %. In the open area scenario, the orchestration-based strategy has a success ratio between 60.6 % and 74.4 %, whereas, the choreography-based strategy scores between 54 % and 62.6 % of success ratio.

These results are explained by the fact that partial results always return to the re-

quester when orchestration-based strategy is adopted. This is not the case in choreography-based composition strategy, where the partial results are passed between service providers. If we suppose that each node can call a service provider k hops away from it ($k \in \{1, 2, 3\}$ in our evaluation), then in the choreography-based strategy, the composition request could travel up to $k * |CR|$ hops away from the requester for all services in the composition request to get executed, where $|CR|$ is the number of services in the request CR. Whereas in the case of the orchestration-based strategy, invocation requests and intermediate results do not get further than k hops from the requester. This proves that the success ratio decreases when the number of hops increases.

Number of compositions executed by a node

Figure 7.4c shows the number of compositions executed by a single node against the maximum number of hops between the service provider and the service client. The four curves represent the same form of a nearly horizontal line. This shows the number of hops between the provider and the client has no effect on the number of compositions executed by a single node. The orchestration-based composition strategy has slightly more compositions per node than the choreography-based composition strategy. For example, in the Vannes city scenario, the orchestration-based composition strategy has 9.5 compositions per node and the choreography-based strategy has around 8.5. In the open area scenario, the orchestration-based strategy scores between 3.52 to 4.46 compositions per node, whereas the choreography-based strategy has only between 2.63 and 2.9 compositions. These results have the same explanation provided in the previous paragraph about the success ratio results. Moreover, we notice that the number of compositions per node, in the Vannes city scenario, is better than the number of compositions per node in the open area scenario. This is due to the fact that in the Vannes city scenario, there are buildings and obstacles that restrain the mobility of users and make them get closer to each other geographically which increases contact between them. Likewise, more service providers can be included in the invocation of the composite service, and more compositions can be completed successfully.

Number of nodes involved in a composition

Figure 7.4d shows the average of the number of nodes involved in an invocation of a composite service against the maximum number of hops between the service provider and the service client. The average number of nodes increases as the number of hops between the client and the provider increases. This can be explained by the fact that more service providers can be reached by increasing the maximum number of hops between the client and the provider, and thus more service providers can be enrolled in the invocation of the composite service. This can be observed by comparing Figure 7.7 and Figure 7.6. The former shows the distribution of the number of nodes per composition when the maximum number of hops between clients and providers is equal to 1 and the latter shows the same distribution when the maximum number of hops between clients and providers is equal to 2. In fact, almost all compositions enroll one

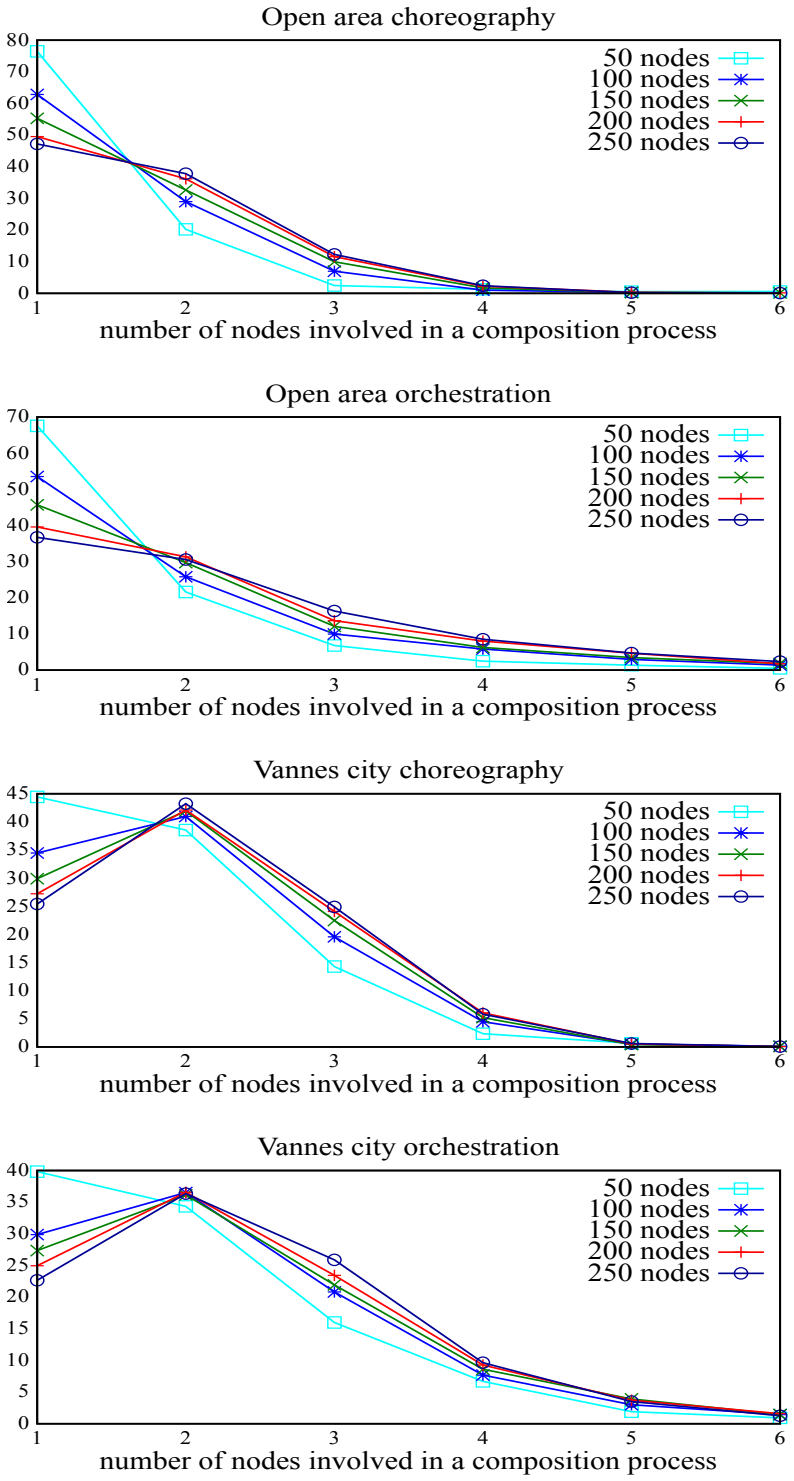


Figure 7.6 – Distribution of node number per composition with two hops away remote services.

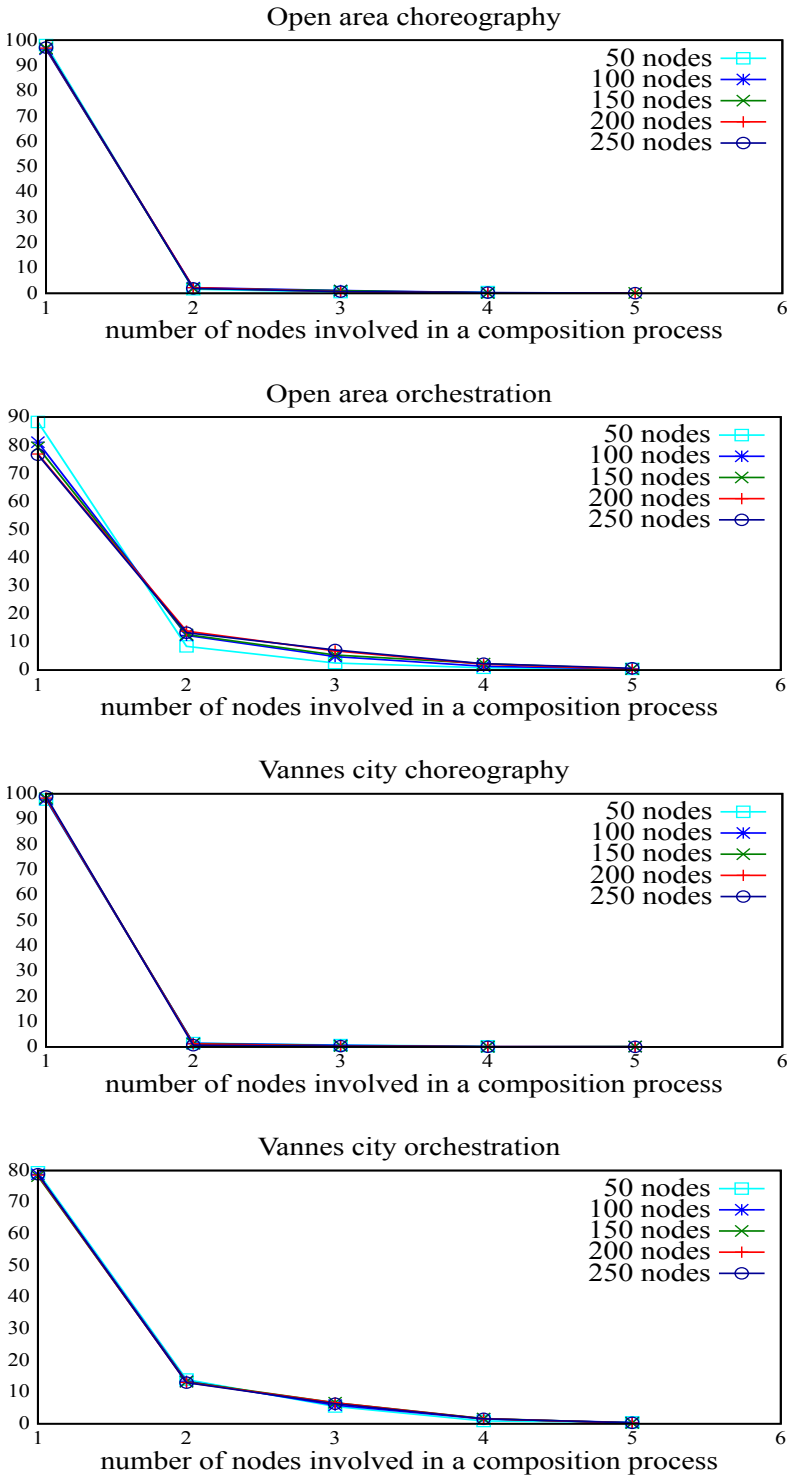


Figure 7.7 – Distribution of node number per composition with one hop away remote services.

single node when the maximum number of hops is equal to 1 due to the scarcity of service providers. Whereas, a considerable percentage of compositions enroll 2 or even 3 nodes when the maximum number of hops is equal to 2, because more service providers become accessible. On the other hand, the number of nodes involved increases less between 2-hops and 3-hops than between 1-hop and 2-hops. This is due to the important service redundancy in our scenarios. This is also confirmed by Figure 7.6. Indeed according to Figure 7.6, a large number of compositions are performed using 2 nodes or less. Consequently, when we have a significant redundancy of service providers, it is not necessary to increase the communication scopes of devices (*i.e.*, a number of hops equal to 2 is enough).

7.5 Conclusion

In this chapter, we compared the two strategies (orchestration and choreography) by analyzing the results from a series of emulations. For these emulations, we used the C3PO framework and the LEPTON emulator with both strategies while relying on a service discovery process that uses the time-based implementation of the utility function, presented in details in the Chapter 4. The results show that the orchestration strategy out-performs the choreography strategy especially when it comes to the success ratio. However, the choreography strategy is better in term of composition time, mainly because the choreography reduces significantly the number of messages needed to complete successfully a composition request. This difference in composition time is also due to the communication technology used in our evaluations, as we explained previously.

These results motivate future works that will try to investigate the possibility of switching from one strategy to another during the execution of the composition request. Indeed switching strategies can be interesting, when the estimated composition time for both strategies is very close. In this case, we can choose the orchestration over the choreography to maximize the success ratio. Moreover, when the estimated success ratio is relatively similar, it would be interesting to favor choreography over orchestration to shorten the composition time.

Chapter 8

Evaluation of the utility function

Contents

8.1 Introduction

In this chapter, we study the impact of both implementations of our utility function on the invocations of composite services in terms of success ratio and composition time by running several sets of emulations in which we vary some parameters, like the number of services per composition and the maximum number of hops between clients and providers, to determine which implementation provides the best performance for the service composition.

The reminder of the chapter is organized as follows. In Section 8.2, we detail the setup of our experiments. In Section 8.3, we present our results and we analyze them. In Section 8.4, we compare our results with those of the close works to this thesis, and finally in Section 8.5, we draw some conclusions.

8.2 Evaluation setup

8.2.1 General setup

Hereafter, we compare and evaluate the two implementations of our utility function. Similarly to the previous Chapter 7, we run a series of experiments using our service discovery and composition system and the lightweight emulator LEPTON. We use,

Parameter	Value
Open area size	500 m x 500 m
Composition request generation	between 2 and 5 min.
Evaluation Duration	1 hour
number of nodes	200
Advertisement period	10 seconds
Speed range	between 0.5 and 2m/s

Table 8.1 – Evaluation parameters.

Parameter	Value(s)			
	E_1	E_2	E_3	E_4
Number of hops	1, 2, 3	2	2	2
Number of local services	5	5	5	5
number of services per request	4	4	3,4,5,6	4
number of services per experience	20	10, 15, 20, 25, 30	20	20
remote service entry inactivity time threshold (seconds)	20	20	20	20, 40, 60, 80, 100

Table 8.2 – Parameters varying according to the evaluations.

also, the same two scenarios (500×500 open area, Vannes city) presented in the evaluation discussed in Chapter 7. We evaluate the two implementations of our utility function using both the orchestration-based strategy and the choreography-based strategy. Nodes involved in our emulations use the Wi-Fi Direct radio communication technology (23) based on the Android implementation with a radio range of 80 m. The speed of pedestrians is between 0.5 and 2 m/s and each experiment is repeated 10 times. These parameters and others are defined in the Table 8.1. Our evaluations focus on comparing both implementations based on success rate and composition time by varying parameters like the maximum number of hops between the service client and service provider, the remote service entry inactivity time threshold $T_{inactive}$ (defined in Chapter 4), the number of services per experience, and the number of services per composition.

8.2.2 Specific setup

To compare the two implementations of the utility function, we use 4 different evaluations as shown in Table 8.2:

- Evaluation E_1 studies the influence of the distance between the service client and the service provider in terms of the number of hops. In these experiments, we

vary the number of hops between 1 and 3. We use composition requests that contain 4 services that are not provided locally. The number of local services per node is 5 and number of services in these experiments is 20.

- In the evaluation E_2, We focus on studying the influence of number of services available per experience on the performance of our algorithms. In this set of emulations, we vary the number of services per experience between 10 and 30 services.
- In evaluation E_3, we study the effect of the variation of the number of services per composition request. Based on the results of this set of evaluations, we also compare how our two implementations of the utility function perform. We vary the number of services per composition between 3 and 6.
- In evaluation E_4, our goal is to study the effect of the remote service entry inactivity time threshold $T_{inactive}$ on both implementations. $T_{inactive}$ is used by the service registry. Indeed, when an entry, in the service registry, does not get updated for more than $T_{inactive}$, this entry is discarded and will not be considered for an execution of an invocation or a composition request. In this set of experiments, we vary $T_{inactive}$ between 20 seconds and 100 seconds.

8.3 Results and analysis

8.3.1 Success ratio

Hereafter, we study and compare the performances of both implementations of the utility function according to the success ratio.

Figure 8.1 shows the success ratio against the maximum number of hops between the service client and the service provider for both time-based and location-based implementations. The general trend shows that when the maximum number of hops between the service client and the service provider increases, the success rate decreases. This is due to the fact that when the number of hops between the client and the provider increases, the likelihood of connection disruption occurring between intermediate nodes increases due to mobility. Consequently, the request has lesser chances of reaching the service provider. Nonetheless, we notice that the success ratio of the location-based implementation decreases significantly less than the success ratio of the time-based implementation regardless of which composition strategy (orchestration *vs* choreography) is adopted. This can be explained by the fact that closer service providers are selected and enrolled in the invocation of the composite service, which helps decrease the likelihood of connection disruptions. Moreover, recoveries, in this case, are easier since proximity increases the probability of devices to meet each other again. In the scenario of Vannes city, the location-based implementation provides a success ratio between 84.39 % and 87.79 % using the orchestration-based strategy and between 76.70 % and 78.28 % using the choreography-based strategy. Whereas, the time-based implementation only offers a success ratio between 75.05 % and 87.51 % with the orchestration-based strategy,

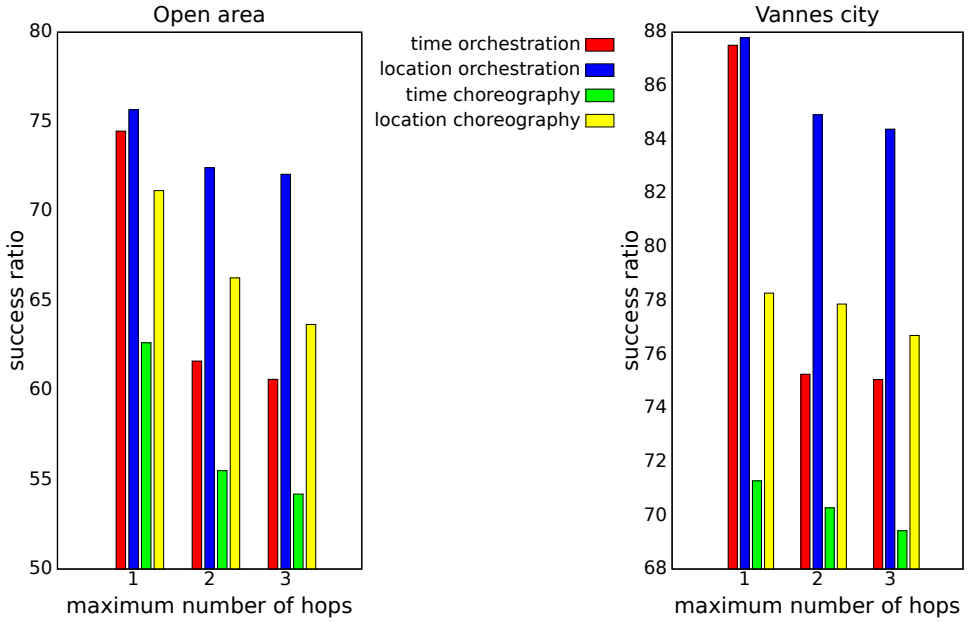


Figure 8.1 – Success ratio against the maximum number of hops between service client and service provider.

and between 69.42 % and 71.28 % with the choreography-based strategy. Our observation does not change in the open area scenario. The location-based implementation has a success ratio between 72.06 % and 75.66 % using the orchestration-based strategy and a success ratio between 63.66 % and 71.15 % using the choreography-based strategy. However, the time-based implementation only scores between 60.6 % and 74.47 % using the orchestration-based strategy and between 54.15 % and 62.64 % using the choreography-based strategy.

Figure 8.2 shows the success ratio against the number of services per composition. The success ratio decreases as the number of services per composition increases. We can explain that by the fact that the connection disruptions tend to increase as the number of services per composition increases. In our case, these failures are directly related to device mobility. Moreover, we notice that regardless of the strategy deployed, the location-based implementation achieves a better performance than the time-based implementation. In the open area scenario, the location-based implementation provides a success ratio between 35.21 % and 71.35 % against only 24.85 % to 63.95 % for the time-based implementation, using the orchestration-based strategy. Using the choreography-based strategy, the location-based implementation scores between 50.58 % and 62.86 %, whereas the time-based implementation has only a success ratio between 36.88 % and 54 %. In the Vannes city scenario, we notice that the performance gap between the two implementations becomes smaller, even though the location-based implementation maintains its advantage. This is due to the people mobility being restrained by a running path and by physical obstacles such as buildings. Indeed, the location-based

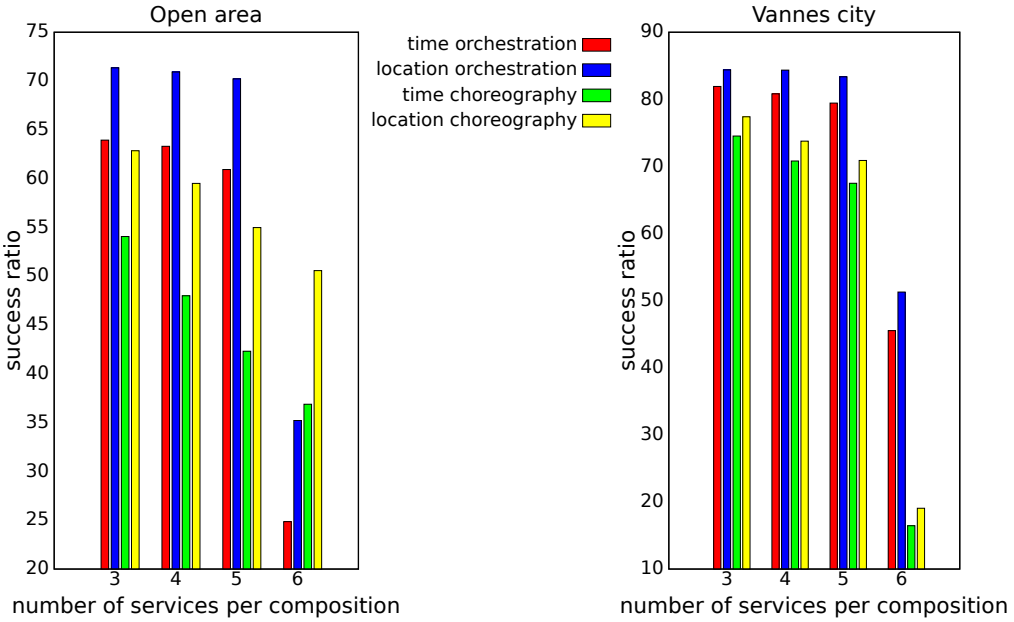


Figure 8.2 – Success ratio against the number of services per composition.

implementation scores between 51 % and 84 % using the orchestration strategy and between 19 % and 77 % using the choreography strategy. On the other hand, the time-based implementation has only a success ratio between 45 % and 82 % using the orchestration strategy, and between 16 % and 74.53 % using the choreography strategy.

Figure 8.3 presents the success ratio against the remote service entry inactivity time threshold $T_{inactive}$. As explained in Chapter 4, when an entry, in the service registry, does not get updated for more than $T_{inactive}$, this entry is discarded. We notice that the success ratio decreases as the $T_{iactive}$ increases. We can explain this by the fact that when $T_{inactive}$ is big, a given remote service entry can stay in the service registry even though the provider associated to it is already out of reach and no longer available probably due to mobility, which consequently affects in a negative way the invocation of the composite service. Moreover, we notice that the location-based implementation provides a better performance than the time-based implementation. Indeed, in the open area scenario, the location-based implementation scores between 33.93 % and 71.5 % in success ratio using the orchestration strategy, and between 12.34 % and 60.38 % using the choreography strategy. On the other hand, with the orchestration strategy, the time-based implementation has a slight advantage over the location-based implementation when $T_{inactive} = 20s$. However, the performance of the time-based implementation between 40 seconds and 100 seconds (between 19.25 % and 44.68 % of success ratio) is still worst than the performance of the location-based implementation. In the Vannes city scenario, the location-based implementation provides a success ratio between 49.15 % and 85.8 % using the orchestration strategy and between 22.37 % and 66.33 % using the choreography strategy. Whereas, the time-based function scores only

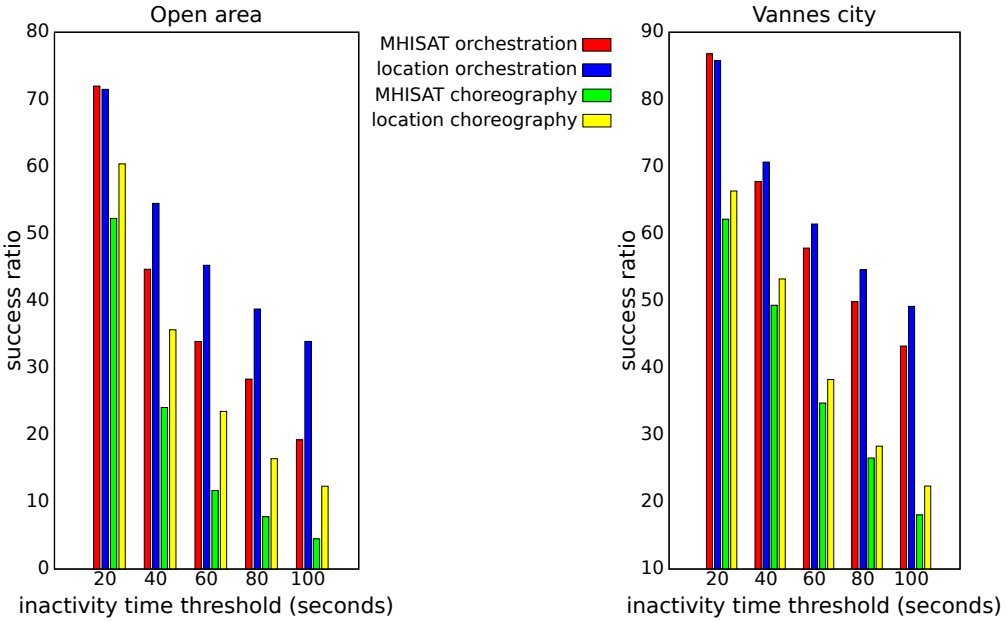


Figure 8.3 – Success ratio against remote service entry inactivity time threshold in service registries.

between 18 % and 62.14 % using the choreography strategy. When the orchestration strategy is used, the time-based implementation, again, has a slight advantage over the location based implementation when $T_{inactive} = 20s$. However, it loses that advantage between 20 seconds and 100 seconds, scoring only between 43 % and 67 %.

Figure 8.4 shows the success ratio against the number of services per experience. The general trend, regardless of the strategy or the implementation, shows that the success ratio stays stable until the number of services per experience reaches 25. At this point, we notice an important deterioration of performances in terms of success ratio. For example in the open area scenario, the location-based implementation provides around 70 % in success ratio between 10 and 20 services per experience. However, once the number of services per experience is over 25, the performances in terms of success ratio fall under 40 %. Indeed, this can be explained by the fact that when we increase the number of services per experience and we keep the number of instances deployed per device constant at the same time, the number of service instances for each service decreases which reduces the chances of invoking them successfully. Consequently, the probability for a composition request, to terminate successfully, decreases. On the other hand, we notice that the location-based implementation still provides a better success ratio than the time-based implementation especially in the open area scenario. Indeed, the location-based implementation scores between 34.07 % and 70.88 % when combined with the orchestration strategy and it, also, scores between between 13.69 % and 58.19 % when combined the choreography strategy. Whereas, the time-based implementation only provides a success ratio between 22.8 % and 62.9 % using the orchestration strategy

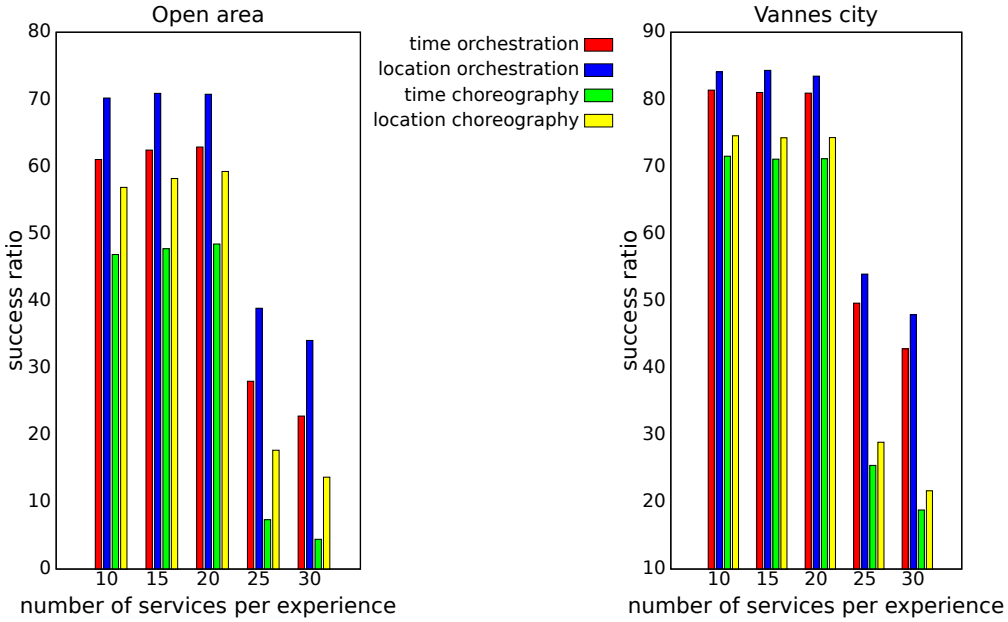


Figure 8.4 – Success ratio against number of services per experience.

and between 4 % and 48.44 % using the choreography strategy.

The conclusion that we can draw from the previous results, is that the location-based implementation has generally a better success ratio than the time-based implementation. This is due to the fact that the location-based implementation is focused on selecting the most geographically close and nearby providers unlike the time-based implementation which is focused on minimizing communication delays between the service client and the service provider.

.

8.3.2 Composition time

Hereafter, we compare both implementations from the perspective of their responsiveness in terms of the composition time that they provide for an invocation of a composite service. This comparison is based on the results from our evaluations that we previously detailed.

Figure 8.5 represents the median of composition time against the maximum number of hops between a service client and a service provider. We notice, generally, that when the maximum number of hops increases, the median of composition time increases, even though this increase is not very important between 2 and 3 hops. This small increase between 2 and 3 hops is explained by the fact that 2-hops-away providers already offer enough redundancy in service instances which makes service clients ignore the 3-hops-away providers. Figure 8.5 also shows that the time-based implementation provides a shorter composition time than the location-based implementation especially in

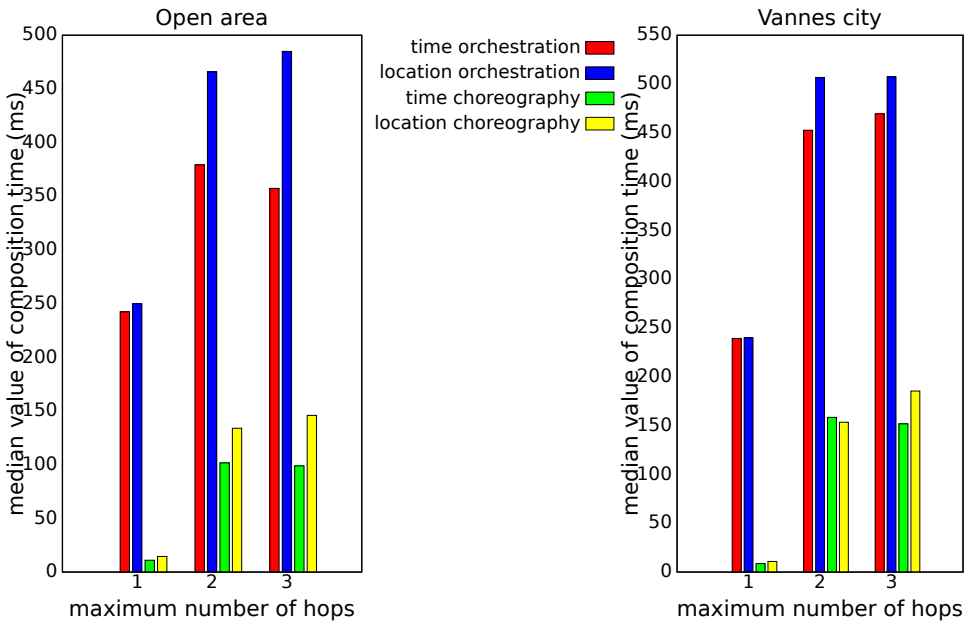


Figure 8.5 – Median of composition time against the maximum number of hops between the service client and service provider.

the open area scenario. In fact in this scenario, the time-based implementation achieves a median of composition time between 11.04 ms and 101.6 ms using the choreography strategy and between 242 ms and 379 ms using the orchestration strategy. Whereas, the location-based implementation provides a slower median of composition time between 14.56 ms and 145.94 ms using the choreography strategy and between 249.88 ms and 485 ms using the orchestration strategy. We notice, however, that in the Vannes city scenario, the time difference between the two implementations shrinks. As explained previously, this is due to the fact that users are restrained by buildings and by the running path which reduces the advantage of the time-based implementation over the location-based implementation, since the users are closer to each other in the Vannes city scenario than in the open area scenario. Nevertheless, the time-based implementation still preserves a slight advantage over the location-based implementation.

Figure 7.4a shows the decimal logarithm of the median value of the composition time against the number of services per composition. Regardless of the implementation or the strategy adopted, the median value of the composition time increases when the number of services per composition increases. This observation is expected since invoking an increasing number of services requires more service execution time and more messages exchanged during the invocation of the composite service. We also notice that the time-based implementation provides a slightly shorter composition time than the location-based implementation. In the Vannes city scenario and using the orchestration strategy, the time-based implementation has a median time between 395 ms and 150 seconds against a median time between 433 ms and 191 seconds for the

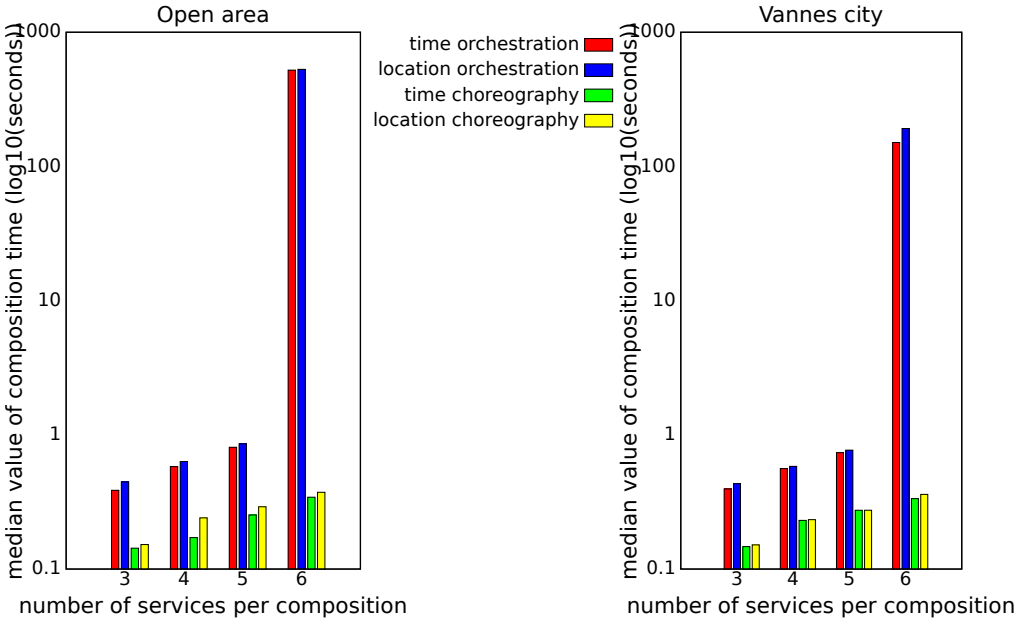


Figure 8.6 – Median of composition time against the number of services per composition.

location-based implementation. Moreover when the choreography strategy is adopted in the Vannes city scenario, the time-based implementation keeps its slight advantage by achieving a median time between 147 ms and 334 ms against a larger median time between 151 ms and 360 ms for the location-based implementation. We also notice that the median grows significantly for 6 services per composition when the orchestration strategy is used. The explanation is that it becomes more difficult to find the sixth service in the composition, knowing that we are choosing 6 services to be enrolled in a composition out of 20 available services, and that each device hosts 5 services. Moreover, we notice that the choreography always provides the fastest composition time. This difference in composition time between the two strategies has the same explanation provided in Chapter 7 for the composition time comparison.

Figure 8.7 shows the decimal logarithm of the median of composition time against the remote service entry inactivity time threshold $T_{inactive}$. We notice that the median value increases as the service entry inactivity time increases. This can be explained by the fact that a bigger $T_{inactive}$ allows service providers to go further away from the service client, while the service entries associated to them still indicate that these providers are reachable. Consequently, invoking a service provider would take a longer time. Figure 8.7 also shows that the time-based implementation achieves a shorter composition time than the location implementation. For the sake of example, in the open area scenario and using the orchestration strategy, the time-based implementation achieves a median time between 33 ms and 327 seconds, whereas the location based implementation terminates a composition request in a median time between 331 ms and

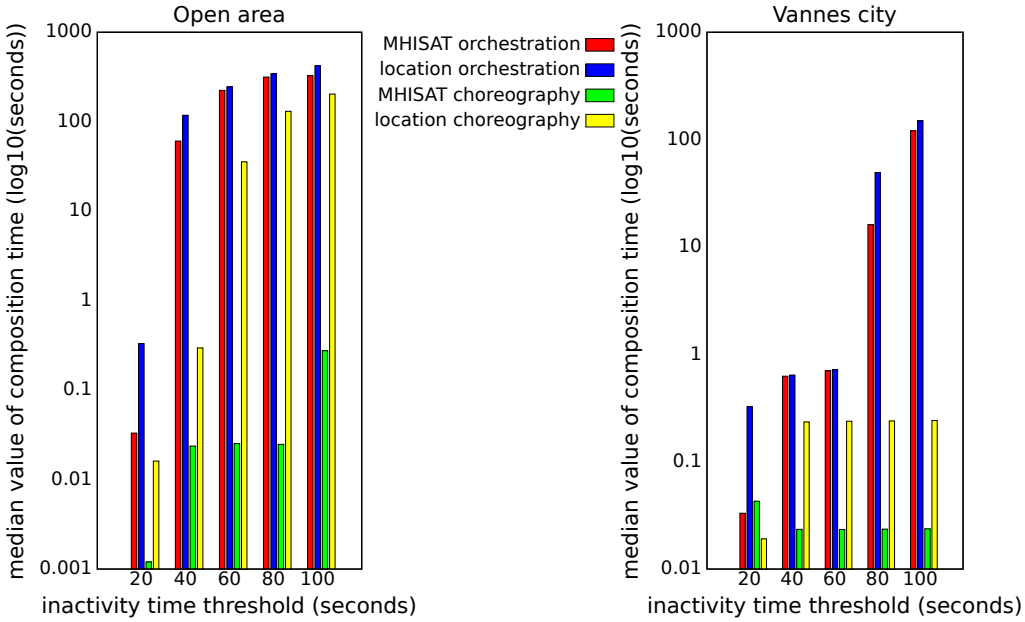


Figure 8.7 – Decimal logarithm of median of composition time against remote service entry inactivity time threshold in service registries.

419 seconds. In the same scenario and using the choreography strategy, the time-based implementation has by far the shortest composition time, comprised between 1.2 ms and 275.95 ms, against a slower composition time for the location-based implementation, comprised between 16.1 ms and 203 seconds .

Figure 8.8 presents the average of composition time against the number of services per experience. We notice that the general trend shows that the average of composition time increases as the number of services per experience increases. This can be explained by the fact that when the number of services increases, consequently, the number of instances per service decreases which reduces service redundancy, and thus the process of invocation takes more time. Moreover, we notice that when the orchestration strategy is adopted, there is no significant difference between the two implementations in terms of average composition time. However when the choreography strategy is adopted, we observe a clear advantage for the time-based implementation over the location-based implementation. Indeed, in the open area scenario, the time-based implementation has an average of composition time between 58 seconds and 110 seconds, whereas the location-based implementation provides a slower time between 190 seconds and 393 seconds. The same thing happens in the Vannes city scenario, where the time-based implementation has an average of composition time between 21 seconds and 61 seconds, against a slower average time between 50 seconds and 237 seconds for the location-based implementation.

We can conclude that the time-based implementation, generally, provides a shorter composition time than the location-based implementation. This demonstrates that the

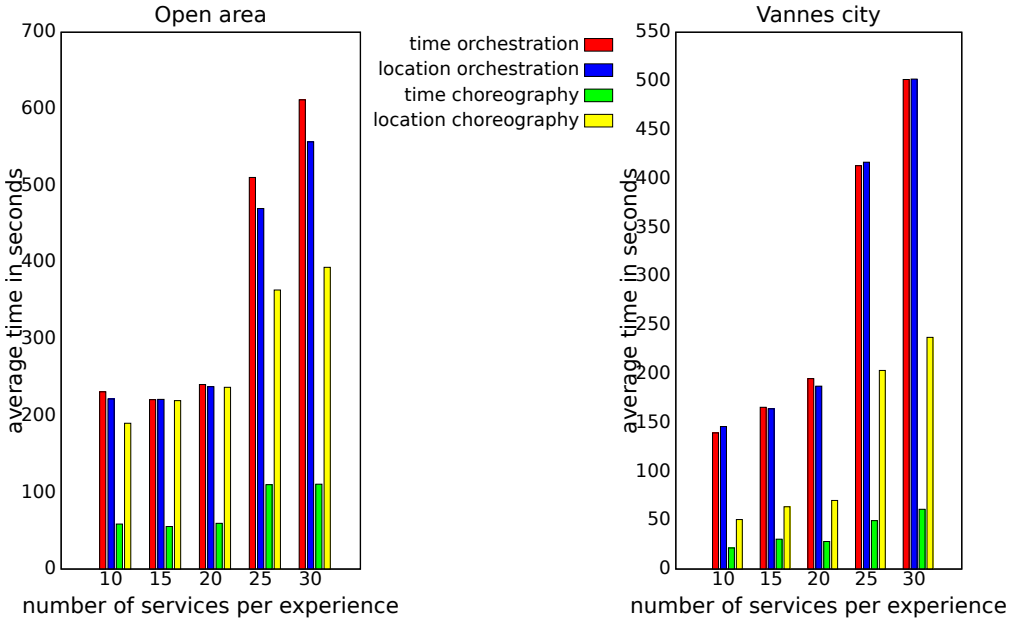


Figure 8.8 – Average of composition time against number of services per experience.

closest providers do not necessarily provide on average the best service composition time. Indeed, since it is based on transmission delays and on inter-advertisement reception times, the time-based implementation is more capable of selecting the most responsive service providers.

8.4 Comparison

We consider the results from the evaluation set E_1 for our comparison, since in E_1 we vary the number of hops between the client and the provider from 1 to 3, and we use 20 services per experiment. E_1 offers a similar context to the evaluations of Sadiq et al., since they also use multi-hops and provide the same number of services per experiment. We also consider the open area scenario due to its similarity with the scenarios used by Sadiq et al. and Groba et al. as Table 8.3 shows.

Sadiq et al. provide a success ratio around 55 %. Even though Sadiq et al. use a radio range 20 m longer than ours, our approach offers a success ratio between 54.14 % and 62.64 % using the choreography/time-based configuration which is the closest to the Sadiq et al. approach. This configuration also offers the quickest median of composition time (between 11.04 ms and 101.6 ms) among all the configurations provided by our approach. The median of composition time of this configuration is significantly shorter than Sadiq et al. median of composition time (less than 5 minutes). Moreover, even the slowest configuration in our approach (orchestration/location-based implementation) provides a shorter composition time (between 249.88 ms and 485 ms) than

	success ratio	composition time	speed	radio range	mobility	area
Sadiq et al. (104)	55 %	less than 5 min median	-	100 m	Levy	500x500 m^2
Groba et al. (48)	55 %-79 %	3-4 min average	1-2 m/s	250 m	Random Waypoint	1000x1000 m^2
choreography /time-based	54.15 %-62.64 %	11.04 ms-101.6 ms median (1 min 19 sec average)	0.5-2m/s	80 m	Levy	500x500 m^2
choreography /location-based	63.66 %-71.15 %	14.56 ms-145.94 ms median (3 min 4 sec average)	0.5-2m/s	80 m	Levy	500x500 m^2
orchestration /time-based	60.6 %-74.47 %	242 ms-379 ms median (3 m 24 sec average)	0.5-2m/s	80 m	Levy	500x500 m^2
orchestration /location-based	72.06 %-75.66 %	249.88 ms-485 ms median (3 min 21 sec average)	0.5-2m/s	80 m	Levy	500x500 m^2

Table 8.3 – Results from other works.

the composition time of Sadiq et al..

Groba et al. achieve a maximum success ratio of 79 %, which is higher than the success ratio of any of our configurations. This can be explained by the long radio range of 250 m used in their simulations. This radio range is more than 3 times longer than ours which is 80 m. Given our shorter range and the constraints imposed by Wi-Fi direct especially on the orchestration strategy (discussed in Chapter 7), our composition approach is in disadvantage compared to the approach of Groba et al., and therefore, it is difficult for us to reach a maximum of success ratio equal to 79 %. Nonetheless, Groba et al. approach only provides a minimum of success ratio around 55 %, which is less than the minimum provided by 3 of our configurations (choreography /location-based, orchestration/time-based and orchestration/location-based) except for the choreography/time-based configuration. This shows that our approach is more stable than the Groba et al. one and offers less fluctuations. Moreover, compositions, using Groba et al. approach, terminate in an average time between 3 and 4 minutes, which is the same in our case except for the choreography/time-based configuration that provides a substantially shorter average of composition time equal to 1 minutes and 19 seconds.

8.5 Conclusion

In this chapter, we compared our two implementations of the utility function (location-based and time-based) by studying their effects on the invocation of a composite (using the orchestration strategy and the choreography strategy both presented in details in Chapter 4) based on the results from a series of emulations. The results show that the location-based implementation has a better impact in terms of success ratio on the invocation of a composite service than the time-based implementation. However, the time-based implementation has overall a shorter composition time than the location-based implementation. Regardless of the strategy or the implementation of the utility function, we always obtain an acceptable success ratio and a relatively short composition time, given the hard conditions imposed by opportunistic networks. Moreover, even though, the performances, in terms of success ratio and composition time, deteriorate when there are 6 or more services to compose, this case is considered rare. Indeed, we should first find 6 or more compatible services that can be combined together in a service composition, which constitutes a complicated task in opportunistic networks, given the lack of service redundancy and availability. In the next chapter, we study the impact of our distributed cache manager on service composition and its potential to improve the composition time and the success ratio.

Chapter 9

Evaluation of the distributed cache performance

Contents

9.1 Introduction

In this chapter, we study the performances of our distributed cache manager (DCM), and its potential to improve and optimize service compositions. We evaluate its efficiency of providing composition results in terms of time saving compared to the execution time of a simple composition request.

Nevertheless, we were not able to test the performances of our proactive service precomputing manager (PSP). Indeed, the evaluation of the PSP should focus on measuring how much the list of composition requests, generated by the PSP, satisfies the user preferences. This evaluation should also measure, based on the user feedbacks, how much the user experience has improved thanks to the role of PSP in populating the cache with composition results by identifying and executing composition requests in advance based on the user preferences, expressed in the form of keyword list. This kind of evaluation requires a user interface that presents the available services to the user (atomic or composite), and allows him to invoke these services. The same interface should allow the user to rate the list of composition requests, generated by PSP, based on how much this list reflects his preferences. We did not implement this kind of interface during this thesis.

We organize the reminder of the chapter as follows. In Section 9.2, we detail our

Parameter	Value
Scenario	Vannes city
Number of topics/composition request	4
Evaluation Duration	1 hour
Result message size	1500 byte
Number of nodes per experience	100
Speed range	between 0.5 and 2m/s
Number of messages per node (number of message per topic per node)	200 (50)

Table 9.1 – DCM evaluation parameters.

evaluation environment. In Section 9.3, we present and analyze our results and finally in Section 9.4, we draw conclusions.

9.2 Evaluation setup

Our DCM exploits the cache memories of nodes in the network. DCM considers the available cache memories as a distributed storage space. Likewise and instead of repeating composition requests, a given node performs a look up in its local cache, using the index table provided by the DCM, to verify if the results are already present. DCM relies on replicating composition results between nodes using a content-based approach based on a publish/subscribe communication mode. In our implementation of DCM, we rely on the C3PO framework "Topic" mechanism for publish/subscribe communications. Indeed, DCM associates a topic to each composition a given node would like to perform. Likewise and by subscribing to these topics, this node can collect and replicate results from other nodes in its own local cache.

To evaluate this idea, we ran a series of experiments to compare common operations, that allow to manipulate results collected from other nodes, to an invocation of a composite service in terms of temporal performance. In these series of experiments, we use 100 nodes per experiment using the Vannes city scenario, we define 4 topics that represent 4 composition requests, and we initially provide each node with 200 results (50 for each topic). These parameters are detailed in Table 9.1. Our goal is to compare the cost of manipulating the cache memory to find composition results and the cost of executing a composition request on the same inputs that gave us the already existing results in the cache.

The operations that we take into consideration in these experiments are:

- *Access:* it consists of reading one single composition result from the cache.

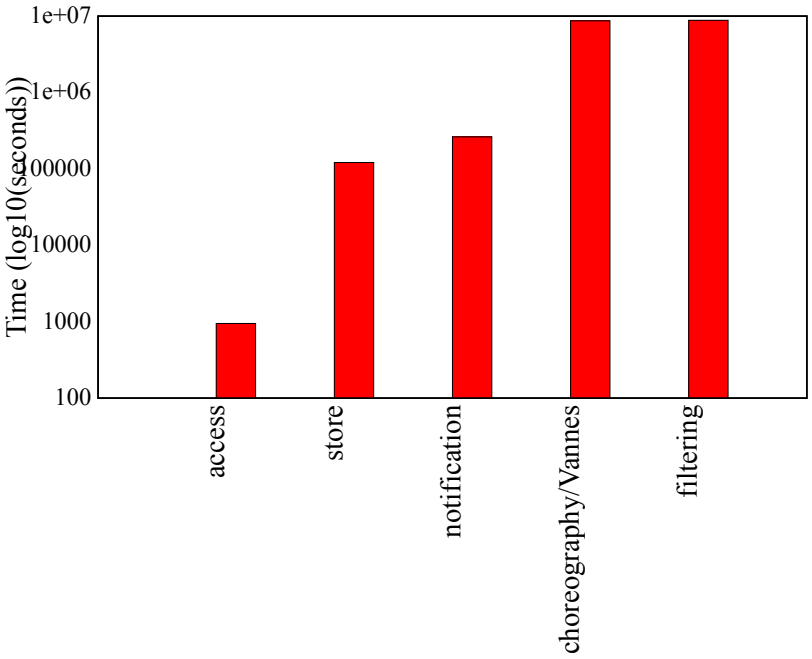


Figure 9.1 – Median time of different data operations.

- *Store:* it consists of writing one single composition result into the cache.
- *Notification:* it is the time elapsed between the reception of a composition result from a remote node and the notification the application layer receives that announces that a new result is available.
- *Filtering:* it consists of filtering all the results that belongs to a certain topic/composition request.

9.3 Results

These results, shown in Figure 9.1, allow us to compare the median time of each operation, detailed previously, against the median of composition time of the fastest configuration (choreography/1 hop between the service client and the service provider) according to the results from Chapter 7. Figure 9.1 shows that the access operation is the fastest one with a median time of 950 nanoseconds, the store operation is the second fastest with a median time of 121470 nanoseconds to terminate, followed by the notification operation that finishes in 262706 nanoseconds. All these three operations are significantly faster than the median of the composition time provided by the fastest configuration which allows a composition request to terminate in 8.68 milliseconds.

Even though, the filtering operation takes 8.81 milliseconds to finish which is slightly slower than the composition time, we must take into consideration that in our scenario,

we filter 50 composition results out of 200 which is quite reasonable considering the fact that we obtain 50 results rather than 1 single result from the execution of a composition request in slightly the same time.

These results clearly demonstrate that accessing a result already available in the cache costs significantly less than producing the same available result through an execution of a composition request. This highlights the importance of our DCM in making data access faster, which helps improve the user experience.

9.4 Conclusion

In this chapter, we compared the performances provided by the DCM against those of a simple execution of a composition request, to see how much improvement we can provide using this DCM. The comparison was between the time needed for local cache memory operations, used by DCM, and the minimum time needed to finish a composition request. We concluded from the results that the distributed cache manager can provide an important time saving by exploiting results from other nodes stored in the local cache memory. This evaluation can be extended by studying the effect of the DCM on the average or the median of composition time, and on the composition success ratio. It would be also interesting to evaluate how much resource saving the DCM can provide in terms of processing power, battery lifetime and bandwidth consumption.

Chapter 10

Conclusions and future works

Contents

10.1 Summary of the contribution

We start in Chapter 1 by introducing the general context of this thesis, the constraints we should face, and the objectives we want to achieve. We propose an opportunistic computing approach based on service-oriented computing (SOC) for opportunistic networks. We focus on devising solutions to perform service discovery, service selection and service composition. Given the challenges and the difficulties imposed by opportunistic networks such as the absence of end-to-end path, unpredictable node mobility and the long transmission delays, conventional SOC solutions proof to be inefficient in this type of environment.

Our work led us to propose a service discovery and composition system, a distributed cache manager (DCM) and a proactive service precomputing manager (PSP). The system and the two managers provide five main contributions:

- A discovery approach that allows different nodes running our system to advertise and to discover services.
- A utility function that rates discovered services based on informations collected from the network. The utility function is also used to select which provider to invoke for an invocation or a composition request. We provide two implementations for this utility function (time-based/location-based).
- Two composition strategies (orchestration/choreography) that dictate how a composition request should be executed.
- A distributed storage space for composition results.

- A mechanism for proactively triggering composition requests in advance based on the user preferences.

Our service discovery and composition system provides a service discovery approach that is directory-less where we do not suppose the existence of a set of special nodes playing the role of a directory that aggregates the descriptions of the services available in the network. Instead, each node has its own service registry that contains the description of services discovered by the node itself. Our approach is also proactive, where service providers periodically announce their local services preferably by piggy-packing service descriptions in device advertisement messages to avoid overloading the network.

Service clients listen to these advertisements and upon reception, they rate the service providers who sent the advertisements using one of the two implementations of our utility function. This utility function relies on two criteria in order to rate a service provider, namely the invocation success ratio and the invocation time. Each implementation of the utility function provides a formula, that takes into consideration these two criteria, in order to assess service providers.

Our system also provides two composition strategies: orchestration and choreography. By using the orchestration, the composition request stays with the requester that is in charge of selecting and invoking services. Whereas with choreography, the composition request is passed from one provider to another until the composition is finished. We also suggest that it is possible to switch from one strategy to the other depending on which has access to a better provider: the current node holding the composition request or another node in its neighbourhood.

PSP automates the execution of composition requests using a composition graph based on service descriptions, and tries to identify the most suitable compositions based on an interest profile provided by the user/developer. PSP proactively triggers composition requests to populate the cache in advance with composition results in order to reduce composition delays and improve the user experience.

DCM allows nodes to share and replicate composition results using content-based communication. DCM relies on an index table to look for existing composition results in the local cache before starting invocations of composite services in order to save time and resources.

Our evaluations show that the orchestration has generally a better success ratio than the choreography. In contrast, the choreography has a shorter composition time than the orchestration. Consequently, we argue that a hybrid strategy can be relevant in two different cases. The first case is when both strategies have a very close success ratio. In this case choosing choreography could reduce composition time. The second case is when both strategies have a very close composition time. In this case, choosing orchestration could help increase the probability of composition success. Our evaluations also show that the location-based implementation of the utility function has a better impact on the success ratio than the time-based implementation, and vice-versa when it comes to composition time. Regardless of the strategy or the utility function implementation, we obtain a reasonable composition time and an interesting success

ratio, given the kind of networks we consider. Moreover, we demonstrate the positive impact of our distributed cache manager on time saving by exploiting local caches to replicate composition results.

10.2 Future works

We conclude this thesis by identifying and discussing several future works and open research issues. These identified issues do not represent an exhaustive list related to the topic of this thesis, but rather the main directions that can be explored in the future. Indeed, there are many other related issues that can be explored or rather improved in our contribution such as security and privacy. In the current implementation, message are exchanged without encryption or checksum. Consequently, they can be corrupted by malicious nodes. Moreover, node authentication is not taken into consideration in the invocations of simple and composite services. Hereafter, we present the main perspectives for our future works.

Hybrid composition strategy As the evaluation results have shown, the orchestration-based composition strategy has a better performance than the choreography-based composition strategy in terms of success ratio, and vice-versa in terms of composition time. It can be argued that these two strategies can be seen as complementary. Indeed, according to the network topology and to the distribution of services on the devices forming the network, sometimes it is better to use the choreography and sometimes it is better to use the orchestration. Our goal will be to find the best way to combine these two strategies and switch from one to the other in the middle of an invocation of a composite service, in a way that optimizes both the success ratio and the composition time. We will also use the same evaluation sets described in Chapters 7 and 8 to compare this hybrid strategy performances with the performances of the other two strategies.

Using semantic service description The current service description that we use is a syntactic one. We believe that by adopting semantics in the future, we can obtain a more detailed and more expressive description for services. Indeed, semantics have been applied before in the context of mobile ad hoc networks (2; 11). Furthermore, we think that a semantic description will help the proactive service precomputing manager build a better and more accurate service graph. Consequently, the PSP will be more efficient in finding the composition requests that match the user interest profile.

Real world experiments Conducting real world experiments is a difficult task to carry out mainly because of the high cost of a large number of devices and the difficulties of gathering volunteers to participate in the experiments. These hurdles led us to rely on emulation using LEPTON. Even though in our experiments we used real code coupled with both mobility model and real mobility traces, emulation does not take into

account all the details involved in wireless communication. For instance our evaluations ignored almost all the details of the physical and data-link layers and only took in consideration the radio range. Thus, the idea of real time experiments is still important even with a limited number of nodes in order to have a more comprehensive idea about the performance of our service discovery and composition system.

Perceived quality of service The general impression of users is important to determine whether our contribution is efficient. It would be interesting to evaluate the perceived quality of service based on users feedbacks especially when the PSP is used to determine the compositions that reflect the user preferences, and to populate the cache with composition results in advance. For this purpose, we should develop an interface that display to the user the list of services (simple or composite) that he can invoke manually. The same interface should allow the user to rate the list of compositions generated by PSP in order to assess how much PSP is efficient with satisfying the user preferences.

Bibliography

[1] Darp agent markup language and ontology inference layer.

[2] G.R. Karpagam A. Bhuvaneswari. Semantic web service discovery for mobile web services. *International Journal of Business Intelligence and Data Mining*, 13(1-3):95–107, January 2018.

[3] Unai Aguilera and Diego Lopez de Ipina. An architecture for automatic service composition in manet using a distributed service graph. *Future Generation Computer Systems*, 34:176 – 189, 2014.

[4] Atif Alamri, Mohamad Eid, Saddik, and Abdulmotaleb El. Classification of the State-of-the-Art Dynamic Web Services Composition Techniques. *International Journal of Web and Grid Services*, 2(2):148–166, September 2006.

[5] Géraud Allard, Pascale Minet, Dang-Quan Nguyen, and Nirisha Shrestha. Evaluation of the energy consumption in manet. In Thomas Kunz and S. S. Ravi, editors, *Ad-Hoc, Mobile, and Wireless Networks*, pages 170–183, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[6] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*. IBM, 2003.

[7] Assaf Arkin, Sid Askary, Scott Fordin, and W. Wolfgang Jekel et al. Web Service Choreography Interface (WSCI) 1.0. Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems, 2002.

[8] Valerio Arnaboldi, Marco Conti, and Franca Delmastro. Cameo: A novel context-aware middleware for opportunistic mobile social networks. *Pervasive and Mobile Computing*, 11:148 – 167, 2014.

[9] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[10] Maël Auzias, Y. Mahéo, and Frédéric Raimbault. Coap over bp for a delay-tolerant internet of things. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 118–123, Aug 2015.

[11] Soheyb Ayad, Okba Kazar, Benharkat Aïcha-Nabila, and labib sadek terrissa. An optimised semantic web services discovery in MANET. *International Journal of Communication Networks and Distributed Systems*, 2017.

[12] Nilanjan Banerjee, Mark D. Corner, Don Towsley, and Brian N. Levine. Relays, base stations, and meshes: Enhancing mobile networks with infrastructure. In *In Proceedings of ACM Mobicom*, 2008.

[13] Luciano Baresi and Liliana Pasquale. Adaptive goals for self-adaptive service compositions. In *2010 IEEE International Conference on Web Services*, pages 353–360, July 2010.

[14] Abdulkader Benchi, Pascale Launay, and Frédéric Guidec. A P2P Tuple Space Implementation for Disconnected MANETs. *Peer-to-Peer Networking and Applications*, 8(1):87–102, January 2015.

[15] Abdulkader Benchi, Pascale Launay, and Frédéric Guidec. JMS for Opportunistic Networks. *Ad Hoc Networks*, 25(part B):359–369, February 2015.

[16] Mahdi Bennara, Michaël Mrissa, and Youssef Amghar. An approach for composing restful linked services on the web. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 977–982, New York, NY, USA, 2014. ACM.

[17] Martin Bichler and Kwei Jay Lin. Service-oriented computing. *Computer*, 39(3):99–101, March 2006.

[18] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[19] Chiara Boldrini, Marco Conti, Franca Delmastro, and Andrea Passarella. Context- and social-aware middleware for opportunistic networks. *Journal of Network and Computer Applications*, 33(5):525 – 541, 2010. Middleware Trends for Network Applications.

[20] Chiara Boldrini, Marco Conti, Jacopo Jacopini, and Andrea Passarella. HiBOP: a History Based Routing Protocol for Opportunistic Networks. In Marco Conti, editor, *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007)*, pages 1–12, Helsinky, Finland, 2007. IEEE CS.

[21] Chiara Boldrini, Marco Conti, and Andrea Passarella. Contentplace: Social-aware data dissemination in opportunistic networks. In *Proceedings of the 11th International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '08, pages 203–210, New York, NY, USA, 2008. ACM.

[22] Celeste Campo, Mario Munoz, José Carlos Perea, Andrés Mann, and Carlos Garcia-Rubio. Pdp and gsdl: a new service discovery middleware to support spontaneous interactions in pervasive systems. In *Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 178–182, March 2005.

[23] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. Device-to-device communications with wi-fi direct: overview and experimentation. *IEEE Wireless Communications*, 20(3):96–104, June 2013.

[24] Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure. In *Revised Papers from the NSF Workshop on Developing an Infrastructure for Mobile and Wireless Systems*, IMWS '01, pages 59–68, London, UK, UK, 2002. Springer-Verlag.

[25] Fabio Casati, Ski Ilnicki, Li-Jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. eflow: a platform for developing and managing composite e-services. In *Research Challenges, 2000. Proceedings. Academia/Industry Working Conference on*, pages 341–348, 2000.

[26] Dipanjan Chakraborty, Anupam Joshi, Tim Finin, and Yelena Yesha. Service composition for mobile environments. *Mobile Networks and Applications*, 10(4):435–451, 2005.

[27] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, and Tim Finin. Gsd: a novel group-based service discovery protocol for manets. In *4th International Workshop on Mobile and Wireless Communications Network*, pages 140–144, 2002.

[28] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1. W3c note, World Wide Web Consortium, March 2001.

[29] Marco Conti, Franca Delmastro, and Andrea Passarella. Mobile service platforms based on opportunistic computing: The scampi project. *ERCIM News*, 2013, 2013.

[30] Marco Conti, Silvia Giordano, Martin May, and Andrea Passarella. From opportunistic networks to opportunistic computing. *IEEE Communications Magazine*, 48(9):126–139, September 2010.

[31] Microsoft Corporation. Upnp device architecture 1.1, 2008.

[32] Scott Corson and Joseph Macker. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. RFC 2501 (Informational), January 1999.

[33] Paolo Costa, Mirco Musolesi, Cecilia Mascolo, and Gian Pietro Picco. Adaptive content-based routing for delay-tolerant mobile ad hoc networks. 09 2018.

[34] Elizabeth M. Daly and Mads Haahr. Social network analysis for routing in disconnected delay-tolerant manets. In *Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '07, pages 32–40, New York, NY, USA, 2007. ACM.

[35] Teodoro De Giorgio, Gianluca Ripa, and Maurilio Zuccalà. An approach to enable replacement of soap services and rest services in lightweight processes. In Florian Daniel and Federico Michele Facca, editors, *Current Trends in Web Engineering*, pages 338–346, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[36] Lucia Del Prete and Licia Capra. Reliable Discovery and Selection of Composite Services in Mobile Environments. In *12th Enterprise Distributed Object Computing Conference (EDOC'08)*, pages 171–180, Munich, Germany, September 2008. IEEE.

[37] Shuiguang Deng, Longtao Huang, Javid Taheri, Jianwei Yin, MengChu Zhou, and Albert Y. Zomaya. Mobility-aware service composition in mobile communities. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(3):555–568, March 2017.

[38] Michael Doering, Sven Lahde, Johannes Morgenroth, and Lars Wolf. IBR-DTN: an efficient implementation for embedded systems. In *Proceedings of the 3rd ACM workshop on Challenged networks*, pages 117–120. ACM, September 2008.

[39] Bluetooth Specification Part E. Service discovery protocol (sdp), 1999.

[40] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall Professional Technical Reference, Upper Saddle River, NJ, 2005.

[41] Shawna Evans. *BizTalk: For Starters*. CreateSpace Independent Publishing Platform, USA, 2017.

[42] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM.

[43] Roy Fielding, James Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Rfc 2616, hypertext transfer protocol – http/1.1, 1999.

[44] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[45] Renato Fileto, Ling Liu, Calton Pu, Eduardo Delgado Assad, and Claudia Bauzer Medeiros. Poesia: An ontological workflow approach for composing web services in agriculture. *The VLDB Journal*, 12(4):352–367, Nov 2003.

[46] Martin Garriga, Cristian Mateos, Andres Flores, Alejandra Cechich, and Alejandro Zunino. RESTful service composition at a glance: A survey. *Journal of Network and Computer Applications*, 60:32–53, jan 2016.

[47] Eduardo Goncalves da Silva, Luis Ferreira Pires, and Marten J. van Sinderen. *Supporting Dynamic Service Composition at Runtime based on End-user Requirements*, pages –. CEUR Workshop Proceedings 540. CEUR Workshop Proceedings, 11 2009.

[48] Christin Groba and Siobhán Clarke. Opportunistic composition of sequentially-connected services in mobile computing environments. In *2011 IEEE International Conference on Web Services*, pages 17–24, July 2011.

[49] Christin Groba and Siobhán Clarke. Opportunistic service composition in dynamic ad hoc environments. *IEEE Transactions on Services Computing*, 7(4):642–653, Oct 2014.

[50] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. *From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices*, pages 97–129. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[51] Erik Guttman, Charles E. Perkins, and James Kempf. Service templates and service: Schemes. *RFC*, 2609:1–33, 1999.

[52] Julien Haillot and Frédéric Guidec. A Protocol for Content-Based Communication in Disconnected Mobile Ad Hoc Networks. *Journal of Mobile Information Systems*, 6(2):123–154, 2010.

[53] Sumi Helal, Nitin Desai, Verun Verma, and Choonhwa Lee. Konark - a service discovery and delivery protocol for ad-hoc networks. In *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.*, volume 3, pages 2107–2113 vol.3, March 2003.

[54] Tristan Henderson, David Kotz, and Ilya Abyzov. The changing usage of a mature campus-wide wireless network. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, MobiCom '04, pages 187–201, New York, NY, USA, 2004. ACM.

[55] Chung-Ming Huang, Kun-chan Lan, and Chang-Zhou Tsai. A survey of opportunistic networks. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 1672–1677. IEEE, March 2008.

[56] Pan Hui, Jon Crowcroft, and Eiko Yoneki. Bubble rap: Social-based forwarding in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 10(11):1576–1589, Nov 2011.

[57] Noha Ibrahim and Frédéric Le Mouël. A Survey on Service Composition Middleware in Pervasive Environments. *International Journal of Computer Science Issues*, 1:1–12, August 2009.

[58] Sushant Jain, Rahul C. Shah, Waylon Brunette, Gaetano Borriello, and Sumit Roy. Exploiting mobility for energy efficient data collection in wireless sensor networks. *Mob. Netw. Appl.*, 11(3):327–339, June 2006.

[59] Wei jen Hsu and Ahmed Helmy. On modeling user associations in wireless lan traces on university campuses. In *2006 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pages 1–9, April 2006.

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

[61] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC 4728 (Experimental), February 2007.

[62] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. *SIGARCH Comput. Archit. News*, 30(5):96–107, October 2002.

[63] Swaroop Kalasapur, Mohan Kumar, and Behrooz Shirazi. Seamless Service Composition (SeSCo) in Pervasive Environments. In *1st ACM International Workshop on Multimedia Service Composition (MSC'05)*, pages 11–20, Hilton, Singapore, 2005. ACM.

[64] Michael Klein, Birgitta Konig-Ries, and Philipp Obreiter. Service rings - a semantic overlay for service discovery in ad hoc networks. In *14th International Workshop on Database and Expert Systems Applications, 2003. Proceedings.*, pages 180–185, Sept 2003.

[65] Janine Kniess, Orlando Loques, and Celio V. N. Albuquerque. Location aware discovery service and selection protocol in cooperative mobile wireless ad hoc networks. In *IEEE INFOCOM Workshops 2009*, pages 1–2, April 2009.

[66] Ulas 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, 2004.

[67] Frédérique Laforest, Nicolas Le Sommer, Stéphane Frénot, François De Corbière, Yves Mahéo, Pascale Launay, Christophe Gravier, Julien Subercaze, Damien Reimert, Étienne Brodu, Idris Daikh, Nicolas Phelippeau, Xavier Adam, Frédéric Guidec, and Stéphane Grumbach. C3PO: a Spontaneous and Ephemeral Social Networking Framework for a Collaborative Creation and Publishing of Multimedia Contents. In *International Conference on Selected Topics in Mobile and Wireless Networking (MoWNet 2014)*, pages 129–134, Rome, Italy, September 2014. Elsevier.

[68] Markus Lanthaler and Christian Gutl. A semantic description language for restful data services to combat semaphobia. In *5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011)*, pages 47–53, May 2011.

[69] Nicolas Le Sommer and Sihem Ben Sassi. Location-based service discovery and delivery in opportunistic networks. In *Networks (ICN), 2010 Ninth International Conference on*, pages 179–184. IEEE, 2010.

[70] Nicolas Le Sommer and Yves Mahéo. OLFServ: an Opportunistic and Location-Aware Forwarding Protocol for Service Delivery in Disconnected MANETs. In Xpert Publishing Services, editor, *Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (Ubicomm 2011)*, pages 115–122, Lisbon, Portugal, November 2011.

[71] Nicolas Le Sommer, Romeo Said, and Yves Mahéo. A proxy-based model for service provision in opportunistic networks. In *Proceedings of the 6th international workshop on Middleware for pervasive and ad-hoc computing*, pages 7–12. ACM, 2008.

[72] Jérémie Leguay, Timur Friedman, and Vania Conan. Evaluating mobility pattern space routing for dtns. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–10, April 2006.

[73] Angel Lagares Lemos, Florian Daniel, and Boualem Benatallah. Web Service Composition: A Survey of Techniques and Tools. *ACM Computing Surveys*, 48(3):1–41, February 2016.

[74] Fei Li, Katharina Rasch, Hong-Linh Truong, Rassul Ayani, and Schahram Dustdar. Proactive service discovery in pervasive environments. In *Proceedings of the 7th ACM International Conference on Pervasive Services (ICPS)*, pages 126–133, 2010. QC 20111207.

[75] Li Li, Wu Chou, Tao Cai, and Zhe Wang. Hyperlink pipeline: Lightweight service composition for users. In *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 1, pages 509–514, Nov 2013.

[76] Anders Lindgren, Avri Doria, and Olov Schelen. Probabilistic Routing in Intermittently Connected Networks. In *Proceedings of the 1st International Workshop on Service Assurance with Partial and Intermittent Resources (SAPIR 2004)*, Fortaleza, Brazil, August 2004.

[77] Yves Mahéo and Romeo Said. Service invocation over content-based communication in disconnected mobile ad hoc networks. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 503–510, April 2010.

[78] Ali Makke, Nicolas Le Sommer, and Yves Mahéo. TAO: A Time-Aware Opportunistic Routing Protocol for Service Invocation in Intermittently Connected Networks. In *8th International Conference on Wireless and Mobile Communications (ICWMC 2012)*, pages 118–123, Venice, Italy, June 2012. Xpert Publishing Services.

[79] Ali Makke, Yves Mahéo, and Nicolas Le Sommer. Towards opportunistic service provisioning in intermittently connected hybrid networks. In *Networking and Distributed Computing (ICNDC), 2013 Fourth International Conference on*, pages 28–32, December 2013.

[80] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. Owl-s: Semantic markup for web services. Internet [http://www.w3.org/Submission/2004/SUBM- OWL-S-20041122/], 2004.

[81] Alessandro Mei, Giacomo Morabito, Paolo Santi, and Julinda Stefa. Social-aware stateless forwarding in pocket switched networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 251–255, April 2011.

[82] Sun Microsystems. Jini architecture specification. 1999.

[83] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[84] Vinicius F.S. Mota, Felipe D. Cunha, Daniel F. Macedo, Jose M.S. Nogueira, and Antonio A.F. Loureiro. Protocols, mobility models and tools in opportunistic networks: A survey. *Computer Communications*, 48:5 – 19, 2014. Opportunistic networks.

[85] Enrico Motta, John Domingue, Liliana Cabral, and Mauro Gaspari. Irs ii: a framework and infrastructure for semantic web services. In *2nd International Semantic Web Conference (ISWC2003)*, Sundial Resort, Sanibel Island, Florida, USA, October 2003. 10.1007/b14287 DOI ISBN 0302-9743.

[86] Michael Mrissa, Lionel Médini, and Jean-Paul Jamont. Semantic discovery and invocation of functionalities for the web of things. In *2014 IEEE 23rd International WETICE Conference*, pages 281–286, June 2014.

[87] Michael Mrissa, Lionel Médini, Jean-Paul Jamont, Nicolas Le Sommer, and Jérôme Laplace. Towards An Avatar Architecture for the Web of Things. Research report, Université Lyon 1 - Claude Bernard, January 2015.

[88] Mirco Musolesi, Stephen Hailes, and Cecilia Mascolo. Adaptive routing for intermittently connected mobile ad hoc networks. In *Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*, pages 183–189, June 2005.

[89] Hemanth Narra, Yufei Cheng, Egemen K. Çetinkaya, Justin P. Rohrer, and James P. G. Sterbenz. Destination-sequenced distance vector (dsdv) routing protocol implementation in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 439–446, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[90] Hoang Anh Nguyen, Silvia Giordano, and Alessandro Puiatti. Probabilistic routing protocol for intermittently connected mobile ad hoc network (propicman). In

2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, pages 1–6, June 2007.

[91] Michael Nidd. Service discovery in deapspace. *IEEE Personal Communications*, 8(4):39–45, Aug 2001.

[92] Panagiotis Pantazopoulos, Ioannis Stavrakakis, Andrea Passarella, and Marco Conti. Efficient social-aware content placement in opportunistic networks. In *Wireless On-demand Network Systems and Services (WONS), 2010 Seventh International Conference on*, pages 17–24, February 2010.

[93] Cesare Pautasso. Composing restful services with jopera. In Alexandre Bergel and Johan Fabry, editors, *Software Composition*, pages 142–159, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[94] Cesare Pautasso. Restful web service composition with bpel for rest. *Data and Knowledge Engineering*, 68(9):851 – 866, 2009. Sixth International Conference on Business Process Management (BPM 2008) Five selected and extended papers.

[95] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, October 2003.

[96] Luciana Pelusi, Andrea Passarella, and Marco Conti. Opportunistic Networking: Data Forwarding in Disconnected Mobile Ad Hoc Networks. *IEEE Communications Magazine*, 44(11):134–141, November 2006.

[97] Yu Yen Peng, Shang Pin Ma, and Jonathan Lee. Rest2soap: A framework to integrate soap services and restful services. In *2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–4, Jan 2009.

[98] Charles Perkins, Elizabeth Royer, and Samir R. Das. RFC 3561 Ad hoc On-Demand Distance Vector (AODV) Routing. Technical report, 2003.

[99] Charles E. Perkins and John Veizades. Service Location Protocol. RFC 2165, June 1997.

[100] Randall Perrey and Mark Lycett. Service-oriented architecture. In *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.*, pages 116–119, Jan 2003.

[101] Mikko Pitkänen, Teemu Kärkkäinen, Jörg Ott, Marco Conti, Andrea Passarella, Silvia Giordano, Daniele Puccinelli, Franck Legendre, Sacha Trifunovic, Karin Hummel, Martin May, Nidhi Hegde, and Thrasyvoulos Spyropoulos. Scampi: Service platform for social aware mobile and pervasive computing. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 7–12, New York, NY, USA, 2012. ACM.

[102] Jinghai Rao and Xiaomeng Su. A Survey of Automated Web Service Composition Method. In Jorge Cardoso and Amit Sheth, editors, *Semantic Web Services and Web Process Composition*, pages 43–54, San Diego, CA, USA, July 2004. Springer.

[103] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, Seong Joon Kim, and Song Chong. On the Levy-Walk Nature of Human Mobility. *IEEE/ACM Transactions on Networking*, 19(3):630–643, June 2011.

[104] Umair Sadiq, Mohan Kumar, Andrea Passarella, and Marco Conti. Service Composition in Opportunistic Networks: A Load and Mobility Aware Solution. *IEEE Transactions on Computers*, 84(8):2308–2322, August 2015.

[105] Françoise Sailhan and Valérie Issarny. Scalable Service Discovery for MANET. In *International Conference on Pervasive Computing and Communications : PerCom 2005*, pages 235–244, Kawai Island, Hawaii, United States, 2005.

[106] Natasa Sarafijanovic-Djukic, Michal Piórkowski, and Matthias Grossglauser. Island hopping: Efficient mobility-assisted forwarding in partitioned networks. In *Proceedings of the Third Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON 2006, September 25-28, 2006, Reston, VA, USA*, pages 226–235, 2006.

[107] Gregor Schiele, Christian Becker, and Kurt Rothermel. Energy-efficient cluster-based service discovery for ubiquitous computing. In *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*, EW 11, New York, NY, USA, 2004. ACM.

[108] James Scott, Jon Crowcroft, Pan Hui, and Christophe Diot. Haggle: a Networking Architecture Designed Around Mobile Users. In *WONS 2006 : Third Annual Conference on Wireless On-demand Network Systems and Services*, pages 78–86, Les Ménuires (France), January 2006. INRIA, INSA Lyon, Alcatel, IFIP. http://citi.insa-lyon.fr/wons2006/index.html.

[109] Keith Scott and Scott C. Burleigh. Bundle Protocol Specification. RFC 5050, November 2007.

[110] K. Seada and A. Helmy. Rendezvous regions: a scalable architecture for service location and data-centric storage in large-scale wireless networks. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pages 218–, April 2004.

[111] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.

[112] Tara Small and Zygmunt J. Haas. The shared wireless infostation model: A new ad hoc networking paradigm (or where there is a whale, there is a way). In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, MobiHoc '03, pages 233–244, New York, NY, USA, 2003. ACM.

[113] Thrasyvoulos Spyropoulos, K. Psounis, and C.S. Raghavendra. Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on*, pages 79–85, March 2007.

[114] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and Wait: an Efficient Routing Scheme for Intermittently Connected Mobile Networks. In *2005 ACM SIGCOMM workshop on Delay-tolerant networking (WDTN'05)*, pages 252–259, Philadelphia, PA, USA, 2005. ACM.

[115] Haiyan Sun, Xiaodong Wang, Bin Zhou, and Peng Zou. Research and implementation of dynamic web services composition. In Xingming Zhou, Ming Xu, Stefan Jähnichen, and Jiannong Cao, editors, *Advanced Parallel Processing Technologies*, pages 457–466, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[116] Spyropoulos Thrasyvoulos and Picu Andreea. *Opportunistic Routing*, chapter 11, pages 419–452. Wiley-Blackwell, 2013.

[117] Leigh Torgerson, Scott C. Burleigh, Howard Weiss, Adrian J. Hooke, Kevin Fall, Dr. Vinton G. Cerf, Keith Scott, and Robert C. Durst. Delay-Tolerant Networking Architecture. RFC 4838, April 2007.

[118] Vladimir Tosic, Babak Esfandiari, Bernard Pagurek, and Kruti Patel. On requirements for ontologies in management of web services. In Christoph Bussler, Richard Hull, Sheila McIlraith, Maria E. Orlowska, Barbara Pernici, and Jian Yang, editors, *Web Services, E-Business, and the Semantic Web*, pages 237–247, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[119] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, 2000.

[120] Jianping Wang. Exploiting Mobility Prediction for Dependable Service Composition in Wireless Mobile Ad Hoc Networks. *IEEE Transactions on Services Computing*, 4(1):44–55, January 2011.

[121] Zijian Wang, Eyuphan Bulut, and Boleslaw K. Szymanski. Service discovery for delay tolerant networks. In *2010 IEEE Globecom Workshops*, pages 136–141, Dec 2010.

[122] Mark Weiser. Hot topics-ubiquitous computing. *Computer*, 26(10):71–72, Oct 1993.

[123] UDDI.org white paper. Uddi technical white paper. September 2000.

[124] Dan Yu and Hui Li. On the definition of ad hoc network connectivity. In *International Conference on Communication Technology Proceedings, 2003. ICCT 2003.*, volume 2, pages 990–994 vol.2, April 2003.

[125] Zhangbing Zhou, Jiabei Xu, Zhenjiang Zhang, Fei Lei, and Wei Fang. Energy-efficient optimization for concurrent compositions of wsn services. *IEEE Access*, 5:19994–20008, 2017.

Publications

[1] Fadhlallah Baklouti, Nicolas Le Sommer, and Yves Mahéo, "Opportunistic service composition in pervasive networks", in 2017 Wireless Days, Porto, Portugal, March 29-31, 2017 (, 2017), pp. 227--229.
[2] Fadhlallah Baklouti, Nicolas Le Sommer, and Yves Mahéo, "Choreography-based vs orchestra-tion-based service composition in opportunistic networks", in 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob) (, 2017), pp. 1-8.
[3] Fadhlallah Baklouti, Nicolas Le Sommer, and Yves Mahéo, "Performing Service Composition in Opportunistic Networks", in 2019 Wireless Days, London, UK, April 24-26 (, 2019) (accepted).

Titre : Composition de services dans les réseaux opportunistes

Mot clés : réseaux opportunistes, communication opportuniste, découverte de services, composition de services.

Resumé : Cette thèse s'inscrit dans le domaine de l'informatique ambiante et de l'Internet des objets, et considère des réseaux qui peuvent se former spontanément et qui sont composés d'équipements fixes ou mobiles. Ces équipements peuvent être connectés à une infrastructure grâce à des interfaces de communication telles que 4G et Wi-Fi, et communiquer à travers celle-ci. Ces équipements peuvent aussi communiquer de gré-à-gré grâce à des interfaces permettant des communications ad hoc.

Des ruptures de connectivité peuvent apparaître dans le réseau d'une manière fréquente et imprévisible du fait de la faible portée des interfaces sans fil fonctionnant en mode ad hoc et de la mobilité de certains équipements. Ces ruptures de connectivité peuvent s'avérer problématiques dès lors que ces équipements souhaitent accéder à des ressources offertes par d'autres équipements, ou mettre eux mêmes à disposition de ces derniers des ressources.

L'informatique opportuniste étend le principe des communications opportunistes en proposant d'exposer les ressources à travers des services et d'accéder à ces services par des techniques et des protocoles de communication opportunistes qui mettent en œuvre le principe du "store, carry and forward".

Dans cette thèse, nous nous intéressons à la composition de services afin de pouvoir combiner les services élémentaires offerts par les équipements et ainsi proposer aux utilisateurs des services de plus haut niveau et plus riches. La composition de services est une tâche complexe dans l'informatique opportuniste car il est nécessaire de sélectionner judicieusement les fournisseurs de services afin de réduire au maximum les échecs et les délais de transmission des messages de services qui sont induits par les ruptures de connectivité, et ainsi pouvoir offrir une certaine qualité de service aux utilisateurs.

Dans cette thèse, nous proposons une solution pour composer les services en utilisant deux stratégies, à savoir la chorégraphie et l'orchestration. Cette solution repose en outre sur une fonction d'utilité qui permet de sélectionner les fournisseurs de services selon deux critères : le taux et le temps de transmission des messages de découverte et d'invocation de services. Nous proposons également une version améliorée exploitant un cache distribué de données et un mécanisme proactif de composition de services exploitant les profils d'intérêt des utilisateurs.

Nous avons évalué ce processus de composition en utilisant 2 scénarios différents. Les résultats obtenus dans ces scénarios réalistes montrent qu'il est possible de composer des services dans un temps raisonnable dans le type de réseaux que nous considérons.

Title : Service composition in opportunistic networks

Keywords : opportunistic network, opportunistic computing, service discovery, service composition.

Abstract : This thesis is related to the domain of Ubiquitous computing and Internet of Things (IoT), and focuses on networks that can be formed spontaneously by mobile or fixed devices. These devices are usually connected to an infrastructure, using communication interfaces such as 4G and Wi-Fi, and communicate with each other through this infrastructure. These devices can also communicate in a peer-to-peer mode using interfaces that implement ad hoc communication.

Connection disruptions may occur in the network frequently and unpredictably due to the short radio range of communication interfaces and to the mobility of certain nodes.These connection disruptions can be problematic when a given device tries to access remote resources provided by other devices, or when it tries to offer its own resources to these ones.

Opportunistic computing extends the paradigm of opportunistic networking by abstracting local resources as services accessible remotely using the protocols of opportunistic networking that implement the "store, carry and forward" principle.

In this thesis, we focus on service composition in order to combine elementary services and offer new, rich and high level composite services to users. Service composition can be a very difficult task to perform in opportunistic networks. Indeed, service composition requires an efficient selection process of service providers to reduce failures and transmission delays, caused by connection disruptions, in order to provide users with a certain quality of service.

In this thesis, we propose a solution to compose services using two strategies : orchestration and choreography. This solution also relies on a utility function that selects service providers based on two criteria : transmission time and transmission success of invocation and discovery messages. We also propose an optimized version of our solution that exploits a distributed cache of data and a proactive service composition mechanism based on the user interest profile.

We evaluated our composition solution using two different scenarios. The results show that it is possible to compose services in a reasonable amount of time in opportunistic networks.