A distributed approach to design federations of agents in an ecosystem of services

José Francisco Cervantes Alvarez

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

préparée dans le cadre d'une cotutelle entre la Communauté Université de Grenoble Alpes et le CINVESTAV Unidad Guadalajara

Spécialité : **Informatique**

Arrêté ministérial : le 6 janvier 2005 - 7 août 2006

Présentée par

José Francisco CERVANTES ALVAREZ

Thèse dirigée par **Michel OCCELLO et Félix RAMOS**
et codirigée par **Jean-Paul JAMONT**

préparée au sein **Laboratoire LCIS**
dans **l'Ecole Doctorale MSTII**

Une Approche Décentralisée pour la Conception de Fédérations d'Agents dans un ÉcoSystème de Services

Thèse soutenue publiquement le **11/07/2016**,
devant le jury composé de :

M. René Mandiau
Pr. Université de Valenciennes et du Hainaut-Cambrésis, Président

M. Olivier Boissier
Pr. Ecole Nationale Supérieure des Mines de Saint-Étienne, Rapporteur

M. Laurent Vercouter
Pr. Institut National Des Sciences Appliquées de Rouen, Rapporteur

M. Michel Occello
Pr. Université Grenoble Alpes, Directeur de thèse

M. Félix Ramos
Pr. CINVESTAV Unidad Guadalajara, Directeur de thèse

M. Jean-Paul Jamont
Dr. Université Grenoble Alpes, Co-Directeur de thèse

Résumé

L'objectif de cette thèse est la conception des fédérations d'agents dans un écosystème omniprésent de service hybride. Une approche distribuée basée sur une métaphore sociale d'un écosystème naturel est proposé dans le but de construire des fédérations d'agent ouvert, où les agents interagissent les uns avec les autres pour atteindre leurs objectifs grâce à la composition dynamique et l'adaptation des services. Dans cette approche, chaque agent régit un appareil mobile sans fil. Chaque agent au sein de l'écosystème est intéressée, et ses comportements sont basés uniquement sur des tâches d'interaction locales. L'interaction et la coopération entre les agents sont guidés par un ensemble de normes sociales établies par les membres de l'écosystème. Deux approches basées sur la satisfaction de contraintes ont été conçues pour comparer les résultats des stratégies d'adaptation de service avec les propositions existantes basées sur la reconfiguration des services à partir de zéro. La première est basée sur un modèle distribué du problème de satisfaction de contraintes. La seconde est une extension du modèle distribué pour aborder la dynamique des environnements ouverts. Les simulations montrent les performances des algorithmes concernant la consommation de temps et le nombre de messages requis pour la composition et l'adaptation des services.

Mots clés: systèmes multi-agents, écosystème de services, composition de service, satisfaction de contraintes distribués.

Abstract

The objective of this thesis is the design of federations of agents in a pervasive hybrid service ecosystem. A distributed approach based on a social metaphor of a natural ecosystem is proposed with the aim of building open agent federations, where agents interact with each other to achieve their goals through dynamic composition and adaptation of services. In this approach, each agent governs a wireless mobile device. Each agent within the ecosystem is self-interested, and its behaviors are based only on local interaction tasks. The interaction and cooperation among agents are guided by a set of social norms, established by members of the ecosystem. Two approaches based on constraint satisfaction were designed to compare the results of strategies of service adaptation with existing proposals based on the reconfiguration of services from scratch. The first is based on a distributed model of the constraint satisfaction problem. The second is an extension of the distributed model to address the dynamics of open environments. The simulations show the performance of the algorithms regarding the consumption of time and the number of messages required for the composition and adaptation of services.

Keywords: Multi-agent systems, service ecosystem, service composition, distributed constraint satisfaction.

Remerciements

Avant toutes choses, je tiens à remercier les membres de mon jury. Merci à Olivier Boissier et à Laurent Vercouter qui m'ont fait l'honneur d'être rapporteurs de ma thèse. Je remercie également René Mandiau pour l'honneur qu'il me fait d'être dans mon jury de thèse. Leurs remarques m'ont permis d'envisager mon travail sous un autre angle. Pour tout cela je les remercie.

Je tiens aussi à remercier les nombreuses personnes qui ont été présentes tout au long de ma thèse et qui m'ont permis de la conclure. Il sera très difficile pour moi de remercier tout le monde.

D'abord, je tiens à remercier grandement les directeurs et superviseurs de cette thèse; Michel Occello, Felix Ramos et Jean-Paul Jamont, merci pour toute votre aide. Je suis ravi d'avoir eu l'occasion de travailler avec vous, vous avez toujours été là pour me soutenir et me donner des conseils lors de la préparation de cette thèse.

Les discussions que j'ai pu avoir durant les réunions d'équipe et en dehors m'ont beaucoup apporté: Antonio, Sonia, Rodolfo, Clément, Daniel et Luis Fernando trouvez dans ces quelques lignes l'expression de mes remerciements.

Je remercie aussi toutes les personnes avec qui j'ai partagé mes études et notamment ces années de thèse : Ji Young Moon qui m'a permis de m'échapper de temps en temps dans la France profonde; Kevin, Milena et Emanuel pour leur gentillesse; GianFranco pour son inévitable rituel du midi et ses animations dans la cafétéria, qui m'ont soutenu jusqu'au bout.

Je remercie ma femme Mary et mes enfants Valeria, Paco et Miguel Angel; vous m'avez soutenu tout au long de cette aventure et vous avez fait preuve de beaucoup de compréhension les jours et nuits où je n'étais pas avec vous.

Encore un grand merci à tous pour m'avoir accompagné jusque ce jour mémorable.

Acknowledgements

Above all, I thank the members of my jury. Thank you Olivier Laurent Boissier and Vercouter who have done me the honor rapporteurs to be my thesis. I also thank René for Mandiau the honor he made me be on my dissertation committee. Their remarks allowed me to consider my work from another angle. For all that I thank them.

I also want to thank the many people who have been present throughout my thesis and that allowed me to conclude. It will be very difficult for me to thank everyone.

First, I would like to greatly thank the managers and supervisors this thesis; Michel Occello, Felix Ramos, and Jean Paul Jamont, thank you for all your help. I am delighted to have had the opportunity to work with you, you have always been there to support me and give me advice in the preparation of this thesis.

The discussions I have had during team meetings and outside brought me a lot Antonio, Sonia, Rodolfo, Clement, Daniel Luis Fernando, there are in these lines the expression of my thanks.

I also thank all the people with whom I shared my studies including the years of thesis: Ji Young Moon which allowed me to escape from time to time in provincial France; Kevin, Milena and Emanuel for their kindness; GianFranco to its inevitable ritual noon and entertainment in the cafeteria, who supported me all the way.

I thank my wife Mary and my children Valeria, Paco, and Miguel Angel; you supported me throughout this adventure and you have shown a great understanding of the days and nights when I was not with you.

Another big thank you to all for having accompanied me until this day memorable.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Today, the boundaries of pervasive computing have changed significantly. These are evolving from closed systems with dedicated infrastructure to open dynamic systems with no dedicated infrastructure, such as dedicated servers and reliable communications networks [1] [2] [3] [4]. This is due to the increasing number of devices in our daily lives that can acquire and process data, and communicate with each other [5]. Moreover, new mobile devices are revolutionizing the way we access and distribute services, and wireless network paradigms are driving the concept of computing toward delay-tolerant networking [6] [7]. The fact that mobile devices can be constantly network-connected and the increasing availability of online services are prompting users to demand a transparent integration of services with their lifestyles and daily activities. This is leading service-oriented computing and networks toward everyday objects such as cars, refrigerators, televisions, washing machines and medical devices, and the concept of pervasive service is becoming a reality [8]. From the Service Oriented Architecture (SOA) perspective, a service is defined as a self-contained unit software that performs a specific task [9]. A pervasive service can be defined as a user-centered service that is available anywhere and anytime.

We distinguish between two types of pervasive services: software and hardware services. The former are services that require only data and processing to generate a result that meets their goals; classical services usually fall into this category. Hardware services, on other hand, have physical dependencies. For example, hardware services require the availability of specific devices such as temperature sensors, Global Positioning System (GPS) and radiators in their immediate physical environment in order to meet their goals

(i.e. the service has physical dependencies on its physical environment). Most research work has focused on software services, and there has been little work in the domain of hardware services.

Solutions based on SOAs are suitable to deploy services in closed and slightly dynamic environments such as buildings with dedicated infrastructure that must always be available. It is necessary, however, to consider environments where services must be available anywhere and anytime, and where SOAs-based solutions are therefore not suitable; these environments call for a pervasive computing approach that focuses on users. Examples of such environments include places where the user does not stay for a long time or that the user does not frequent very often, such as airports, hospitals, shopping malls, or cafes. The question is how devices can keep hardware services available while users move among such environments. In general the strategy is to take advantage of distributed resources such as sensors, actuators, memory, and communication interfaces available in the form of services in the user's vicinity.

1.2 Problem Description

From a pervasive computing perspective, the hardware services must be user-centered and remain available while users perform their daily activities regardless of their displacement among environments. These services must always be available by means of adapting to the resources available on other devices in the users' vicinity. The following scenario helps to convey the idea of such a world.

Miguel is driving his car from his home to his work in the city of Guadalajara. He wants to get to work as soon as possible. For this Miguel is assisted by his mobile device to choose the best route. Internally the device uses its GPS and interacts regularly with systems that provide the status of road traffic in the city. Suddenly the car suffers a mechanical failure, and Miguel has a serious road accident. His car automatically reports the event to the emergency systems. It triggers the cooperation between fire brigades, police officers and the nearest ambulances to provide assistance to Miguel. Meanwhile, the traffic control system modifies its signaling so that the help arrives soon. Concurrently all systems are coordinated to continue assisting other drivers who also need to reach their destination as quickly as possible. Once Miguel is in the ambulance, bio-medical devices transmit all information about his health status to the hospital. Again, the traffic control system coordinates its signaling so that the ambulance can reach the hospital quickly.

This scenario embodies many open issues in pervasive services. In order to achieve the above vision, this dissertation addresses three main problems concerned with deploying pervasive hardware services:

- **Interaction mechanisms.** Organizational models for deploying pervasive services are commonly based on interaction mechanisms that are not suitable for open environments or that need previous knowledge of the global model of the environment, for example, the semantic chemistry approach proposed by Viroli et al. [10]. This increases the amount of a priori knowledge required to deploy hardware services. However, mobile devices usually have limited resources due to non-functional requirements such as energy limitations. Many organizational structures such as hierarchies [11] [12] have been used to deploy pervasive services; however, most of them provide a monolithic, hard-wired, rigid structure that limits the interaction to previously defined and fixed patterns. Pervasive services need to adapt to dynamic and unforeseen situations, requiring flexible and dynamic organizations to adjust the system's course of action in the pursuit of their objectives.

- **Dynamic service composition.** In pervasive computing, devices can be mobile, and services can have physical dependencies. For example, in order to provide a hypothetical medical service, a set of devices with certain resources must be in the user's vicinity. This underscores the complexity of service composition, in which no single device has all the required resources and logic to succeed. Classical composition techniques such as SOA-based mechanisms [13] [14]) are not suitable to handle hardware service composition; one of the main reasons is that SOA-based solutions require dedicated infrastructure. Hence, suitable approaches are required.

- **Service Adaptation.** The primary objective of pervasive systems is to provide services that are required by the user despite variations in his or her environment; this is achieved by means of hardware service adaptation. Adaptation aims to make the pervasive system capable of fulfilling requirements in spite of changes in its environment [15] [16]. Diverse techniques for service adaptation have been used, such as dynamic service selection and dynamic coordination pattern selection. However, most of these assume that resources are always available in closed environments, and they only consider the dynamic degradation of service quality [17]; hence, a suitable mechanism is required for the adaptation of hardware services in open environments.

1.3 Thesis Objectives

Pervasive systems based on the web and mobile devices are currently garnering considerable attention due to their ability to provide users with services for data processing and information. These pervasive systems naturally have an interest in multi-agent systems because of certain features they offer, such as the autonomy of the agents, and their interaction mechanisms. The use of multi-agent systems can provide service providers with greater autonomy in making decisions; it is important because the autonomy improves how these types of systems deal with dynamic scenarios. The thesis that we defend is that it is possible to create ecosystems based on multi-agent systems capable of providing hardware services that are adaptable to user needs and changes in their environment. For this, we propose to model hardware, software and resources as agents that provide pervasive services within an ecosystem, using a social metaphor, for the purpose of creating flexible organizations, unlike current pervasive hardware services that rely on organizations of limited flexibility that restrict users' mobility to closed environments.

We are interested more particularly in ecosystems comprising mobile devices with few capabilities in terms of the number and types of services they can provide. Such devices require the collaboration of other members of the ecosystem, in order to achieve the objectives for which they were designed. For this, we propose designing a framework for supporting the dynamic composition of hardware services, with the goal of permitting service composition in several environments through the use of resources that are close to the user. The dynamic composition of services is common in SOA; however most of the mechanisms used are not suitable for the deployment of hardware services in environments.

It is unlikely to assume that once the system has created a service, user needs and environmental status will not change. Such composite services need to be able to adapt to changes and fulfill the new requirements with little or no participation from the user. For this, we propose an approach to adapt the functionality of hardware services provided by agents that take advantage of the available resources and services of other agents in the user's vicinity.

1.4 Thesis Contributions

This dissertation proposes the modeling of an open and dynamic organizational model of hardware services by means of an ecosystem metaphor . Members of the ecosystem interact

with each other in order to achieve their objectives. Social ecosystem metaphors have not often been used to model and constrain behaviors of pervasive hardware services. In this domain, the contributions of this dissertation are the following:

- Definition of an ecosystem of hardware services based on a social metaphor in order to support the dynamism and openness of the environment. This aims to provide a flexible organization for pervasive systems that will make it possible to offer services in environments where there may be communication failures or intermittent availability of resources. In addition, the ecosystem provides the framework for the development of mechanisms for service composition and adaptation.

- Definition of an interaction model based on social obligations that regulates the behavior of agents in the ecosystem and their interaction with each other. Here, the concept of adaptation on obligations is introduced, a concept that has not been addressed before in the state of the art.

- Definition of acts for an agent communication language based on obligations endowed with social semantics that explicitly considers system autonomy and is suitable for open systems.

In the domain of services-oriented computing, the following contributions are made:

- For the purpose of modeling the distributed and dynamic composition process of pervasive services while the agents' autonomy is preserved, we define the composition process as a dynamic and distributed constraint satisfaction problem.

- With the aim of carrying out the composition and adaptation of services, through solving constraint satisfaction problems, we have proposed decentralized and asynchronous mechanisms.

1.5 Outline

This thesis is organized into three parts as described in Figure 1.

The general objective of the first part is to introduce the reader to the context within which the research was conducted. This general objective is addressed in chapters two and three. The particular objective of chapter 2 is to introduce the reader in our vision of a pervasive system as an services' ecosystem and it provides the basic concepts that

will be useful to understand and appreciate this document. Chapter 3 gives the reader an overview of the various mechanisms for service composition, beginning with the classical approaches and moving on to those based on constraint satisfaction.

The general objective of the second part, comprising chapters four, five and six, is to present and describe in detail our proposal and to address the issues described in the first chapter of this manuscript. The particular objectives of each of the chapters are as follows: Chapter 4 introduces a new ecosystem approach based on a pervasive social metaphor for providing services; each metaphor element is described and the chapter ends with a conceptual architectural ecosystem. Chapter 5 introduces and describes in detail our approach to the interaction of agents based on social obligations; it describes the obligations, their life cycle, and operations for the management of these obligations; from basic operations for defining the basic acts of communication to implementation of interaction protocols. Chapter 6 presents the mechanisms proposed for the composition and adaptation of services in our ecosystem; these mechanisms are implemented on the basic acts described in the previous chapter.

The general objective of the third part, which includes chapters 7 and 8, is to show the evaluation of the approach proposed in the previous chapters. As for particular objectives, Chapter 7 describes the implementation of the proposal and the tools used for this purpose, while Chapter 8 measures the performance of implementing interaction mechanisms proposed for adapting the composition and ecosystem services.

Chapter 9 presents some concluding remarks and the future research direction in the domain of pervasive hardware service ecosystems, and service composition and adaptation.

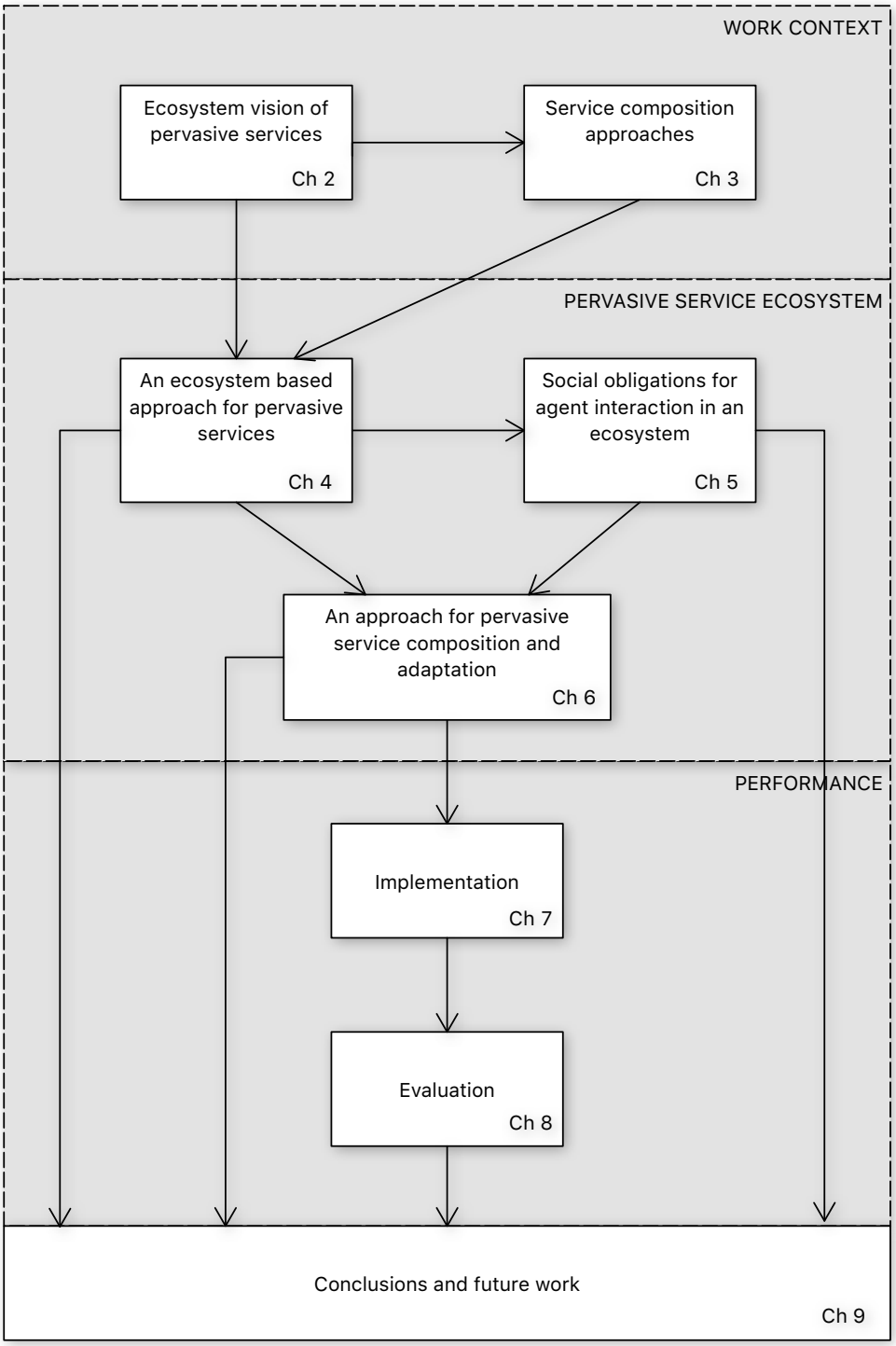

Figure 1.1: Organization of the manuscript

Chapter 2

Ecosystem Vision of Pervasive Services

This chapter introduces the core concepts and heart of the matter for the rest of chapters, which is pervasive services. Additionally, this chapter presents a description of the main metaphors for developing digital service ecosystems. A general vision of services as an ecosystem is presented and particular care is paid to the dynamic aspect and open nature of pervasive services. Lastly, a summary and discussion are provided setting the ideal of this dissertation with respect to current challenges in the field of pervasive services.

2.1 Services

Nowadays, the term service is used in a broad range of domains, where different characteristics have to be fullfilled by an entity in order to be called a service. Georgakoulos defines a service as the fundamental unit of a logical solution, which exists as physically independent software associated with a functional context [18]. Its functions can be invoked by external software through service contracts. These contracts describe the service, and define interaction requirements, constraints and other required service data. However, the service paradigm goes beyond simple entities with public functionalities. According to Erl in [19] a service must have the following main properties.

- Autonomy: from the SOA perspective, services are autonomous, which refers to the independence with which a service can carry out its logic [20]. Therefore services that make up a system can be performed independently, i.e., there is no need for binary dependencies between services. Each service can be developed on a different platform, using a different language and tools.

However, to deploy pervasive services in highly dynamic and open environments, autonomy as conceived by SOA-based approaches is not sufficient. In these environments it is necessary for services to have the capability to make decisions, and their functionality must not be governed or inhibited by external entities.

- Discoverability: it must be possible to search for and locate services. This implies that services must be supplemented with communicative metadata by which they can be effectively located and interpreted. In SOA-based approaches, services must be registered and information about available services published (service's registry). This way the external software, named service's consumer, can simply ask the registry for the needed service and get the details of a fitting service implementation; subsequently, the service's consumer can connect with the service's provider in order to invoke the service (see Figure 2.1).

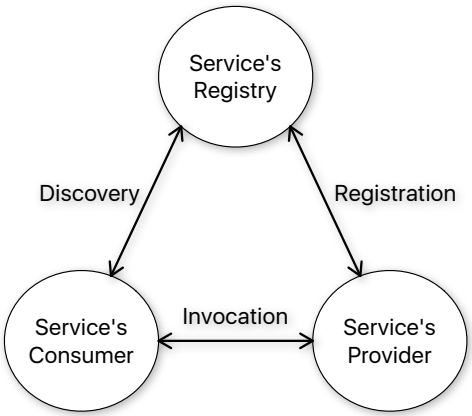

Figure 2.1: Architecture for service discovery based on a centralized approach

Since centralized approaches for service discovery in SOA such as Universal Description, Discovery, and Integration (UDDI) are not scalable [21], several decentralized approaches have been proposed, such as those presented in [22] [23] [24]. Most decentralized SOA-based approaches manage the discovery of services by relying on a distributed registry of services on powerful, reliable and dedicated infrastructure such as servers and reliable communication channels. Thus, discoverability as considered by SOA-based approaches is not suitable for highly dynamic and open environments, because in these kinds of environments shaped by devices in Mobile and Ad Hoc Networks(MANETs), resource availability is not guaranteed and communication channels are not always reliable. Therefore, appropriate discovery mechanisms

must still be developed for these types of environments.

- Composability: composition enables software applications to provide complex services based on the combination of simpler services. Sometimes, a single service is not sufficient to fulfill the user's requirement and often services are combined through service composition to achieve a specific goal. For example, when a user wants to travel, it is not sufficient to book a flight; she must also take care of reserving a hotel, and so on. Services must be designed to participate as effective members of multiple service compositions.

In the remainder of this chapter, the notion of pervasive services as members of ecosystems will be explained, along with the main metaphors involved.

2.2 Toward Pervasive Service Ecosystems

Unlike traditional services that are aimed at the business level and have dedicated infrastructure (such as reliable networks and servers with high processing capacity), pervasive services focus on people moving between environments where normally there is no dedicated infrastructure [25]. Pervasive services pose challenges to service-oriented computing: requirements such as adaptability and self-management must be considered in the service composition in environments without dedicated infrastructure.

Adaptability is the ability of a service to meet user requirements despite changes in the environment; for example, changes in resource availability could change the service functionality. Changes in services' and users' requirements and/or changes within the network may require the use of mechanisms to deal with these changes. Moreover, adaptation is necessary when a significant mismatch occurs between the provider and the request for a service. As the service's execution environment changes due to the user's mobility, the vital services need to be substituted by corresponding services in the new environment in order to ensure continuous operation.

Self-management refers to a system entity's ability to control and manage its resources, functions, security and performance in the face of failures and changes, with little or no human intervention. The complexity of future pervasive environments will be such that it will be impossible for human administrators to manage configuration, performance, and security. Instead, it will be necessary to resort to automation for most of these functions, allowing humans to concentrate on the definition and supervision of high-level management

policies, while the system itself takes care of the translation of these high-level policies into automated control structures.

Usually pervasive services are available only in closed environments such as buildings, especially if they have physical dependencies, usually because they may need specific resources such as sensors, actuators, and communication interfaces. However, if the user moves, he could move outside the range where the service is available and the service becomes unavailable. This means that pervasive systems are centered on isolated physical spaces and are not centered on the users, who are nomadic by nature (i.e., they move from one environment to another). Additionally, the service requirements may change over time depending on the user's context [26].

In order to deploy pervasive service and address its open challenges, several authors such as [27] [28] [29] [30] have taken inspiration from natural systems. A brief description of the main natural metaphors used to build service ecosystems is provided in the following sections.

2.3 Classical Metaphors for Modeling Service Ecosystems

The proliferation of web-based services and the rapid adoption of service-oriented architecture have not only changed the perception of services; they have also changed the way services are offered and consumed. An emerging development in this area is the notion of service ecosystems [30]. An abstract definition, or meaning, of the ecosystem was provided by Tansley [31]: the ecosystem is defined as a biotic community or assemblage and its associated physical environment in a specific place. This general definition has been applied in several forms in different knowledge domains. In computation, the ecosystem is interpreted as a virtual space where digital entities interact with each other to achieve specific objectives [27] [31]. Dynamicity is one of the desirable properties of a service ecosystem because it allows services to appear and disappear at any time. It enables the creation of dynamic solutions composed of various services [32].

The key difference between possible approaches to the realization of frameworks inspired by service ecosystems lies in the metaphor adopted to model the ecosystem. The main metaphors that have been used are: physical [33] [34], chemical [29] [35] [36] [37], biological [28] [38] [39], and ecological [40] [41] [42]. Each metaphor provides its own interpretation for its residents, the environment in which they live, and its laws (see Figure 2.2).

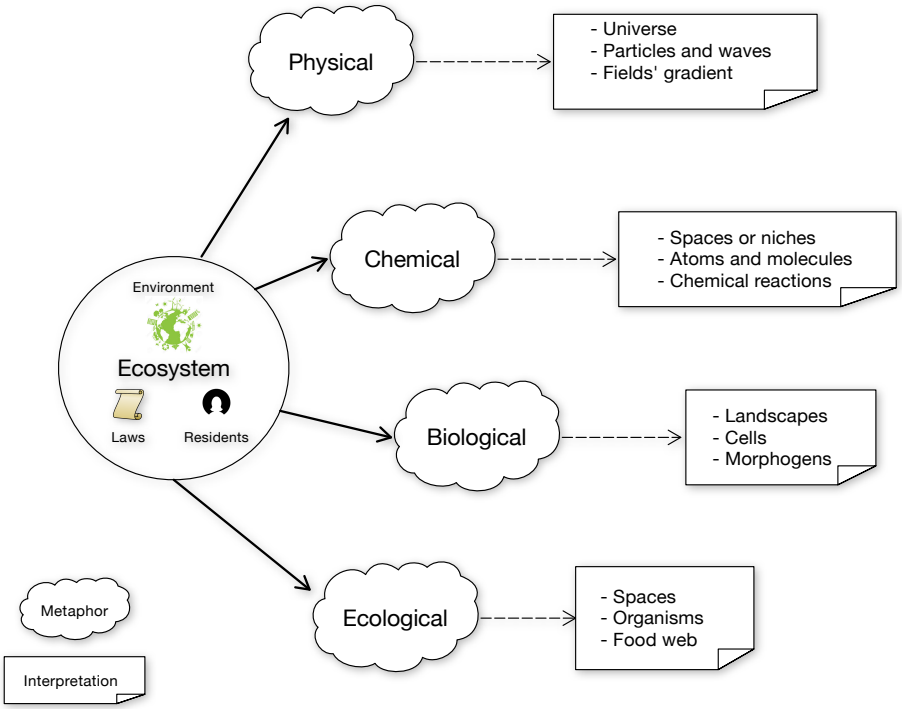

Figure 2.2: Ecosystem metaphors

2.3.1 Physical Metaphor

Physical metaphors are based on the way particles in the universe move according to their local environment, which is made up of gravitational and electromagnetic fields. Particles interact weakly coupled way, using the fields, without the need for particles to know their neighbors in the local environment. Fields (gravitational or electromagnetic) represent information about other particles in the system as summarized contextual information. This information is distributed in fields and the local perception can lead what to do, simply by following the local field's gradient [34]. The physical metaphor considers the particles as residents of the ecosystem. These particles refer to computer components living in the network (i.e., their universe). Additional messages among computer components represent waves in their universe. Activities of particles are driven by laws that determine how particles should be influenced by local gradients and the computational field. In particular, based on the perceived fields, particles can change their status, move or exchange data by means of navigation through such fields.

The physical metaphor has inspired the development of several proposals such as the

one proposed by Crowcroft in [33]. Crowcroft adopted from physics the wave-particle principles to define a network paradigm. In his proposal, a network with swarms of coded content is viewed as dual packets. Here the waves mean traffic in the network and it is started by sources of content such as video and audio input, and sensors. This content spreads by matching subscriptions and interests to content descriptions throughout the network at rendezvous locations. Tuples On The Air (TOTA) [43] is another proposal inspired by a physical metaphor. TOTA was developed to support adaptive context-aware activities in pervasive computing scenarios. Here the universe is a computer network and its laws are based on application-specific rules for representing contextual information and supporting weakly coupled interactions between application components. The key idea of TOTA is to rely on spatially distributed tuples, propagated across the computer network based on the system's laws. The system is organized into three principal parts (see Figure 2.3).

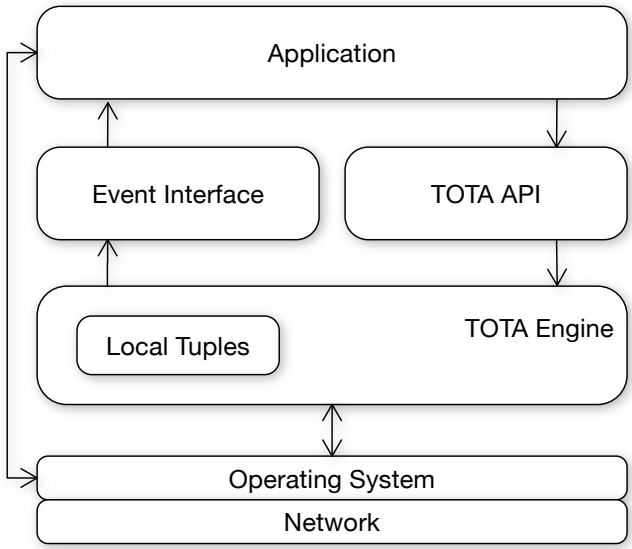

Figure 2.3: Abstract architecture of TOTA [34]

TOTA Application Program Interface (API) is the main interface between the application and the middleware. The API provides functionality to allow an application inject new tuples into the system, retrieve tuples, and place subscriptions in the event interface. The event interface is responsible for notifying the application about subscribed events. The TOTA engine is the core of TOTA. It is responsible for storing references to nodes and managing tuples' propagation. In addition, each TOTA node is provided with a local tuple space to store the tuples that reached that node during their propagation. TOTA has

been tested in application domain such as control algorithms for modular robots, and sensor networks.

Motivated by spatial self-organization features, researchers have studied and applied the physical metaphor in different knowledge domains, particularly due to its ability to facilitate coherent behaviors, even in large scale-systems, for load balancing and data distribution. However, the physical metaphor is inappropriate when the system involves a large variety of components and behaviors.

2.3.2 Chemical Metaphor

Chemical metaphors consider that the residents of the ecosystem are computational molecules, with properties described by some kind of semantic descriptions, which are the computational counterpart to the description of the bonding properties of physical atoms and molecules. The laws that govern the ecosystem's behavior take the form of chemical rules. They dictate how reactions and bonding between residents take place, and can lead to the production of aggregated components or new composite components. In this case, the environment is typically formed by a set of localities. These localities can be interpreted as solutions in which chemical reactions can occur.

Chemical metaphors have been used in several application domains such as service composition. Quitadamo et. al in [29]] proposed a composition model for pervasive communication services. The key idea in their proposal is to exploit semantics as an overlay for service aggregation. Here, the authors confront the discovery and interoperability problems: first, finding and organizing communication services into the environment, and second, enabling them to interact when aggregated into more complex services. An important feature of this work is its focus on the environment (i.e., infrastructure) rather than on a single service. In the environment, pervasive services are joined together by special enzyme components distributed among nodes. Enzyme components are responsible for handling service request messages to process referenced semantic concepts and try suitable adaptation strategies. Using knowledge as the substrate for aggregation reactions, enzymes enable the discovery and interoperability of communication services. An important issue with enzymes is the degree of reasoning these enzymes must have. Napoli and Giordano [35] [36] use a chemical approach to model the process of matching required service functionalities to required conditions using higher-order chemical language [44]. They propose decoupling the workflow instantation from its execution; thus instantation can be

modeled as an independent, autonomous and running system. Angelis et al. propose in [37] a chemical model for service composition operating in a pervasive system. The model is grounded in Self-adaptive Pervasive Service Ecosystems (SAPERE) [10],which propose a multi-agent framework for pervasive computing, and it is inspired by chemical reactions. The main elements of the abstract model are tuples, services and/or applications, agents, and chemical reactions (see Figure 2.4). Service/Application entities produce results by processing data inputs. Each of these entities that want to request or provide information must create an agent. Agents are used as interfaces to exchange data within the tuple space. Tuples are passive entities located in a tuple space that represents a shared container. These tuples are vectors of properties and are used to describe services, applications and contextual information. Chemical reactions are defined by a set of rules governing the tuples and are used to manipulate, update and delete them.

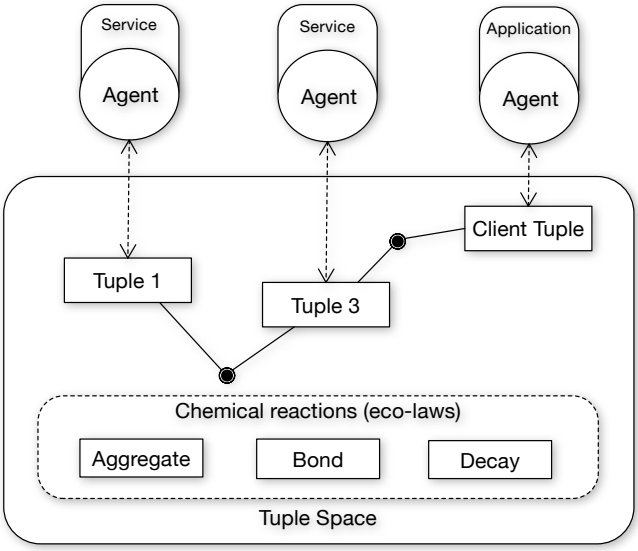

Figure 2.4: A framework based on a chemical metaphor [37]

2.3.3 Biological Metaphor

Biological metaphors typically focus on small biological organisms such as cells and their interactions. The residents are therefore either simple cells or very simple unintelligent organisms. These residents act on the basis of simple goal-oriented behaviors such as moving and eating. Additionally, their behaviors can be influenced by the strength of specific chemical signals in their surroundings with which there is a match. As in physical systems,

residents are expected to be able to spread around and diffuse signals; thus they can influence the behavior of other residents. The laws, together with the spatial computational landscape in which residents exist, determine how the signals should diffuse, and how they can influence residents' behavior and properties.

There are several applications of the biological metaphor. Shen et al. in [28] used a biologically inspired method for controlling robot swarms. They have used a method named *digital hormone model* as a control method for robot swarming behaviors and self-organization. It is based on local communication, signal propagation, and stochastic reactions. The advantages of their approach include its locality, simplicity, robustness, and self-organization.

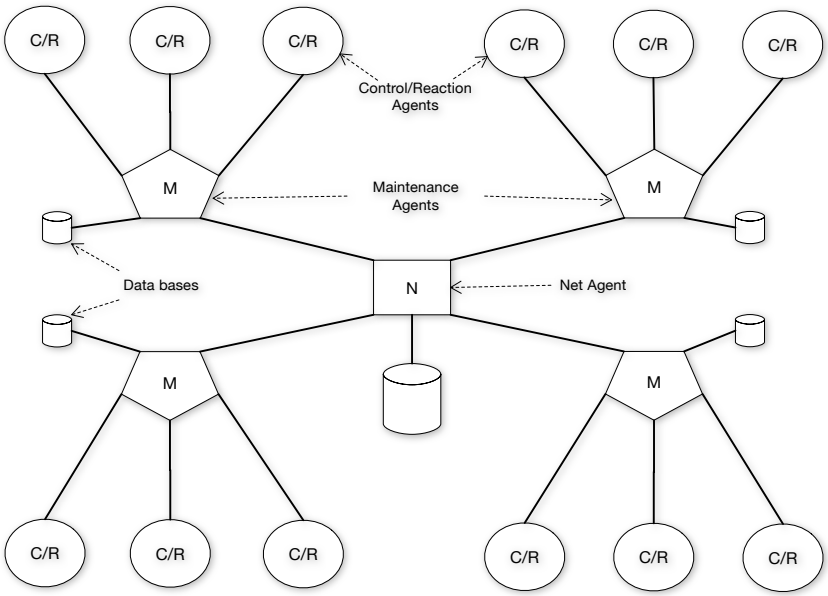

Figure 2.5: Abstract architecture based on a biological metaphor [45]

Flóres et al. in [45], inspired by the immunological system, presented an architecture for intruder detection. The proposal is based on mechanisms that can detect the presence of an intruder, which is a strange resident in the system. For this, the authors proposed using two types of agents: recognizers (lymphocytes-B) and macrophages (lymphocytes-T). Figure 2.5 shows the abstract architecture of the bio-inspired agent-based model. Agents have different roles: Control Reaction, Maintenance and Net. Control-Reactive agents read information about activity, find patterns, trigger alarms and perform protection actions. Maintenance agents create or delete Control-Reaction agents, and eliminate redundant data. Net agents create or delete Maintenance agents. These agents have a global vision

of the network.

Coulter and Ehlers in [38] used a biologically-inspired model to develop a prototype system oriented toward services. It recast distributed resource allocation in mobile multi-agent systems as a variation of the clonal expansion immune algorithm [46]. Briscoe et al. in [39] focus on the digital ecosystem, which provides applications as a counterpart to biological ecosystems. Their proposal includes an optimization technique for the migration of agents, which are distributed in the network; there is a second optimization phase based on evolutionary computing and operating locally on each peer. This second optimization aims to find solutions to satisfy local constraints.

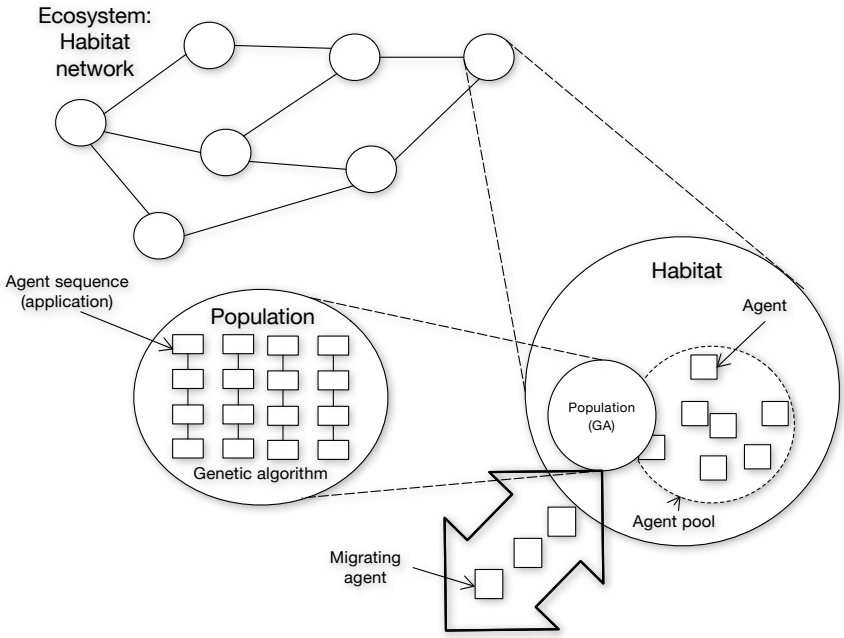

Figure 2.6: Abstract digital ecosystem based on a biological metaphor [39]

Figure 2.6 shows the abstract digital ecosystem proposed by Briscoe. Here we can see habitats interconnected through a network. This connection creates the digital ecosystem based on the agents, populations, agent migration, and the environment. Agents can migrate peer-to-peer; in each habitat local optimization is performed using an evolutionary algorithm, where the set of agents present in the habitat represents the search space.

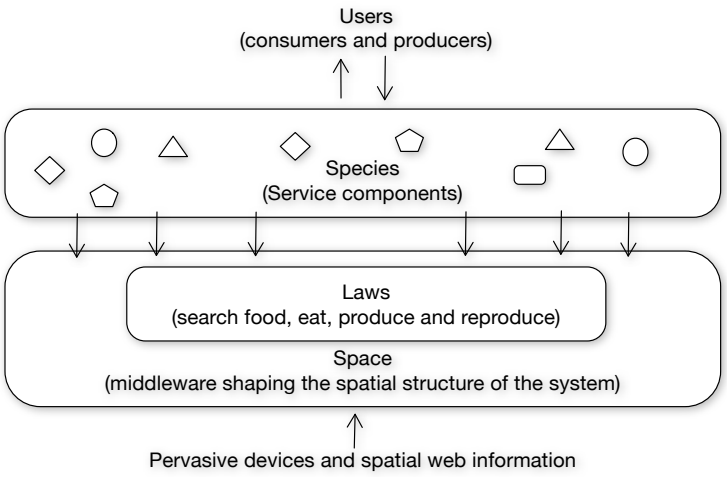

Figure 2.7: An architecture based on an ecological metaphor [41]

2.3.4 Ecological Metaphor

Ecological metaphors focus on animals and their interactions. Usually, residents of the ecosystem are goal-oriented agents of a specific agent class. These agents need search resources to survive and prosper. The laws of the ecosystem are determined by the food web, i.e,, the food web determines how and under which conditions agents are allowed to search for food, eat, and possibly produce and reproduce. The space is organized around a set of localities, and agent migration is possible between these localities. This metaphor has been applied for several purposes. Peysakhov et al. in [40] proposed an ecology-based simulation model for managing agent populations on MANETs, such as the wireless network testbed of the Philadelphia urban area. Residents in the model are agents, whose population can grow and/or migrate. Agents can perform roles as food producers or consumers. In the context of deploying and executing pervasive services, Villalba et al. in [41] argue how model and deploy services as natural systems. i.e., services as autonomous residents, spatially situated with other services, data sources, and devices where all residents act, interact, and evolve according to laws. Villalba proposed an architecture to frame the conceptual structure for service ecosystems (see Figure 2.7).

The architecture is composed of four levels. The lowest, is that of networked devices and information resources. That is the physical environment in which the ecosystem is deployed. Over this, there is a middleware shaping the space of the ecosystem and it contains the interaction rules (i.e. the ecosystem's laws). The species' level contains the living resid-

ents of the ecosystem space: devices, services, data, events and information requests. They will have different features from each other; thus they will belong to different species. At the highest level, producers and consumers of services and data can access the framework.

2.4 Key Differences between Ecosystem Metaphors

The main metaphors to model service ecosystems were covered in this section: physical, chemical, biological and ecological. Table 2.1 shows a comparison of their interpretations of an ecosystem and their components.

Table 2.1: Overview of the interpretation of residents, their environment and its laws

Metaphor	**Environment**	**Resident**	**Laws**	Drawbacks/advantages
Physical	The universe represented by a computer network.	Particles and waves represented by computer components and messages.	Fields determine navigation and activities of particles by means of the gradient.	It has been extensively studied for their spatial self-organization features. It facilitates coherent behaviors in large-scale systems.There are well-developed conceptual tools for controlling spatial behaviors and dynamics. However, it seems to fall short in evolution and time adaptation. It seems unsuitable to support the high heterogeneity of components and residents in real scenarios.
Chemical	Spaces represented by localities of computer components.	Atoms and molecules represented by semantic descriptions and their compositions.	Chemical reactions by means of semantic descriptions' matching.	It can effectively lead to local self-organizing structures. It can accommodate many different components and composites with a simple set of basic laws. As far as self-management is concerned, can be used reagent components can be used to control the dynamics of the ecosystem.
Biological	Spaces represented by abstract or physical scenarios.	Cells represented by self-organized computer components.	Diffusion of morphogens by means of messages. Cells are influenced in their activities by the strength of specific signals.	These allow the formation of spatially localized activity and morphological patterns. Although the number of patterns that can be supported by the propagation of signals and chemical reactions of simple residents seems rather limited, they have been used in several distributed applications. One of the main difficulties is to understand how to adequately control the overall behavior.
Ecological	Spaces represented by physical localities.	Organisms represented by agents, species representing agent' classes and resources representing data.	Eat, produce, and reproduce in order to survive.	It promises to be suitable for local forms of spatial self-organization. It is particularly suited for tolerating evolution over time. However, unlike chemical systems, understanding how to control the equilibrium of real ecological systems is hard. It would probably be difficult also in their computational counterparts.

2.5 Conclusion

In this chapter basic concepts of pervasive services were introduced, and main metaphors to develop digital ecosystems were described and discussed. Pervasive services and digital ecosystems are not new, but previous efforts have focused generally within the context of

software services and closed environments, where there is dedicated infrastructure such as servers with high processing capabilities and reliable communications networks. We do not know how to address pervasive services within the context of services and open environments where there is no dedicated infrastructure.

The main metaphors for modeling service ecosystems were covered in this chapter. An overview of the principal features and different proposals based on these metaphors have been presented. Most of the works inspired by these metaphors have focused on very specific application scenarios; to our knowledge, they have not been implemented for the study of real, open environments and general purposes. Others have simply proposed forward-based systems such metaphors or simulation models.

Autonomous management and adaptability of pervasive services still have to be improved. There are no proper mechanisms for the dynamic composition of adaptable services. Service availability is limited to closed environments; thus user mobility is limited. The way towards the deployment of usable and effective pervasive services still requires designing interaction mechanisms for composition and adaptation of autonomous services, with autonomy from the perspective of multi-agent systems and suitability for open environments.

Members' autonomy is not explicitly considered in terms of the definition of members' interaction in the classical ecosystem metaphors described in this chapter, contradicting the agent paradigm. Furthermore, there is no suitable description and definition of interaction protocols in the ecosystem metaphors.

We now turn our attention to multi-agent systems as a suitable paradigm to represent residents in the ecosystems. Nowadays, the term agent is used in a board range of domains. Wooldridge defines an agent as a "computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives" [47]. However, the agent paradigm goes beyond situated reactive entities in an environment: cognitive capabilities are included, giving rise to rational agents. These agents have four properties:

- Reactivity: agents perceive the environment and proceed accordingly.
- Proactiveness: agents are goal-oriented and act towards the achievement of their objectives.
- Autonomy: agents are self-governed, i.e., not controlled by others.

- Social ability: agents interact with other agents in order to reach their objectives.

The social aspect of agents gives rise to Multiagent Systems(MASs), which are a group of agents that interact among themselves, with the aim of satisfying a global objective. These agents can be either homogeneous or heterogeneous. A MAS should be provided with interaction patterns that direct agents in a coherent manner towards MAS design objectives. Interaction protocols are what give MAS this coherence. An interaction protocol is a set of rules that harmonizes the exchange of messages among agents in order to coordinate their activities in the achievement of objectives. Interaction protocols establish the roles that agents can play during the interaction. A role indicates an expected behavior for each agent that participates in the interaction.

These properties make the agent-oriented paradigm appropriate to address the needs of autonomy and adaptation presented by pervasive services. The integration of multi-agent systems with SOA and web services has recently been investigated. Some researchers focus on the communication between both models, whereas others focus on the integration of distributed services, especially web services, in the agents' structure.

Chapter 3

Service Composition Approaches

This chapter is intended to provide an overview of classical mechanisms for service composition and the suitability of using a CSP formalism to model the hybrid service composition.

3.1 Service Composition

Service requests sometimes include many related functionalities to be fulfilled by the service. Usually, a service has a limited functionality which is not enough to meet the multiple functional needs. The discovery of such services requests involving many tasks could fail due to unavailability of suitable services advertised in the registry. In such scenario, arises the need to compose services with available resources to satisfy the requested complex functionality.

Reusing is an important characteristic of services when complex tasks are carried out [48]. Indeed, composition is usually the reason why tasks are broken down in the first place. A big thing is fragmented because there is a potential benefit in things having to do with the individual pieces (for example reusing individual pieces to recompose different services). Rules of composition describe how different services can be assembled into a coherent global service. In particular, the order in which the services may or may not be invoked is specified, along with the conditions under which a particular service.

Applying this approach establishes an environment where solution logic exists as composable services. As a result, there is the opportunity to recompose the same service in order to fulfill new tasks (see Figure 3.1).

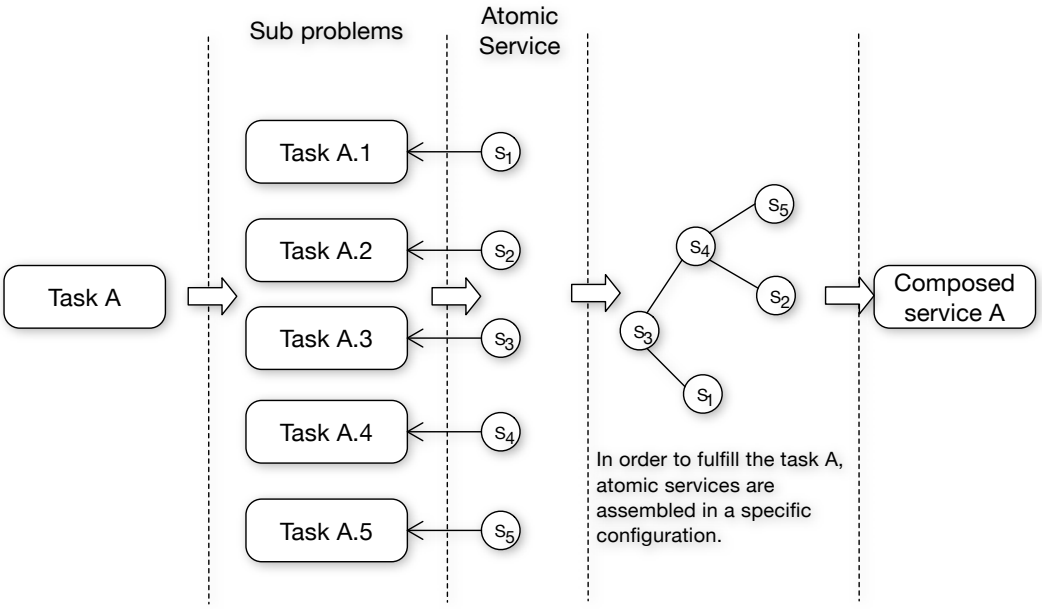

Figure 3.1: Basic idea of service composition

3.2 Classical Service Composition Approaches

In workflow-based composition methods, it is possible to distinguish two main types. Static composition means that the service requester must build a process model before the composition starts [49]; the model includes a set of tasks and their dependency. Each task has a query statement that searches the service instance to accomplish the task. The dynamic composition both creates a process model and selects atomic services automatically (or with low user intervention) [49].

3.2.1 Static Service Composition

There are two principal approaches for static service composition. The first approach, named service orchestration, combines available services by means of a central coordinator (the orchestrator). The orchestrator is responsible for invoking and combining atomic services. The second approach, named service choreography, does not assume the existence of a central coordinator. Service choreography defines complex services via the conversation that must be carried out by each participant. Following this approach, the service composition is achieved through peer-to-peer interactions among the collaborating services. While several proposals exist for orchestration languages (e.g. Business Pro-

cess Modeling Language (BPML) [50] and Business Process Execution Language (BPEL) [51]), choreography languages are still in the preliminary stages of definition.

This type of composition is not flexible for the final user because most proposed static composition techniques and languages are usable only by software developers, not by end users. Moreover, static composition is not suitable for dynamic environments because service composition is wired off line.

3.2.2 Dynamic Service Composition

Dynamic composition of services is a more challenging problem. When a functionality that cannot be realized by the existing services is required, the existing services can be combined to fulfill the request. Dynamic service composition requires the location of services based on their capabilities and the recognition of the specific services that can be useful to create a composed service [52]. The full automation of this process is still an open challenge. This type of composition is more flexible; however the drawback of this kind of composition is that functionality is not guaranteed.

In the remainder of this chapter, several approaches for modeling service composition will be explained.

3.2.3 Classical Formalisms for Service Composition Specification

In this section, we describe the main formalisms and languages that have been used for modeling service composition; it considers the two approaches discussed in the previous section.

3.2.3.1 Automata

An automata consists of a finite set of states, actions, labeled transitions between states and initial states. The labels represent actions, and a transition label indicates the action that causes the transition from state to state. The intuitive way in which an automata can model a system's behavior has led to a variety of automata-based specification models. Some examples are Input/Output automata [53] [54] [55] and their many variants: timed automata [56] [57], team automata [58] [59] and others. Automata-based models are being used more and more to make formal descriptions and compositions, and to verify compositions of web services.

In [60] the authors provide an encoding of BPEL processes into web service timed state transition systems, a formalism that is closely related to timed automata, and discuss a framework in which timed assumptions can be model checked. The authors analyze and verify properties of web service compositions of BPEL processes that communicate via asynchronous Extensible Markup Language (XML) messages. Their approach first translates the BPEL processes into a particular type of automata whose every transition is equipped with a guard in the form of an XML Path Language (XPath) expression, after which these guarded automata are translated into Process Meta Language (PROMELA), the input language of the model checker Spin. Finally, Spin can be used to verify whether web service compositions satisfy certain linear temporal logic properties.

3.2.3.2 Petri Nets

Petri nets were introduced as a framework to model concurrent systems in [61] . Their main advantage is the natural way in which many basic aspects of concurrent systems are identified both mathematically and conceptually. Within a Petri Net, one distinguishes between places and transitions. Transitions are connected to places and places to transitions, by arcs. Hence, a net is a bipartite directed graph. In some models, certain elements may be labeled. The dynamics of a net are given in the form of rules. These rules determine when a transition can be fired and what its effects are on the current state. Several authors have used Petri Networks to model service compositions. For example, Narayanan and Mcllraith describe web service compositions in a Petri-net-based formalism, complete with an operational semantics; they discuss the implementation of a tool to describe and automatically verify the composition of web services [62]. In [63] a Petri-net-based design and verification framework for service composition on the web was proposed.

The frameworks and tools described above have the advantage of allowing one to simulate and verify the behavior of one's model at design time, thus enabling the detection and correction of errors as early as possible. As such, these approaches help increase the reliability of web service applications. However, they are not suitable for service composition in runtime.

3.2.3.3 Process Algebra

Process algebra is a popular formalism to describe and reason about process behaviors. Like Petri nets, process algebras are precise and well-studied formalisms that allow the

automatic verification of certain properties of their behaviors. Likewise, they provide a rich theory of bisimulation analysis in order to establish whether two processes have equivalent behaviors. Such analyses are useful for establishing whether a service can substitute another service in a composition or verifying the redundancy of a service. π-calculus is a process algebra that has inspired composition languages such as BPEL and XML Language (XLANG) that provide a formal model and theory for the automatic verification of properties of the behaviour of models. For example, in [64] the authors use process algebras to describe services, compose services, and verify them with a particular focus on their interactions. They present a case study in which they use Milner's Calculus of Communicating Systems(CCSs) to specify and compose web services as processes, and the concurrency workbench to validate properties such as correct web service composition. As is the case with Petri-net-based frameworks and tools, process-algebraic tools are also well-suited to improve the reliability of web service development by simulating and verifying the behavior of one's model at design time.

The Automata, Petri Net and Process Algebra formalisms described above have different characteristics, advantages and disadvantages. Each of them allows us to model different aspects of a problem as concurrency. However, these formalisms focus on the simulation and analysis of problems, then subsequently implement the right solution for a particular problem class. However, they are not suitable for deploying pervasive composite services, because it is necessary to model and solve the composition problem in runtime. Several authors from Yokoo [65] to Ghedira [66] have explored the constraint satisfaction problem as an alternative formalism that allows us to model problems and solutions in run-time. In this formalism, simulation is not necessarily indispensable to find suitable solutions. In the following section the constraint satisfaction problem is described and presented as an alternative formalism to model and solve the pervasive service composition problem.

3.3 Service Composition as a Constraint Satisfaction Problem

Classical constraint-satisfaction-based problems can be expressed by using the CSP formalism [65]. A CSP is a triplet X, D, C where $X = \{x_1, ..., x_n\}$ is the set of variables to instantiate, $D = \{D_1, ..., D_m\}$ is the set of domains, each variable x_i is related to a domain

of value, and $C = \{c_1, ..., c_k\}$ is the set of constraints, which are relations between some variables from X that constrain the values the variables can be simultaneously instantiated to. Therefore, making a decision consists of finding a solution, for instance a complete and consistent assignment of X.

Many problems of different domains can be solved by CSP, for example, planning, scheduling and circuit analysis. Let us consider the typical toy problem of *n-queens*. In this problem the goal is to put n queens on a chessboard of $n*n$ size. On the chessboard, none of the queens should be are able to attack any other. This problem is called CSP because the goal is to find a configuration that satisfies the given constraints. In the case of *4-queens*, we can model the problem using the CSP formalism as follows:

- $X = \{q_1, q_2, q_3, q_4\}$, each variable q_i corresponds to the queen placed in the ith column.
- $D = \{D(q_1), D(q_2), D(q_3), D(q_4)\}$, where $D(q_i) = 1,2,3,4 \forall i \in 1,2,3,4$. The value $v \in D(q_i)$ corresponds to the row where the queen representing the ith column can be placed.
- $C = \{c_{i,j} : (|q_i - q_j| \neq |i - j|) \wedge (q_i \neq q_j) \ \forall \ i, j \in \{1,2,3,4\}$ and $i \neq j\}$ is the set of constraints between each pair of queens. These constraints forbid the involved queens to be placed in the same row or diagonal line.

We can use a matrix to represent the *4-queen* problem (see Figure 3.2): In the matrix, columns represents the variables X and rows represents the values in D that could instantiate each variable.

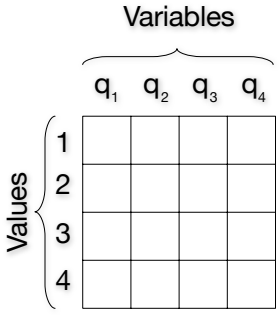

Figure 3.2: The n-queen problem as a CSP

CSP is a powerful formalism for modeling and solving problems. However, it is not suitable for addressing pervasive service composition problems because CSP is based on a

centralized approach, while the composition problem of pervasive services is a distributed problem by nature. In order to address these problems several extensions of the original CSP have been developed. In the next section, a brief description of these CSP extensions is provided.

3.3.1 Approach Taxonomy

The original CSP framework has several limitations. For example, the original CSP assumes that: the problem is static, there is a global state, the problem can be solved by a single entity, and the problem's constraints are categorical. Because of this, many extensions, such as Dynamic Constraint Satisfaction Problem (DynCSP), DisCSP, and Soft Constraint Satisfaction Problem (SoftCSP), have been proposed. In each of these frameworks, there are several approaches to solving constraint satisfaction problems (see Figure 3.3).

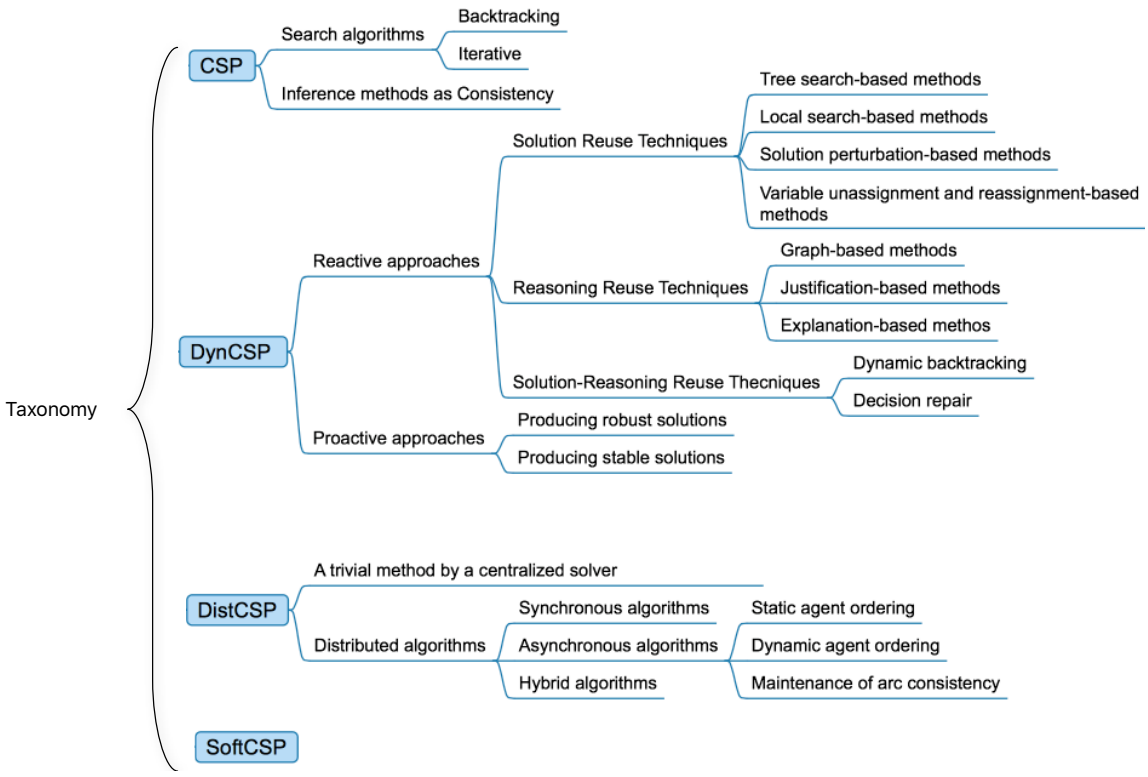

Figure 3.3: Taxonomy of CSP solver approaches

The CSP framework has two main approaches to solve constraint satisfaction problems: search algorithms [67] and inference methods [68]. In general search algorithms can

be classified as iterative and backtracking. In backtracking approaches algorithms build a partial solution that satisfies all of the constraints within the subset. Then the partial solution is expanded by adding new variables one by one [69]. In iterative approaches, on the other hand, there are not partial solutions: instead, a whole flawed solution is revised by a hill-climbing search. In the inference-method approach, consistency is an example of this kind of algorithms. This approach needs, pre-processing procedures that are invoked before the search algorithm.

DynCSP allows problems of constraint satisfaction to be solved in dynamically changing environments, for example, dynamic scheduling where tasks arrive continuously, and sensor networks where targets to be tracked move. There is a wide body of research on these topics, for example [70] [71] [72] [73]. However, DynCSP does not support changes in the number of problem constraints or in the elements involved in the construction of the solution. Solution methods in DynCSP can be classified as reactive or proactive approaches. In the reactive approach, algorithms are based on three type of techniques: solution reuse, reasoning reuse and solution-reusing reuse. [74] [75] [76] [77] are some examples of proposals based on these techniques.

SoftCSP framework is a generalization of CSP in which the constraints are not categorical [78]. SoftCSP enables modeling constraint satisfaction by considering aspects such as preferences, costs, or probabilities [76]. Constraint use in CSP and DynCSP could be too strict and not suitable for some problems. Because of this, there has been an interest in using SoftCSP for modeling these problems. To name some examples, in [79] a SoftCSP approach is proposed for modeling and reconciling non-functional requirements in web services based on multiagent systems and the authors demonstrate the framework in the potential-use case of a demand-driven supply network, and in [80] presents a constraint satisfaction system for designing and manufacturing.

As a distributed problem, one possible way to represent service composition is the DisCSP formalism, which considers a problem as a set of variables (required services) to be assignes to a given set of constraints between the possible values (service instances) for these variables [81]. The DisCSP framework has been introduced by Yokoo in [65]. Solving a problem is finding an assignment for each variable that respects the constraints. There are several methods to solve CSPs in a centralized manner [66], but they are beyond the scope of this study, since we will address specifically distributed problems. The main distributed algorithms to solve DisCSPs can be classified in two types: synchronous and asynchronous algorithms. Several of these are distributed versions of centralized

algorithms such as backtracking and weak-commit search. Because events in pervasive service composition are normally concurrent, in this work we are interested in asynchronous algorithms. The following section describes the DisCSP framework and the main asynchronous algorithms for solving DisCSP problems.

3.3.2 Distributed CSP Framework

In a DisCSP variables and constraints are distributed among a set of systems that have to collectively solve the CSP. Each system is responsible for assigning one or more variables by interacting with other systems. This does not imply that the solving process is decentralized. In most approaches, systems behave concurrently during a loop consisting of waiting for messages and reacting to received messages. Such messages contain information about the chosen values, the conflictual values, the violated constraints or even organizational information such as priorities.The topology of a constraint-based problem can be represented by a constraint network, in which vertexes represent variables, and edges represent binary constraints between variables.

There are several methods to solve CSPs in a centralized manner [66].

3.3.3 Distributed Constraint Satisfaction Problem Solving

A DisCSP could be solved using centralized algorithms such as min-conflict backtracking [82], breakout [83], and branch and bound [84]; however they are beyond the scope of this study. Since the service composition problem is naturally distributed, we will address specifically distributed problems, i.e., problems where a leader agent is selected among all agents, and all the information about variables, variables' domains and constraints is gethered into the leader agent. Thus the leader agent can solve the CSP alone using centralized CSP algorithms. However, the cost of collecting all information about a problem can be prohibitively high [65].

In this section, we classify constraint problem solving approaches and evaluate each category with respect to the characteristics presented below in order to identify commonalities and differences between categories.

- **Global state is unknown:** micro-level entities are not conscious of the global state of the system, which is however valuable at the macro-level.
- **Local actions:** actions of a system are limited to its own limited neighborhood.

- **Decentralized decision:** decentralization means that no system has the power or the capability to decide for the others, or to solve the whole part of the problem, at a given time.

- **Problem distribution:** this refers to the manner and the degree of distribution of the problem among systems (for example one variable per system)

- **Bounded known:** a system cannot know the values of other systems.

We will further use these characteristics as an analysis matrix for determining whether or not an existing approach can be used as a starting point for designing ecosystems of pervasive services. We organize main existing methods for solving DisCSPs into three categories: distributed local research methods, population-based methods, and complete and asynchronous methods. These methods constitute an inspiration source for developing composition mechanisms for pervasive services, even if they are partially relevant in a dynamic and distributed context.

Distributed local research methods include Distributed Breakout Algorithm (DBA) and Environment, Reactive Rules and Agents (ERA). These methods explore the search space from state to state, from one complete assignment to another complete assignment (i.e., from one potential solution to another potential solution). The main advantage of this anytime behavior is that it can naturally handle dynamics (added constraints, changing values) because it always tends to improve the current state of the system, specifically when the state has been altered by environmental disturbances. While often efficient, these methods are not complete and require some subtle parameter tuning.

Population-based methods include Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). In population-based approaches, agents use simple local rules to govern their actions, and via the interactions of the entire group, the population achieves its objectives. The cooperative individual behavior leads to an emergent collective one. From an engineering point of view, population-based algorithms are largely applied to optimization problems. Each agent has the capability to find the solution of the global problem autonomously, without following central orders or some global plan. The advantage of these methods is their inspiration from natural phenomena adapted to dynamic environments, which may facilitate their applicability to dynamic problem solving contexts. However, they require more memory than the previous approaches since the search space is explored concurrently at several states.

We pay special attention to Asynchronous Back Tracking (ABT) and Asynchronous Weak-Commitment Search (AWCS) algorithms because they are complete (i.e. if there is a solution, they guarantee the finding of the solution) and they are the base of the main algorithms to solve DisCSPs [65]. These methods rely mainly on an ordered organization. A priority level is assigned to each member of the organization; that priority may be assigned using a certain established policy, for example, seniority, amount of resources, confidence level, among others. Within the organization agents only communicate with others agents of lower priority for informing value changes and with the agent with directly higher priority for informing that there is a conflict in their variable assignments. The agent with the highest priority is responsible for initiating the termination procedure. This priority order is fixed in ABT. With this algorithm, a partial solution is never modified unless it is certain that the partial solution cannot be a part of any complete solution.

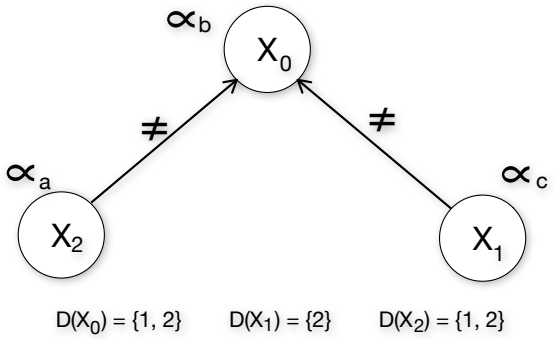

Figure 3.4: Example of constraint network

The ABT algorithm allows agents to run concurrently and asynchronously. Thus each agent instantiates its variable and communicates the variable value to higher-order agents. A constraint network can be used to represent a DisCSP (see Figure 3.4). In the constraint network, nodes represent agents with one variable (for the sake of simplicity) and each link represents a constraint. One agent is assigned that constraint, and receives the other agent's value. The links are directed between two agents: from the value-sending node to the constraint-evaluating node. The ABT algorithm is driven by messages and each agent can handle multiple messages concurrently. This algorithm uses two main messages: *ok*? and *noGood* (see Figure 3.5). A brief description of the main idea of the ABT algorithm is provided below:

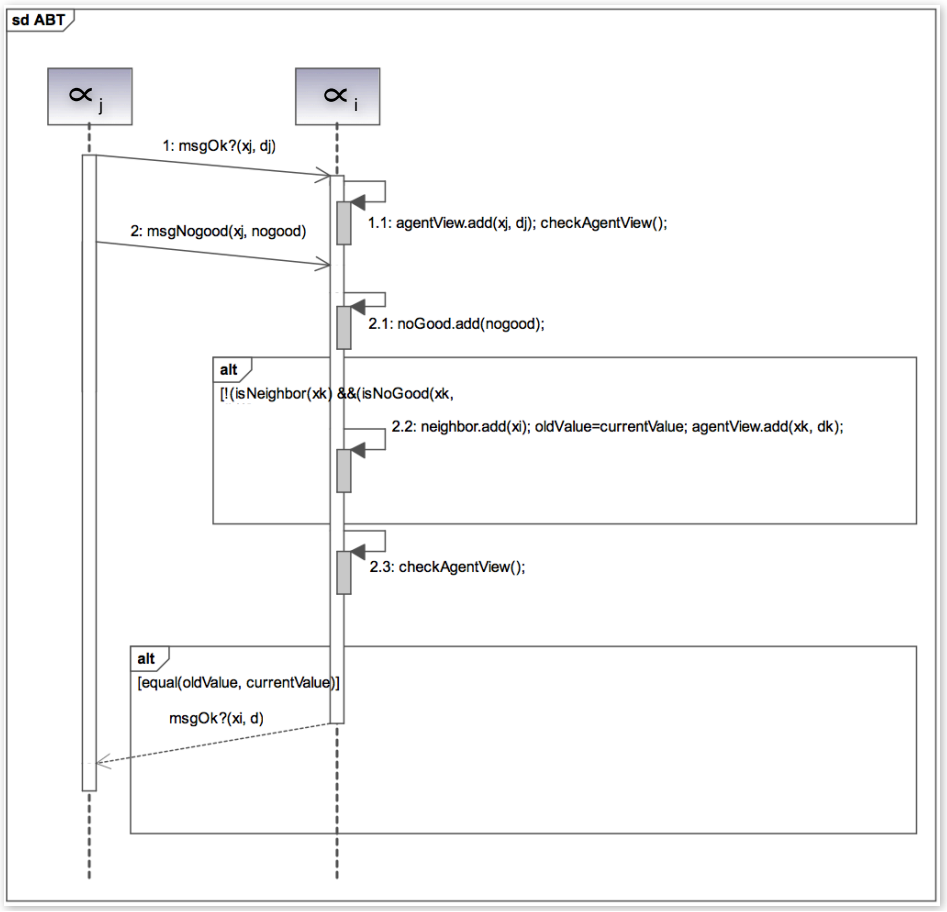

Figure 3.5: ABT Messages between agents

1. When an agent receives a message *ok*? from another agent, it asks whether the value chosen is acceptable (see Algorithm 3.1).

when received (ok?, (x_j, d_j)) **do**
    agentView.add(x_j, d_j);
    checkAgentView();
end do

Algorithm 3.1 Agent behavior when receiving a message *ok*?

2. When a value-sending agent receives a *noGood* message, it indicates that the constraint-evaluating agent has found a violation of some constraint. Algorithm 3.3 describes the basic idea to managing *noGood* messages.

when received (noGood, x_j, nogood) **do**
    noGoodList.add(nogood);
    when (x_k, d_k) where $x_k \notin$ neighbors and
    $x_k \in$ noGood **do**
        neighbors.add(x_k);
        agentView.add(x_k, d_k);
    end do
    oldValue $\leftarrow$ currentValue;
    checkAgentView();
    when oldValue = currentValue **do**
        send(ok?, (x_j, currentValue)) to x_j
    end do
end do

Algorithm 3.2 Agent behavior when receiving a *noGood* message

3. Agent view is constituted by a set of values received from other agents that are connected to it (see Algorithm 3.3). Thus, the evaluating agent adds the pair (x_j, d_j) to its agent view and checks whether its own value assignment $(x_i, currenValue)$ is consistent with its agent view.

checkAgentView()
    when agentView and currentValue are inconsistent **do**
        if $\nexists$ consistent(value) $\in D_i$ with agentView **then**
            backtrack();
        else
            select $d \in D_i$ where agentView and consistent(d);
            currentValue $\leftarrow d$;
            send (ok?, (x_i, d)) to outgoings links;
        end if
    end do

Algorithm 3.3 CheckAgentView agent behavior

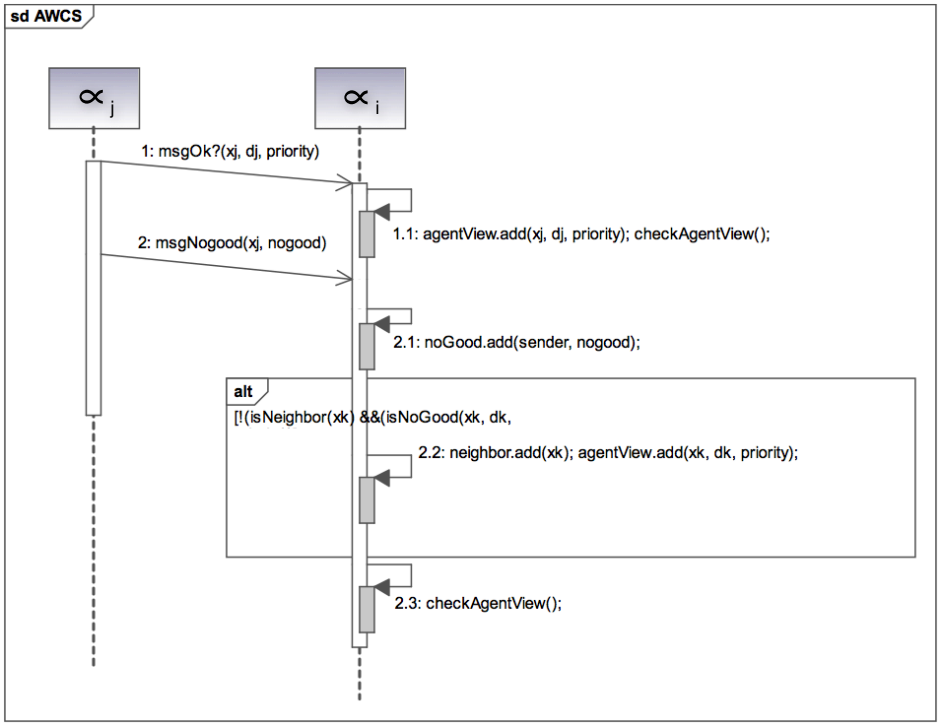

Figure 3.6: AWCS messages between agents

backtrack()
noGoods ← { V | V is inconsistent subset of agentView };
when an empty set is an element of noGoods **do**
broadcast to other agents that there is no solution;
terminates this algorithm;
end do
for each V ∈ noGoods **do**
select (x_j, d_j) where x_j has the lowest priority in V;
send(nogood, x_i, V) to x_j;
remove(x_i, d_j) from agentView;
end do
checkAgentView();
end do

Algorithm 3.4 Backtrack agent behavior

4. A *noGood* is a subset of an agent view if the agent is not able to find any consistent

value with the subset. If the agent finds a *noGood*, the assignments of other agents must be changed. Therefore, the agent triggers a *backtrack* and sends a *noGood* message to one of the other agents (see Algorithm 3.4).

In the AWCS algorithm, priority order changes at every conflict detection, in order not to keep a part of the organization that is known to be incorrect. That is, in the AWCS algorithm an agent can revise a bad decision without an exhaustive search by dynamically changing the order of agents. Here each agent assigns a value to its variable, and sends the variable value to other agents. Agents behaviors are driven by messages(see Figure 3.6).

A brief description of the AWCS algorithm is provided below:

1. In the AWCS algorithm, each agent sends its variable value to both lower and higher neighbors. The priority value and the current value assignment are communicated through the **ok?** message. When agent i receives this messages it executes the following behavior (see Algorithm 3.5).

when received (ok?, (x_j, d_j, priority)) **do**
    agentView.add(x_j, d_j, priority);
    checkAgentView();
end do

Algorithm 3.5 Agent behavior when receiving an *ok*? message

2. As in the ABT algorithm, here when a value-sending agent receives a *noGood* message, it indicates that the constraint-evaluating agent has found a violation of some constraint. Below a pseudocode shows the key idea to managing *noGood* messages (see Algorithm 3.6).

when received (noGood, x_j, nogood) **do**
    noGoodList.add(nogood);
    when (x_k, d_k, priority) where $x_k \notin$ neighbors
    is contained in noGood **do**
        neighbors.add(x_k);
        agentView.add(x_k, d_k, priority);
    end do
    checkAgentView();
end do

Algorithm 3.6 Agent behavior when receiving a *noGood* message

3. The priority order is determined using communicated values. If the current value is not consistent with the agentView, the agent changes its value so that the value is consistent with the agentView, and also the value minimizes the number of constraint violations with variables of lower-priority agents (see Algorithm 3.7).

```
checkAgentView()
    when agentView and currentValue are inconsistent do
        if !consistent(value) ∈ D_i with agentView then
            backtrack();
        else
            select d ∈ D_i where agentView and consistent(d)
            minimizes the number of constraint violations with lower
            priority agents;
            currentValue ← d;
            send (ok?, (x_i, d, currentPriority)) to neighbors;
        end if
    end do
```

Algorithm 3.7 CheckAgentView agent behavior

4. When x_i cannot find a consistent value with its agentView, x_i sends noGood messages to other agents, and increments its priority value. If x_i has already sent an identical noGood, x_i will not change its priority value but will wait for the next message (see Algorithm 3.8).

backtrack()

noGoods ← { V | V is inconsistent subset of agentView };
when an empty set is an element of noGoods **do**
broadcast to other agents that there is no solution;
terminates this algorithm;
end do
when no element of noGoods is included in noGoodSent **do**
for each V ∈ noGoods **do**
noGoodSent.add(V);
for each $(x_j, d_j, p_j) \in$ V **do**
send (nogood), x_i, V) to x_j
end do
for each
$p_{max} \leftarrow max_{(x_j,d_j,p_j) \in agentView}(p_j)$
currentPriority ← 1 + p_{max};
select $d \in D$ where d minimizes the number
of constraint violations with lower priority agents;
currentValue ← d;
send(ok?, (x_i, d, currentPriority)) to neighbors;
end do

Algorithm 3.8 Backtracking agent behavior

We illustrate the AWCS algorithm by means of the distributed 4-queen problem. Figure 3.7 describes the initial scenario. Agents communicate these values with their neighbors. The value within parentheses, over each X_i, represents the priority value (start with 0). Because the priority value of all agents are equal, the order is defined by the alphabetical order of the identifier. Thus, only the value of x_4 is inconsistent with its agent view. Agent x_4 therefore sends *noGood* messages and increments its priority order. In this example, the value minimizing the number of constraint violations is 3, because it conflicts only with x_3. Thus, x_4 selects 3 and sends *ok*? messages to its neighbors (see Figure 3.7 (ii)).

Then, x_3 tries to change its value. Since there is an inconsistent value, agent x_3 sends noGood messages and increments its priority order. In this case, the value that minimizes the number of constraint violations is 1 or 2. In this example, x_1 selects 1 and sends *ok*? messages to the other agents (Figure 3.7 (iii)). After that, x_1 changes its value to 2, and a

solution is obtained (see Figure 3.7 (iiii)).

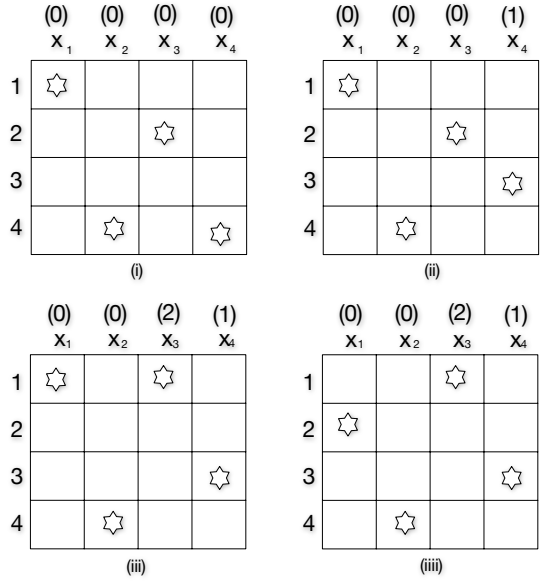

Figure 3.7: Example of an AWCS algorithm execution

For more details on these algorithms, we refer the reader to Bessiere's article and Yokoo's article on DisCSP [85] [86].

3.3.4 Comparing the Main Features of DisCSP Solving Algorithms

Table 3.1 summarizes the main characteristics of the algorithms described above according to the criteria described at the start of this section (global state, local actions, decentralized decision, problem distribution and bounded known).

Table 3.1: Main characteristics of reviewed algorithms

	ERA	**PSO**	**ACO**	**ABT**	**AWCS**
Global state is unknown	●	❍	❍	❍	❍
Local actions	●	●	●	●	●
Decentralized decision	❍	❍	❍	❍	❍
Problem distribution	●	●	●	●	●
Bounded known	❍	◗	❍	◗	◗

●: Yes, ❍: No, ◗: Partially

Although all the algorithms described above are based on the distribution of the resolution process among the participants (for example, agents, systems, ants and particles), these algorithms require centralization of some processes or data in one element of the system. Therefore they can not be considered fully decentralized. For example, some algorithms such as *ACO* and *ERA* centralize the data, while algorithms such as *AWCS* and *ABT* centralize the consistency of the solution.

3.4 Conclusion

Two main topics were covered in this chapter: classical service composition approaches and their formalisms, and the constraint satisfaction problem formalism as an alternative approach to model service compositions. Both drawbacks and advantages were discussed.

First, the composition of services was described as a key feature for solving complex problems using simple services; then, the static and dynamic composition approaches were introduced; next, the main formalisms used for service composition were introduced: the automata as a formalism for modeling the behavior of systems, then Petri Nets, which make it possible to model synchronization and concurrency in system behavior, and finally, process algebra to describe or reason about the behavior of processes. These formalisms focus on simulation and analysis of problems, and then implement suitable solution for a particular problem. This was the starting point to describe the constraint satisfaction problem formalism as a tool to model service composition.

The above approaches and formalisms have generally been used to develop mechanisms for the composition of services that are deployed in closed environments. Still pending is the development of composition mechanisms that are appropriate for open environments that do not have dedicated infrastructure, for example, environments with infrastructure based on mobile devices and ad hoc networks.

The constraint satisfaction problem was presented as a formalism for service composition, and a set of approaches to solve DisCSP problems was described. There are distributed versions of classical algorithms that still remain centralized, even though the execution and the solving process are concurrent; for instance, several sub-processes solve sub-problems of the whole problem in a parallel manner but are coordinated by a single system that may fail.

In order to deploy pervasive services, automatic composition, scalability and exception handling still have to be improved. Automatic composition means that the end user or ap-

plication developer may request a task, and a composition engine should select adequate services and provide the user with the composed service in a transparent fashion. Some of the main open challenges remaining in automatic composition are how to identify potential services, how to compose them, and how to verify how closely they match a request. Moreover, composing two services is not the same as composing hundreds of them. In the real world, users would typically need to interact with several services while applications may invoke possibly several hundred services. Therefore, one of the critical issues is the scaling of the proposed approaches to the number of services involved. Finally, pervasive service composition uses external services that are controlled by the service owner, so the handling of exceptions during the process of invocation must be taken into account in case external services do not respond.

The rest of this document will describe our distributed approach to designing service ecosystems to address the issues described in the sections 2.5 and 3.4.

Chapter 4

An Ecosystem-Based Approach for Pervasive Hardware Services

As stated in the introduction to this manuscript, our first goal is to develop a framework for pervasive hardware service ecosystems. To this end, we define a conceptual architecture around which to frame the main components of the service ecosystem inspired in a social metaphor.

4.1 A Social Metaphor

The challenges to deploying pervasive hardware services in open and dynamic environments led us to take inspiration from natural systems, as other authors have done [33] [35] [39] [37]. In our case we use the ecosystem concept to model a service ecosystem using a social metaphor. As was seen above in chapter 2, an ecosystem can be defined as a loosely coupled environment organized by species, where each species conserves the environment, and is proactive and responsive for its own benefits [31]. There are two key elements in an ecosystem: species and their environment. Species interact with each other and balance each other (even though some species might play a temporary leading role) and an environment supports species' ecological needs.

The development of a pervasive service ecosystem inspired by a social metaphor should conceive of the species in it as the members of a society, each having the goal of reaching its objectives by finding the appropriate resources while following social norms. The social norms ruling the dynamics of the members' interaction, and the organization of the ecosystem (typically structured around spatially confined groups of members with similar

goals) determine how the members of the ecosystem can look for and find resources.

In general, an ecosystem considers the presence of members with several skills and objectives. These members may play one or more roles (such as provider and customer), which are dynamic and refer to the behavioral expectations of the individuals in their relations with others. A member of the ecosystem may play the role of customer when it requires the assistance of another member of the ecosystem to achieve its goals. However, it may also play the role of provider when it offers assistance to other members of the ecosystem. Social interactions among ecosystem members (with heterogeneous skills and resources) playing different roles drive them to reach their goals.

4.1.1 Ecosystem Members and Species

As presented in Chapter 2, there are several metaphors for the construction of ecosystem services. Depending on the metaphor used, the characteristics of the environment may vary, as well as the members of the ecosystem and how they interact. With the aim of establishing an appropriate framework for our purpose, we have proposed a formal definition of each of these elements.

4.1.1.1 Ecosystem Members

Members of the ecosystem are agent-managed devices. Device can be defined as a 3-tuple $< SR, HR, \alpha_i >$, where $SR = \{f_0, f_1, ...\}$ is its finite set of software resources (i.e. a set of functionalities the device can provide); $HR = \{h_0, h_1, ...\}$ is its finite set of hardware resources over which the services are carried out; and $\alpha =< G, O, S, K >$ is the agent, where $G = \{g_0, g_1, ...\}$ is its finite set of goals (what the agent must fulfil), $O = \{o_0, o_1, ...\}$ is its finite set of obligations to search for a state of affairs (i.e. provide services), $S = \{s_0, s_1, ...\}$ is its finite set of device functionalities $(S \subseteq SR)$ made public as services to be requested by other agents, and $K = \{P, I, D\}$ is its knowledge made up of the agent's perceptions (P) of its environment, internal states (I) and application domain knowledge (D).

4.1.1.2 Services

A service is considered as a computational wrapper around some set of functionalities. The functionalities that are represented by the service can be abstract (e.g. algorithms for data processing), or manage resources in the real world (e.g. sensors or actuators). A service is able to represent these different kinds of resources by providing each resource with a

well-defined interface. Formally a service can be defined as a 3-tuple $s =< contract, F >$ where *contract* describes the service interface, the set of assumptions that must hold in order to perform the service, as well as the effects of performing the service, and $F \subseteq SR$ is the set of functionalities that make up the service. Computing devices typically have limited resources; therefore, providing complex services could require the cooperation among devices managed by agents. In this context, there are two kind of services: atomic and composite services. An atomic service is one in which all the required functionalities and resources are in the same agent; and a composite service is one whose required functionalities and resources are distributed in two or more agents.

4.1.1.3 Species

Species are key elements in a natural ecosystem. A species can be defined as a collection of ecosystem members who share common characteristics, abilities, needs and often goals. For our purpose, the service ecosystem species are represented by groups of agents with similar resources and functionality who are likely to provide common services. Therefore, agents who are in a common environment (such as a room, a building or a city) may be members of one service ecosystem. A service ecosystem is defined as a tuple $< CAG, GSR, SO >$, where *CAG* denotes a collection of agent groups (i.e a species in the ecosystem); *GSR* denotes social relations among agent groups; and, *SO* is set of social norms. Below is a detailed definition of each element:

a) **Species as a collection of agent groups**(CAG)**:** In this collection of agents groups, each group is an association of agents with common interests that is regulated by control mechanisms; these control mechanisms are provided by social norms (i.e. rules governing agents' behaviors). Formally an agent group is defined as a 5-tuple $AG =< R, M, rm, RS, GO >$, where $R = \{r_0, r_1, ...\}$ is the set of roles that an agent can play, $M = \{\alpha_0, \alpha_1, ...\}$ is the set of agent members of the *AG* group, $rm \in M$ is the archetypical agent for its group called representative member of the *AG* group (this agent has the best characteristics of its group), $RS = \{rs_i | rs_i = (\alpha_x, \alpha_y), x \neq y\}$ is the set of relationships (peer to peer) among members α_x of the *AG* group, and *GO* is the set of social norms for the *AG* group. This is the disjuntion of the social norms of all α_x that belonging to the group *AG*:

$$GO = \bigcup_{i=1}^{|M|} O(\alpha_i, s);$$

Where $O(\alpha_i, s)$ is the obligation of agent α_i to provide the service s in support of another agent or for itself.

b) **Social relations among agent groups** (SR)**:** a set of relationships (peer to peer) among representative agents *RM* of *AG* groups. This allows interaction among agents belonging to different groups. It is defined as

$$SR = \{srs_i | srs_i = (rm_x, rm_y), x \neq y\}.$$

c) **Set of social norms in the society** (SO)**:** union of the sets of social norms of the society groups, where N is the number of species in the ecosystem and GO_i the set of social norms (in terms of obligations) in each group.

$$SO = \bigcup_{i=1}^{|N|} GO_i$$

4.1.2 Social Interaction Norms

The way agents interact is determined by the set of fundamental social norms regulating the ecosystem model. The enactment of social norms by agents will typically affect and be affected by the local environment and by the other agents in the area. Chapter 5 will address the question of interaction mechanisms between members of the ecosystem (i.e., agents).

4.1.3 Service Ecosystem Environment

There are several definitions and models of the environment, however, for our purposes, the most appropriate definition is porposed by Bossier et al. in [87], where they define the environment as "the place in which it is possible to place services, resources, and institutions, and it must contain support for processes such as service description and discovery of resources." In our case, the service ecosystem environment is the location where it is possible to situate the resources, services and agents; it should provide interaction mechanisms for such processes as service composition and adaptation.

Up to now, our model is restricted to environments that have at least one permanent agent and some temporary agents who join later, at specified -itstart-times, and also leave the environment at the expiration of their *life-times*. We look at these types of open systems initially because in them, the social norms set by *GO* to build agent groups of a service ecosystem can be kept constant (the temporary agents will build groups based on the same *GO* as the permanent ones). In this way, our approach can focus solely on the changes to the overall capacity (resulting from the temporary agents). Consequently, these open environments represent distributed systems in which additional resources might be added to extend agents' functionalities. For this, the ecosystem organization is based on agents' similarities, considering their skills, objectives and locations [88].

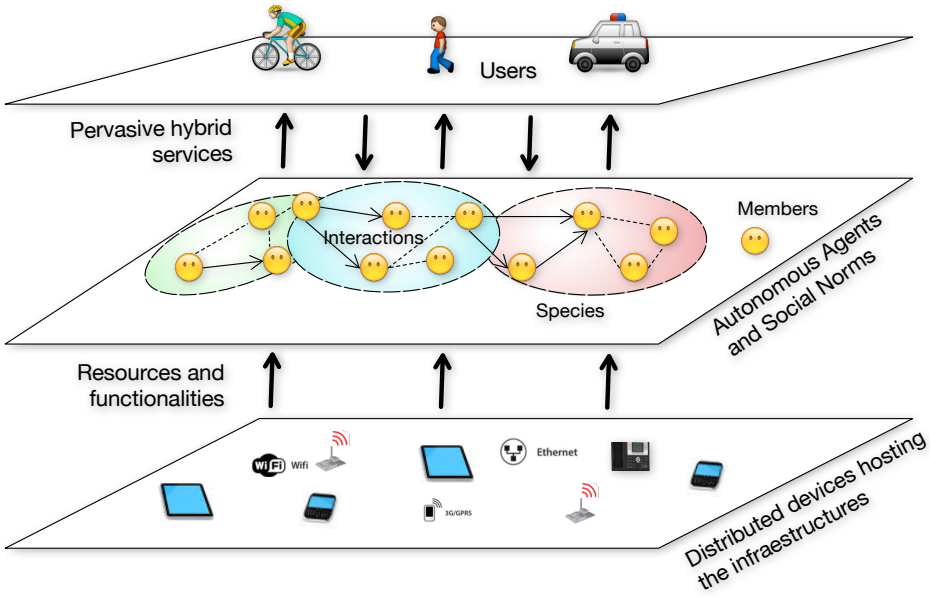

Figure 4.1: A Conceptual Architecture for Pervasive Hardware Service Ecosystems

4.2 A Conceptual Architecture for Pervasive Hardware Service Ecosystems

In order to develop a framework for deploying pervasive hardware services, we have defined a conceptual architecture based on the social metaphor of an ecosystem, an architecture around which to frame the key components of an ecosystem. Figure 4.1 presents a conceptual architecture view of the framework, which establishes a bridge between a group of

distributed heterogeneous devices and the pervasive hardware services requested by users.

The lowest level is the concrete physical and digital ground on which the services will be deployed, i.e., networked computing devices and data sources. Devices and data sources are geographically distributed and interconnected through different type of networks; for instance, devices at the edge can setup direct ad hoc connections without requiring the involvement of a managed network. At the top level, users access the ecosystem in order to consume hardware services, as well as to produce and deploy new hardware services and data in the ecosystem or to make new devices available. At the lowest and top levels openness and dynamics arise: new devices can join/leave the system at any time, and new users can interact with the ecosystem and deploy new services and data items in it. At the middle level are the abstract computational components of the ecosystem architecture.

4.3 Crowd Evacuation: A scenario

In order to fully understand the potential behind the pervasive hardware service ecosystem approach, we now introduce the crowd evacuation case, an application scenario where the proposed ecosystem architecture could potentially be applied.

The application scenario starts with the observation that today's urban landscape is becoming an intricate ecosystem where information, originating from a variety of heterogeneous sources is being gathered, stored, processed, and utilized by pervasive services. In particular, such information is being generated by sensors embedded in the environment, sensors available on mobile devices, mobile services, people on the web, people from the mobile devices, etc. The common denominator of all this information is that it can be linked (either directly or indirectly) to a specific geographical location or region; it is extremely dynamic, since it reflects the social dynamics of an urban environment; and it is vast, as it is originates from a huge number of sources.

Let's now imagine a mobile user moving in a completely new urban setting, such as in an international airport, and looking for the nearest exit because there is fire in the building. This includes information and services related to the specific location of the nearest exits, to the safest routes, etc. Luckily enough, the user has the Crowd Evacuation service installed in his mobile device. The Crowd Evacuation service is a pervasive hardware service that is able to retrieve, in real time, information about the emergency and safe evacuation routes originating from a potentially unlimited number of sources (e.g., sensors, users, web, etc.), to match this information to the user's profile and situation (e.g., user capabilities, location,

age, etc.) and to deliver it in an interactive way. Proposing convenient evacuation routes for the user and the possibility of directing him towards safe locations based on his personal profile, situation and the fire's evolution is one possible example. This scenario requires:

- Real-time data retrieval from a variety of heterogeneous sources.
- Dynamic linkage of user data (e.g., capabilities, age, location etc.) with the available information and pervasive services in the local environment.
- Real-time processing of gathered data in order to extract a high-level interpretation of the information and provide it as an independent service to third parties.
- Dynamic service composition for providing composite services to users.

Combining all these features into a dynamic hardware service delivery platform requires a radical change in the way services are composed and provided at run-time, and the way information is gathered and processed. State-of the-art technologies can barely support all these features, since this requires architectural changes in which the complexity is shared across the many distributed autonomous devices involved in the delivery of a composite service.

Most existing approaches to the case described above provide monolithic solutions, in which a single application is in charge of interfacing with various sources of information as well as implementing the required algorithms for processing this information. Furthermore, limited support exists to fully exploit the environmental data, and to extend them to an enlarged participative approach in which data are spontaneously and anonymously provided by user devices to serve other users.

The proposed hardware service ecosystem starts from these considerations and defines an architecture based on the concepts of autonomous agents, social-norm-based mechanisms, and distributed service composition, this in order to support a dynamic, adaptive and run-time pervasive hardware service mash-up. Such interaction mechanisms, run-time service composition, adaptation and delivery represent the key innovation of the proposed service ecosystem.

4.4 Conclusion

In this chapter we defined a conceptual architecture around which to frame the key components of the service ecosystem inspired in a social metaphor. The social metaphor of

the ecosystem and its elements were introduced. The service ecosystem notion is not new, but previous efforts have generally focused on the context of software services and closed environments where there is dedicated infrastructure such as servers with high processing capabilities and reliable communications networks. In the remainder of this document, a) we define the interaction among ecosystem members by means of social obligations and introduce an agent communication language based on these obligations, and b) we model the hardware service composition process as a distributed constraint satisfaction problem; in addition, we present a mechanism to compose hardware services; and c) we present an extension for modeling the hardware service adaptation process as a dynamic and distributed constraint satisfaction problem.

Chapter 5

Social Obligations for Agent Interaction in the Ecosystem

This chapter presents an Agent Communication Language Agent Communication Language (ACL) based on social obligations for agent interaction that can provide and request hybrid services in open environments. An open environment is a space in which computational devices, including sensors and actuators, can freely enter and leave. Computational devices, which provide services for users, are considered physical agents. Typically, these devices have limited resources to provide their services. In order to extend their resources in open environments, we focus on the design of a service ecosystem based on social interactions among agents. For this purpose, an ACL suitable for open organizations is required. Therefore, we have used a socially inspired approach to propose an ACL based on obligations for physical agents in open environments.

5.1 Agent Interaction

The social ability of the ecosystem' members (agent interaction) is fundamental for them to reach their objectives. The ecosystem should be provided with interaction patterns that direct agents towards the ecosystem's design objectives. Interaction protocols allow agents to do this in a consistent manner. An interaction protocol is a set of rules governing the exchange of messages among agents in order to coordinate their actions for the achievement of their objectives.

There are several formalism to model interaction protocols between agents; however, Deterministic Finite State Machines(DFSMs) and Petri Nets (PN) deserve special attention

due to their mathematical background and graphical representation.

5.1.1 Deterministic Finite-State Machines

A DFSM is a directed graph whose nodes represent states of a protocol, and each edge means the transition from one state to another [89]. Edges are labelled with the messages that agents may communicate; subsequently, depending on the transmission of messages, the interaction state evolves. In addition, a DFSM needs initial and end states, as well a transition function. This function establishes how the protocol evolves, for a given sequence of messages. The use of DFSM has spread widely for the purpose of modeling interaction [90] [91] [92]. Even though DFSMs provide a graphical formalism and a variety of validation tools, statecharts are the most common instrument for modeling interaction protocols. In fact, a modified version of statecharts becomes part of the Unified Modeling Language (UML), which is the specification language used most often for software projects [93].

The disadvantages of DFSMs-based approaches are related to their power of expression, because only regular languages can be represented. This makes DFSMs unsuitable for representing aspects of synchronization required for many interaction protocols.

5.1.2 Petri Nets

PN formalism provides both graphic and mathematical representations. A PN is a directed graph with two types of nodes: transitions and places [61]. Places are connected with transitions by means of output and input edges. Transitions represent the emission or reception of messages. With regard to places, the number of tokens of all places represents the state of the interaction. There are several variants of PN; however the most commonly used to model interaction protocols is the Colored Petri Netss(CPNs). Tokens in a CPN have a color, which is a data type; places only contain tokens of a particular color; arcs have attached expressions, which receive tokens as input parameters and evaluate them in order to enable transitions [94]. Usually, in a CPN that models a interaction protocol, transitions represent the reception and execution of messages, and places provide the interaction state of the protocol[95].

One disadvantage of CPNs-based approaches relates to their lack of specialization in the modeling of interaction protocols. Even for simple interaction protocols, CPNs are hard to read by designers, and for this reason, their further modification and adaptation to

similar protocols is not trivial [96] [97].

In MAS, approaches for modeling agent interactions can be classified as mentalist or socials. The best-known mentalist approach is based on the Beliefs, Desires and Intentions (BDI) agent model [98]. The BDI model has been used by the Foundation for Intelligent Physical Agents (FIPA) to provide interaction semantics and it has been adopted for specifying its agent communication languages [99].

Despite its acceptance in the multiagent research community, proposals based on BDI semantics are only suitable for closed environments [100], where agents are homogeneous with standardized beliefs. To address the open environment issues, there are proposals such as [101], [102] and [103] for the use of ACLs based on social semantics to model interactions by means of social norms [104]. Social norms are public by nature and suitable for open environments; therefore, agents are aware of the norms they have to follow.

Even though there are ACLs based on social norms such as obligations, commitments, prohibitions and permissions, they are implemented following mentalist approaches, which prevents them from being suitable for open environments [100]. Some other proposals focus on solutions that rely on an Internet connection and ignore the physical dependencies of services. We propose a communication language based on social norms for supporting the openness and dynamism of the environment in real scenarios, while preserving the autonomy of agents. Additionally, devices must adapt their services constantly to meet user requirements and preferences, as well as environmental changes. To deal with this, we propose using an agent approach in order to provide devices with autonomy. Devices are able to make decisions and modify their local environment with the least user intervention as possible.

5.2 Obligations

Our service ecosystem is inspired by the social metaphor of a natural ecosystem, where species are represented by agent groups. Thus the interaction among agents is governed by social norms. Social norms can be modeled in terms of commitments, prohibitions, permissions or obligations. Our proposal is based on obligations in order to support agent interaction in open environments. Definitions of obligation, their life cycle and basic operations are provided.

Obligation are social impositions on oneself to provide a service. This notion is similar to those presented in [105] and [106]. Using event calculus formalism, an obligation is

represented as $O(\alpha, \beta)$, which means that agent α is committed to providing service β (this service can be atomic or composed). The motivation of agents to cooperate with each other is based on the following assumption: an agent α with limited resources requires help from another agents (e.g. from agent δ) to achieve its goals. To this end, agent α accepts obligations induced from other agents with the intention of exchanging them for the rights to induce obligations in other agents (e.g., on agent δ). Similarly, if agent α receives support from agent δ, then agent α is bound to assist the agent δ.

In order to manage obligations, a set of basic operations is required. These operations are defined by means of event calculus, which was selected for its intuitive management of events. The main elements of a predicate in event calculus are events and fluents. In our case, events represent sending or receiving messages, or a sensor activation. Fluents are boolean properties that can be affected by events, and their values change over time. Here we used a subset of the event calculus predicates presented by Shanahan in [107]. These predicates are defined in table 5.1.

Table 5.1: Subset of event calculus predicates

Predicate	**Description**
HoldsAt(f, τ)	Fluent f holds at time τ.
Initiates(ε, f, τ)	Fluent f holds after the execution of event ε at time τ
Terminates(ε, f, τ)	Fluent f does not hold after the execution of event ε at time τ
Happens(ε, τ)	Event ε is executed at time τ

For a detailed explanation of the calculation of events predicates the reader is referred to [107] and [108].

5.2.1 Basic Operations on Obligations

Obligations can be created, released and canceled. This notion is akin to the one presented in [109] and [106]. However, assuming that the environment is open, it is unlikely that an obligation will remain static. For this reason we consider it necessary to add support for adapting obligations after they are created. In the following sections, the operations for managing obligations are defined:

5.2.1.1 Create

$O(\alpha, \beta)$, can be created only by the receiver agent (there could be the case of internal messages, where the sender of the message is also the receiver). An obligation is created

when an agent α generates an event ε (for example, a service request), denoted by $\varepsilon(\alpha)$, at time τ. Figure 5.1 illustrates the interaction between agents:

$$CreateO(\varepsilon(\alpha), O(\alpha, \beta), \tau) : \{Happens(\varepsilon(\alpha), \tau) \wedge Initiates(\varepsilon(\alpha), O(\alpha, \beta), \tau)\}$$

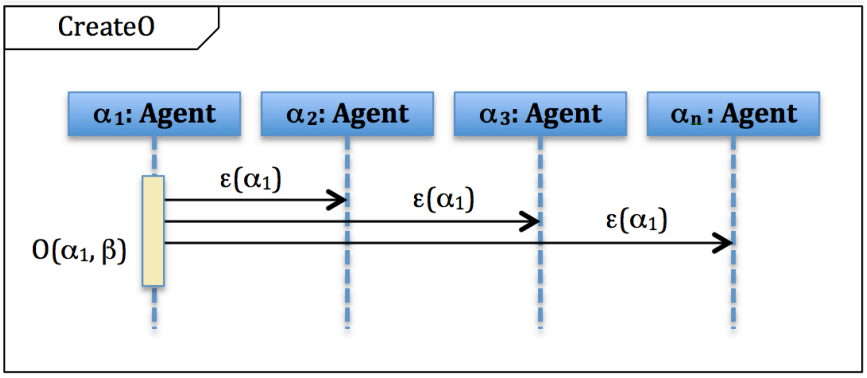

Figure 5.1: Agent α_1 has assumed an obligation β

The occurrence of event ε commits agent α to providing service β at time τ. Obligations can be induced from external agents; however, only the receptor agent can choose to accept (see Figure 5.2) or reject (see Figure 5.3) these obligations; an induced obligation is defined as:

$$CreateIO(\varepsilon(\alpha), IO(\delta, \beta, \omega), \tau) :$$
$$\{Happens(\varepsilon(\alpha), \tau) \wedge Initiates(\varepsilon(\alpha), IO(\delta, \beta, \omega), \tau) \wedge HoldsAt(\delta, \tau)\}$$
$$\text{where } \omega = AllowedBy(\delta, O(\delta, \beta))$$

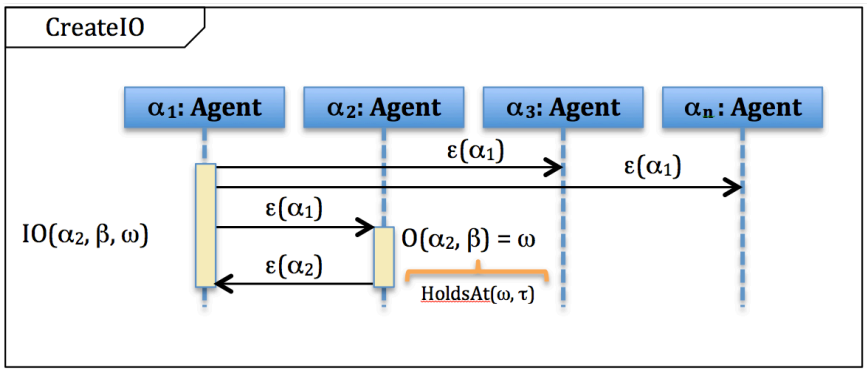

Figure 5.2: Agent α_2 has accepted obligation β induced by agent α_1

Here, agent α induces an obligation in the physical agent δ, adding a necessary condition ω that corresponds to the predicate $AllowedBy(\delta, O(\delta, \beta))$, which must be evaluated by receptor agent δ. The *AllowedBy* predicate means the decision of δ to accept or reject the obligation.

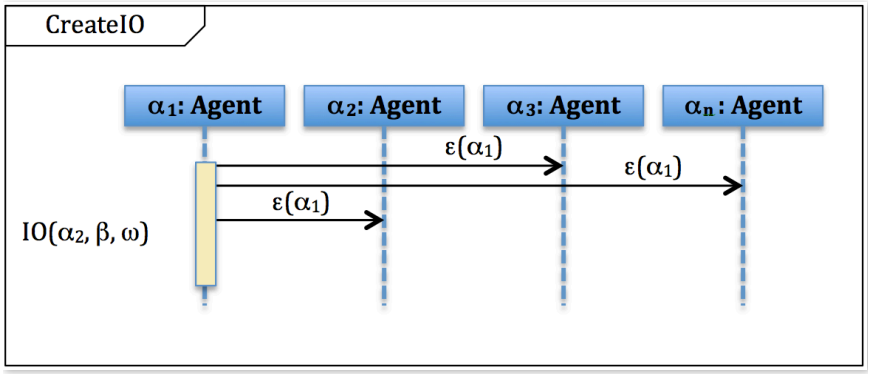

Figure 5.3: Agents have rejected (overlooked) the obligation induced by agent α_1

5.2.1.2 Release

An obligation $O(\alpha, \beta)$ is released by an agent α when service β has been provided. Figure 5.4 illustrates the interaction between agents:

$$ReleaseO(\varepsilon(\alpha), O(\alpha, \beta), \tau) :$$
$$\{Happens(\varepsilon(\alpha), \tau) \wedge Initiates(\varepsilon(\alpha), \beta, \tau) \wedge Terminates(\varepsilon(\alpha), O(\alpha, \beta), \tau)\}$$

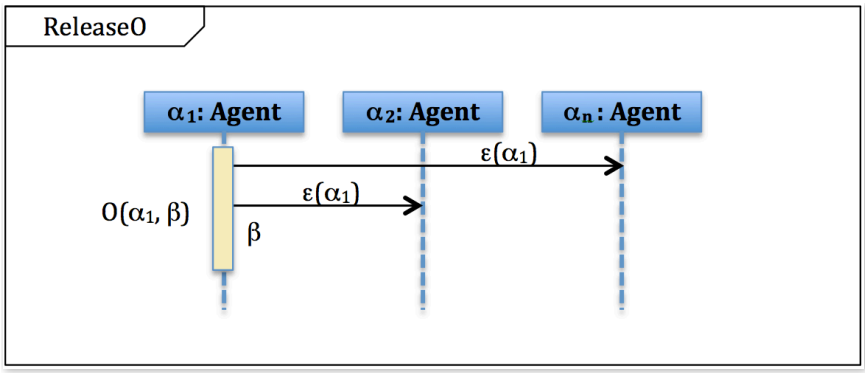

Figure 5.4: Obligation β assumed by agent α_1 is released

The release of an obligation means the occurrence of an event $\varepsilon(\alpha)$ that initiates service β and terminates the obligation $O(\alpha,\beta)$. An induced obligation can only be released by the receptor agent.

5.2.1.3 Cancel

An obligation $O(\alpha,\beta)$ can be canceled by agent α when the obligation is no longer required or the agent is not available to provide service β; however, the cancellation of an obligation must lead to the acquisition of compensatory obligations (Φ) in order to make up for the cancellation of the obligation. In addition, the cancellation of an obligation implies the cancellation of its linked obligations (Γ) that have a dependence on the canceled one. Figure 5.5 illustrates the interaction between agents. This is formally defined as follows.

$$CancelO(\varepsilon(\alpha),O(\alpha,\beta),\tau):$$
$$\{Happens(\varepsilon(\alpha),\tau)\wedge Terminates(\varepsilon(\alpha),O(\alpha,\beta),\tau)\wedge Initiates(\varepsilon(\alpha),\phi,\tau)\wedge$$
$$CancelO(\varepsilon(\alpha),\gamma,\tau)|\gamma\in\Gamma,\phi\in\Phi\}$$

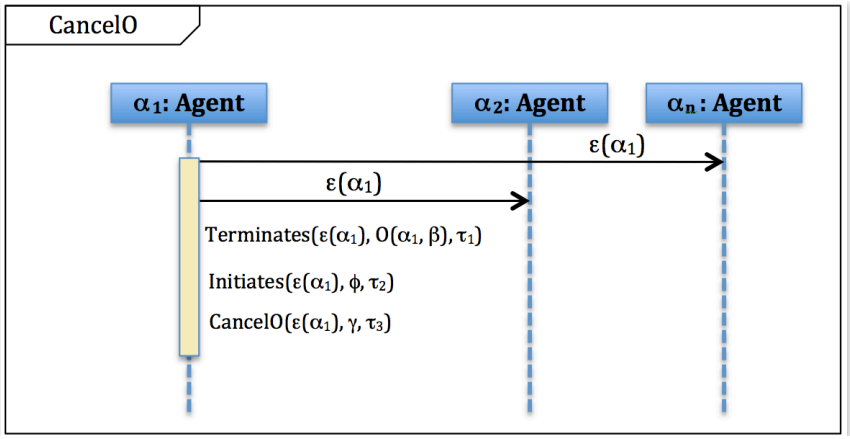

Figure 5.5: Obligation β is canceled by agent α_1

In the case of induced obligations, these can be canceled without collateral effects, because they depend on the physical agents' autonomy. In addition, the cancellation of an induced obligation requires the occurrence of an event $\varepsilon(\alpha)$ that terminates the obligation $IO(\alpha,\beta,\delta)$ (see Figure 5.6); this is defined as follows:

$$CancelIO(\varepsilon(\alpha),IO(\delta,\beta,\omega),\tau):$$
$$Happens(\varepsilon(\alpha),\tau)\wedge Terminates(\varepsilon(\alpha),IO(\delta,\beta,\omega),\tau)$$

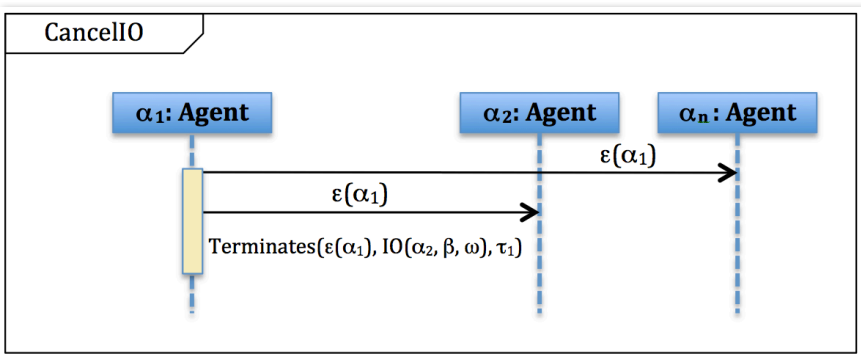

Figure 5.6: Induced obligation β is canceled by agent α_1

5.2.1.4 Adapt

Assuming that the world is dynamic, it is unlikely that an obligation will remain static over time; for this reason adaptable obligations are required. As defined above $O(\alpha,\beta)$ means that agent α is obligated to provide service β. Based on this definition, an adaptation's operation modifies the required service β. In obligation terms, the occurrence of an event ε could commit agent α to adapting its original obligation $O(\alpha,\beta)$ to $O(\alpha,\beta')$ at time τ; however, the adaptation of an obligation could lead to the adaptation of linked obligations (Λ). In addition, the adaptation of an obligation could means the acquisition of new obligations (Ξ) and the cancellation of some obligations(Γ) linked to it. This is formally defined as follows.

$$AdaptO(\varepsilon(\alpha),O(\alpha,\beta),O(\alpha,\beta'),\tau):$$
$$\{Happens(\varepsilon(\alpha),\tau)\wedge CancelO(\varepsilon(\alpha),O(\alpha,\beta),\tau)\wedge Initiates(\varepsilon(\alpha),O(\alpha,\beta'),\tau))\wedge$$
$$AdaptO(\varepsilon(\alpha),\lambda,\beta'(\lambda),\tau)\wedge Initiates(\varepsilon(\alpha),\xi,\tau)\wedge CancelO(\varepsilon(\alpha),\gamma,\tau)|\lambda\in\Lambda,\xi\in\Xi,\gamma\in\Gamma\}$$

An $O(\alpha,\beta)$ can be adapted only by the agent α that has assumed this obligation. However agent α can request the adaptation of induced obligations on another agent δ. In the case of the adaptation of conditional obligations, these can be adapted without collateral effects, because they depend on the agent autonomy; this is defined as follows:

$$AdaptIO(\varepsilon(\alpha),IO(\delta,\beta,\omega),IO(\delta,\beta',\omega),\tau):$$
$$\{Happens(\varepsilon(\alpha),\tau)\wedge CancelIO(\varepsilon(\alpha),IO(\delta,\beta,\omega),\tau)\wedge CreateIO(\varepsilon(\alpha),IO(\delta,\beta',\omega),\tau)\}$$

A set of messages is not enough to enable agent cooperation, an agent communication language is required. The next sections explains the obligation life cycle and basic acts of the agent communication language.

5.3 Obligation Life Cycle

As with other approaches, an interaction mechanism based on social norms expresses both static and dynamic aspects of systems. The dynamic aspect arises from the interpretation of such interaction mechanisms and the consequent acts on obligations by each agent in the system. Thus, obligations have a life cycle in runtime. In previous works, as in [106], authors consider that obligations may be acquired, released or canceled. That is, an obligation can be in one of three possible states.

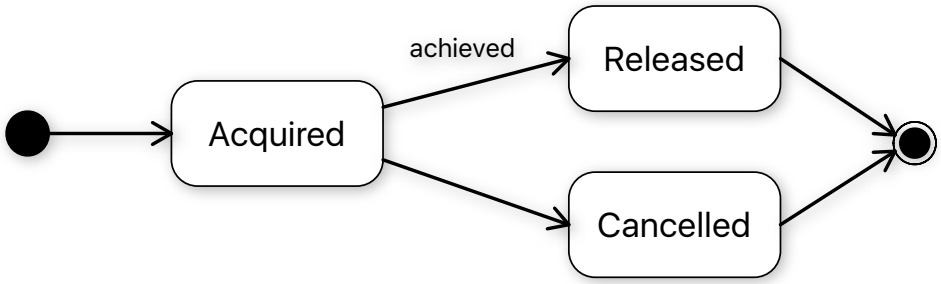

Figure 5.7: Three-state model for obligations

In this model based on three states (see Figure 5.7) an obligation is created when the condition for assuming it is true. Once the obligation is created, the state changes to *released* when the obligation is achieved by the agent. The obligation state changes to *canceled* when the agent is not able to achieve the obligation.

Other authors have proposed early obligation life-cycles (e.g. [110]) based on four states: active, fulfilled, unfulfilled, and inactive . Thus, the life cycle of an obligation is based on four states (see Figure 5.8). An obligation is created when the condition of a rule is satisfied, and its initial state is *active*. The state becomes $fulfilled$ when the obligation is achieved before the deadline. However, if the obligation is not achieved by the agent before the deadline, then the status changes to $unfulfilled$. Finally, the state of the obligation changes to *inactive* when its trigger condition ceases.

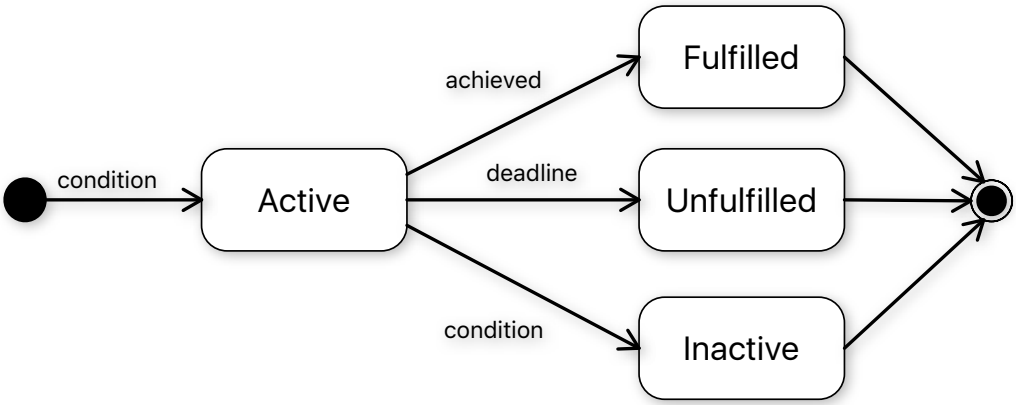

Figure 5.8: Four-state model for obligations [110]

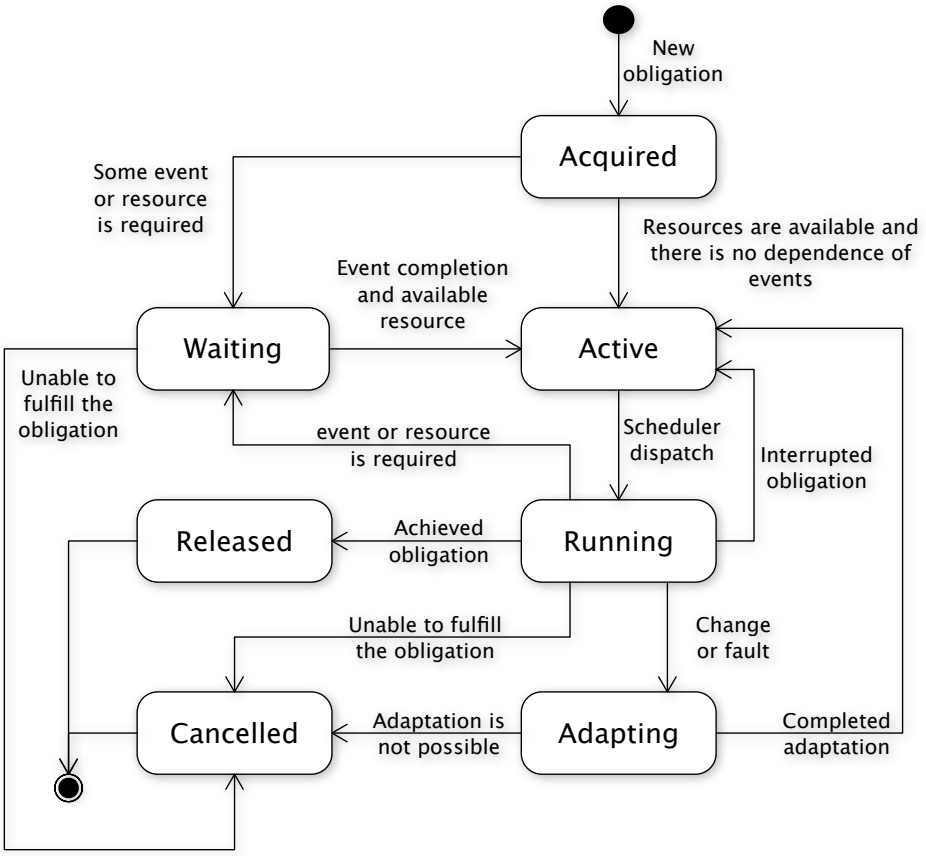

Figure 5.9: Proposed model for obligation

Although the models described above define different states that allow an agent to manage its obligations, they are not appropriate when considering the management du-

ties within dynamic ecosystems in which the agents need to share resources in a dynamic world.

In our case, based on the fact that an obligation means providing a service and a service represents a set of functionalities to be performed in some order, then obligations cannot be atomic. Furthermore, after an obligation is acquired and before it is released, there may be changes in the environment and user requirements. Additionally, agents can enter and leave the environment and even the availability of resources in agents may change. Therefore, it is necessary to consider various intermediate states in the life cycle of an obligation. Taking inspiration from the scheduler of an operating system for concurrent operations, we propose a life cycle for obligations with seven possible states as defined in Figure 5.9 and explained below.

1. An obligation $O_i(\alpha, \beta)$ for providing a service is *acquired* and created when an agent α assumes the obligation.

2. Once the obligation is created, its initial state becomes *active* if the resources required to meet the obligation are available and there is no dependency on some event. Otherwise, the initial state of the obligation changes to *waiting*.

3. When an obligation's state is *waiting*, it can change to *active* if the resources required to meet the obligation become available and there is no dependency on some event, and *cancelled* if the agent is unable to fulfill its obligation before the deadline.

4. The obligation's state changes from *active* to *running* according to a policy for dealing with obligations; in our case, for the sake of simplicity we have chosen a non-apprehensive First Input, First Output (FIFO) policy.

5. Once the running time for the obligation has finalized, the execution is interrupted and the obligation's states changes to *released* if the obligation was achieved before its deadline, *active* if the obligation has not yet been achieved within its time limit, *cancelled* if the agent was unable to fulfill its obligation before the deadline, and *adapting* if there was a change in the environment or a failure in the system.

6. The obligation's state changes from *adapting* to *cancelled* if the adaptation was not possible, *active* when the adaptation is completed.

7. Obligation $O_i(\alpha, \beta)$ ends its life cycle in runtime when its state becomes *released* or *cancelled*.

Unlike the models proposed by other authors, our proposal, inspired by early models such as [110] and [106], defines a lifecycle for obligations that supports the use of shared resources in dynamic environments. The objective sought with our model is to reduce the time required to reach multiple obligations within an ecosystem of services where resources can be shared and their availability is not guaranteed. This is achieved by allowing the agents to interact and make collective decisions based on possible intermediate obligation states (i.e. obligation states after being acquired and before being released). In order to provide the proposed life cycle with support, we have implemented a Java package for handling obligations, to be used by subsequent deployments agents (see the conceptual model in Figure 5.10).

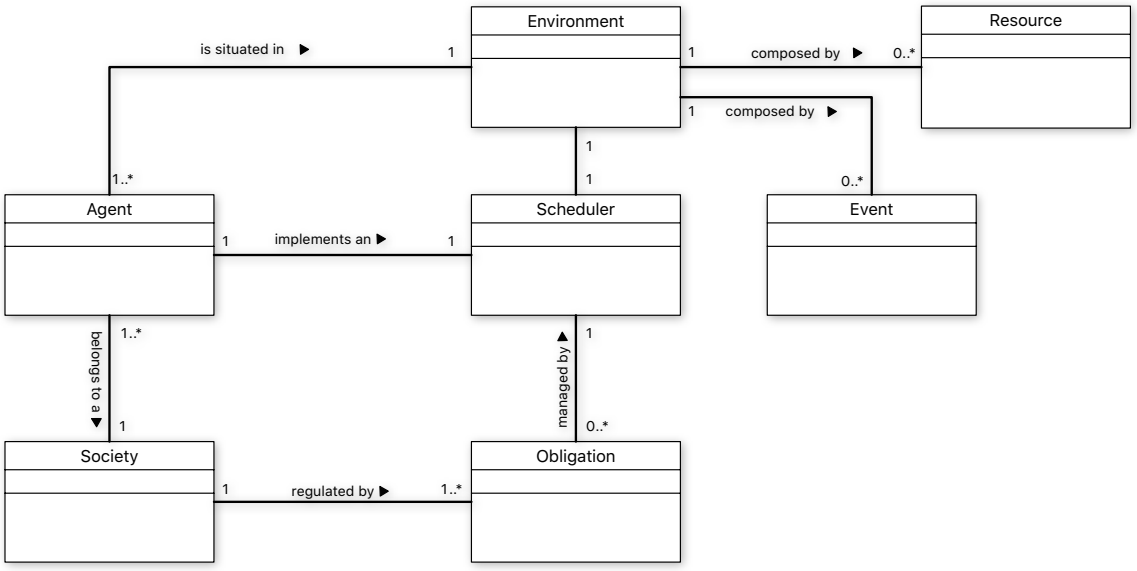

Figure 5.10: Conceptual Model for Managing Obligations

5.4 Basic Acts for a Social ACL

In this section, from the notion of obligation we derive a set of basic communicative acts for a social agent communication language. The obligation-based agent communication language is derived from speech act theory [111], comparable to the ones presented in (Colombetti, 2000), (Fornara and Colombetti, 2002), and (Singh, 1999). The illocutionary acts are defined by means of operations over the obligations of agents. The basic communicative acts are categorized as assertive, directive, commissive and declarative. Some FIPA-ACL primitive acts (FIPA, 2002b) are used, but their BDI definition is replaced by

a definition based on obligations, which does not depend on the beliefs of agents. In the following sections communicative acts are defined using the operations over obligations presented in the previous section.

5.4.1 Assertive Acts

This type of act attempts to convince the receptor agent of a message that some fluent β was achieved by the sender agent, i.e. an obligation was released. $Inform$ is our prototype of assertive acts and it is defined as follows:

$$InformO(\alpha,\delta,\beta,\kappa) \equiv$$
$$ReleaseO(InformO(\alpha,\delta,\beta,\kappa),O(\alpha,\beta),\tau)$$

The symbols α and δ represent the sender and receptor of the message, respectively; β means the service that was provided; the symbol κ is a set of domain data that results from the performance of β. The message $Inform(\alpha,\delta,\beta,\kappa)$ means that the sender releases a previously assigned obligation to provide service β.

5.4.2 Directive Acts

This type of act attempts to induce obligations in other agents. We are considering two cases: a) an agent α requests another agent δ to accept an obligation to provide a service β or b) an agent α requests another agent δ to accept one change of an obligation previously induced to provide service β'. This is formalized using induced obligations, where only the receptor agent can choose to accept or reject the creation of an obligation or the adaptation of an induced obligation. This is stated with the predicate $AllowedBy$. $RequestO$ and $RequestA$ are our prototypes of directive acts their definitions are as follows:

$$RequestO(\alpha,\delta,\beta) \equiv$$
$$CreateIO(RequestO(\alpha,\delta,\beta),IO(\delta,\beta,\omega),\tau)$$
$$|\omega = AllowedBy(\delta,O(\delta,\beta))$$

β is service to be induced as an obligation. The message $RequestO$ means that an induced obligation is created; however, the obligation is only active if the agent δ (receptor) chooses to assume the obligation, i.e. agent δ evaluates the $AllowedBy$ predicate as $true$.

$$RequestA(\alpha,\delta,\beta,\beta') \equiv$$
$$AdaptIO(RequestA(\alpha,\delta,\beta,\beta'),$$
$$IO(\delta,\beta,\omega),IO(\delta,\beta',\omega),\tau)$$
$$|\omega = AllowedBy(\delta,O(\delta,\beta'))$$

β is a service of a previously induced obligation that must be adapted to provide service β'. The message *RequestA* means that an induced obligation is adapted; however, the adapted obligation is only active if the agent δ (receptor) chooses to assume the adaptation of the obligation, i.e. agent δ evaluates the *AllowedBy* predicate as $true$.

5.4.3 Commissive Acts

These types of acts are promises to provide a service. Commissive acts are part of small protocols, where an agent α sends a *RequestO* or *RequestA* message to another agent δ and the obligation is accepted or rejected according to decision ω of agent δ. To express this, four communicative acts are defined:

$$AgreeO(\delta,\alpha,\beta) \equiv$$
$$CreateO(Agree(\delta,\alpha,\beta),O(\delta,\beta),\tau) \wedge HoldsAt(AllowedBy(\delta,O(\delta,\beta)) = true,\tau)$$

$$RejectO(\delta,\alpha,\beta) \equiv$$
$$CancelIO(Reject(\delta,\alpha,\beta),IO(\delta,\beta,\omega),\tau) \wedge HoldsAt(AllowedBy(\delta,O(\delta,\beta)) = false,\tau)$$

After the reception of a *RequestO* message that attempts to induce an obligation, the receptor agent can send an *AgreeO* (agent δ accepts the induced obligation) or *RejectO*(agent δ rejects the induced obligation) message. The *AgreeO* message means the induced obligation is activated in the sender agent and an obligation is created in the receptor agent. If agent δ rejects the induced obligation, it sends a *RejectO* message that cancels the induced obligation in the sender agent.

$$AgreeA(\delta,\alpha,\beta,\beta') \equiv$$
$$CreateO(Agree(\delta,\alpha,\beta,\beta'),O(\delta,\beta'),\tau) \wedge HoldsAt(AllowedBy(\delta,O(\delta,\beta')) = true,\tau)$$

$$RejectA(\delta,\alpha,\beta,\beta') \equiv$$
$$CancelIO(Reject(\delta,\alpha,\beta),IducedO(\delta,\beta,\omega),\tau) \wedge CancelO(Reject(\delta,\alpha,\beta),O(\delta,\beta),\tau) \wedge$$
$$HoldsAt(AllowedBy(\delta,O(\delta,\beta)) = false,\tau)$$

After the reception of a *RequestA* message attempting to induce the adaptation of an obligation, the receptor agent can send an *AgreeA* (agent δ agrees to adapt a previously induced obligation) or *RejectA*(agent δ refuses to adapt a previously induced obligation) message. The *AgreeA* message means the induced obligation adaptation is activated in the sender agent and an obligation is adapted in the receptor agent. If agent δ rejects the induced obligation adaptation, it sends a *RejectA* message that cancels the induced obligation in the sender agent and also cancels the previously assumed obligation in the receptor agent.

5.4.4 Declarative Acts

These types of acts are useful for making self-induced obligations public, i.e. declarative acts notify other agents about the creation of self-induced obligations. Two primitive acts are considered:

$$ConfirmO(\alpha,\delta,O(\alpha,\beta)) \equiv$$
$$CreateO(Confirm(\alpha,\delta,O(\alpha,\beta)),O(\alpha,\beta),\tau)$$

An agent α can sends a *ConfirmO* message to another agent δ to declare its obligation to achieve a state of affairs. However, if some obligation was already released, agent α sends a *DisconfirmO* message.

$$DisconfirmO(\alpha,\delta,Obligation(\alpha,\beta)) \equiv$$
$$ReleaseO(Disconfirm(\alpha,\delta,O(\alpha,\beta)),O(\alpha,\beta),\tau)$$

5.5 Modeling a Protocol by Means of Obligations

This section describes the implementation of an interaction protocol by means of obligations enabling agent organization. The protocol is based on the acts described earlier in this chapter. This protocol is based on the following idea: members of same species work together to achieve common goals. We propose building species of agents (for the shake of simplicity, each agent α represents a device) to provide services and achieve common aims. However, the physical environment and functional requirements are constantly changing; therefore it is false that the agents' organization in an ecosystem must remain static. For this reason we propose a clustering algorithm based on skills and locations to build initial species, where each cluster is adapted on the basis of the current aims of each agent.

In the design of wireless networks, it is often necessary to connect the whole network using the least amount of resources. However, our objective is to create suitable network connections for pervasive hybrid services. Cluster-based control structures allow a more efficient use of resources because a hierarchical view of the created network through clustering decreases the complexity of the procedure for creating network. This is especially true in mobile and ad hoc networks made up of a large number of individual devices.

On a topology level, our clustering is done by grouping devices (sharing current aims and with similar skills) inside a certain transmission area (see figure 5.11). Members α of each group AG_i can play different roles such as *participant* and *representative*.

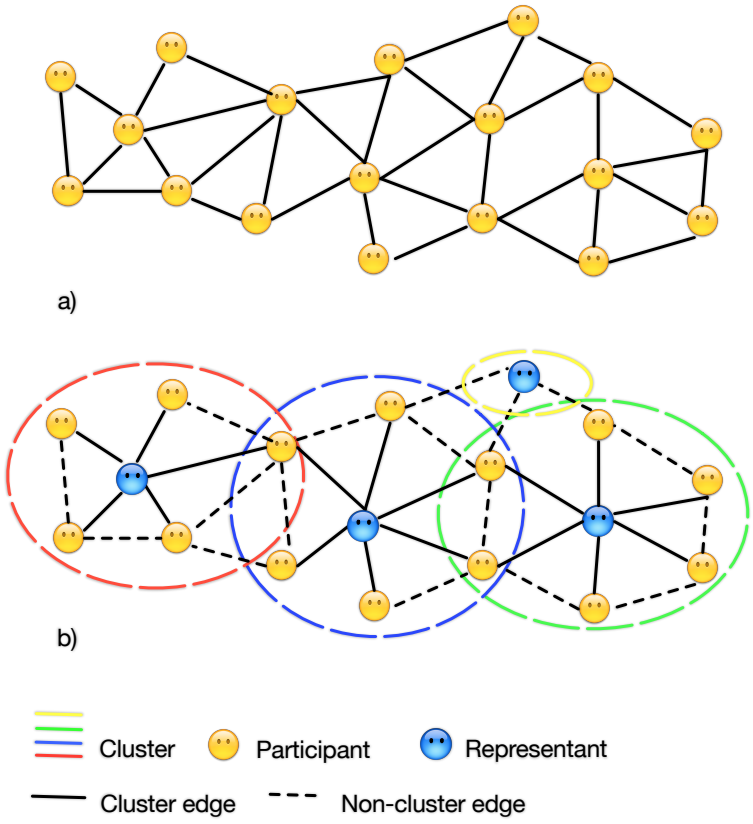

Figure 5.11: Clustering approach - a) Original connection graph, b) Possible organization of species

5.5.1 Preliminaries

Each participant is instantiated and represented by an agent; this agent has a unique identifier denoted by α. Each α has a set of skills to provide one or more services. In this

paper we assume the emergence of clusters based on a minimal percentage of similarity in three dimensions: skills, locations and current aims. In the ecosystem of services, these clusters mean agents working to provide services and achieve common aims. However, these clusters are not fixed and may change over time.

In this work, it is beyond our scope to derive similarity functions. In this regard for the shake of simplicity we assume the existence of three functions: (i) a similarity function based on agents' skills that enables agents to determine distances between them, (ii) a function to determine a similarity percentage between two aims and (iii) a function to determine the geographical distance between two agents α and δ. Each cluster in the ecosystem should have a representative participant agent α_R and some participant agents α. The choice of the α_R is based on the skills associated with each agent α and its location; the agent α with best skills to represent a cluster plays the α_R role. In order to achieve an appropriate partition of initial clusters in the ecosystem, the clustering process must satisfy the following properties:

- Each agent α has at least one agent α_R as a neighbor (two agents α and δ are neighbors if they belong to the same cluster).
- Each agent α must affiliate with the neighboring agent α_R that has the greatest similarity to it, based on skills, location and current aims.
- Two agents α_R cannot be neighbors.

The clustering process is executed in all agents and each one decides its own role (α_P, α_C or α_R); depending only on the neighbors' decisions. Thus initially, only the agent α with the best skills will broadcast a message to its neighbors stating that it will be the α_R. When one or more of these messages are received, agent α will choose to join the cluster of the agent α_R with the best skills. If any message has been received by agent α from another agent δ with higher skills, then α will send a message to promote itself as the new α_R.

5.5.2 Clustering-Based Protocol

The process of creating and adapting clusters in the ecosystem, is driven by messages: a specified behavior will be executed by the agents depending on the reception of the corresponding message. The main messages used by agents in the clustering process are:

- $InformO(\alpha, \delta, ChangeRPA, \kappa)$ is used by an agent α to inform its neighbors that it is going to be the agent α_R.

- $InformO(\alpha, \delta, GoInto, \kappa)$, with which an agent α communicates to its neighbors that it will be part of a cluster whose agent α_R is a neighbor.

- $InformO(\alpha, \delta, CurrentAim, \kappa)$ is used by an agent α to inform its neighbors that it has a new aim. If its neighbors' aims are similar to its current aim, then agent α remains in the current cluster. Otherwise α finds, in its vicinity, another agent α_R with goals similar to its current aim. An agent α may have one or more current aims; it can thus belong to more than one cluster.

- $InformO(\alpha, \delta, AddRPA, \kappa)$ is used by an agent α to inform its neighbors that it will be part of other clusters, but remain in the current cluster too.

- $InformO(\alpha, \delta, NewPA, \kappa)$ is used by an agent α that has come to the ecosystem. These messages start the clustering process based on three dimensions: skills, location and aims. The priority of each dimension can be adjusted depending on the problem domain.

- $InformO(\alpha, \delta, LostPA, \kappa)$ is sent when an agent α detects a link failure with another agent δ (we assume the existence of a low-level function to detect link failures).

All species in the ecosystem are dynamic and each agent α can belong to one or more species at one time.

5.6 Conclusion

In this chapter, we extended the notion of *obligation* as social norm (presented in [103]) and introduced the adaptation of obligations and their life cycle. We defined an obligation-based agent communication language, made up of a set of elocutionary acts with semantics expressed in terms of obligations. The aim of the ACL is to provide semantics according to the kind of message sent, and to establish an explicit link between messages and the domain knowledge attached to them.

There are some approaches that are close to the one presented in this chapter. The closest one is presented in [112], where a commitment-based ACL is defined, making use of commitment objects to provide semantics for speech acts. Elocutionary acts are also based on Searle's taxonomy of acts [111]. The assertive speech act (inform) and the main commissive speech act (promise) have the same explicit definition, leaving their differences

to the interpretation (i.e., there are ambiguous definitions). Another difference is the definition of a promised primitive act for commissives, while here, comissive acts are managed as responses to the reception of directive acts.

Another approach that proposes commitment semantics for agent communication languages is presented in [113]; there, a commitment requires the participation of three agents: a debtor, a creditor, and an authority, that validates the commitment. This meaning of commitment has similarities with the concept of obligation that we present in this chapter, given that obligations are self-commitments. In our approach, the debtor and the creditor are represented by the same agent. This extends the scope of the validity of obligations to all the contexts of the agent's interaction and not only to the context of its creditor, as in commitments. Nevertheless, obligations require external mechanisms to guarantee the fulfillment of the agents' obligations. These mechanisms could be provided by the proprietor of the organization (i.e., the representative agent).

Chapter 6

An Approach for Pervasive Hardware Service Composition and Adaptation

Hardware services composition in ecosystems enables devices to use resources in the local environment in order to provide pervasive services. Current work in the development of service ecosystems on mobile and ad hoc networks has yet to address the dynamic composition of hybrid services. To address this issue, in this chapter, we model the service composition as a distributed constraint satisfaction problem and a service relevance model. Simulation results show the performance of our protocol in terms of messages and composition time. Finally an illustrative study case is discussed.

6.1 Pervasive Hardware Service Composition

An open challenge in achieving an ecosystem of pervasive hardware services is how to allow for the automatic composition of hardware services (based on the services available in the user's local environment), in order to fulfill user requirements with limited human intervention. Thus, the aim is to automate the composition process through the discovery of new hardware services and to determine the required services based on current conditions of the local environment. In this chapter we examine the possibility of performing hardware service composition by modeling and solving it as a DisCSP.

Unlike traditional schemes, DisCSPs can be solved without the need for agents to directly divulge complete and precise information about their domain and constraints. This is relevant when it comes to hardware service composition for privacy and security reasons, as there may be information pertinent to composition that an agent does not wish to

divulge.

The resulting contributions of this Chapter are a DisCSP model for hardware service composition in ecosystems based on MANETs, and a hardware service composition protocol utilizing an asynchronous backtracking-based algorithm for solving the respective DisCSP.

The rest of this chapter presents a model of the service composition process as a DisCSP and DynCSP problem, and describes our proposed protocol for performing the service composition as well as our heuristic for service adaptation.

6.2 Problem Formulation

We start by considering an ecosystem composed of mobile and fixed devices. These devices are modeled by agents providing one or multiple services, able to be invoked by peer agents. Deploying pervasive hardware service involves the composition of hardware services provided by devices in the user's local environment and offers a pervasive hardware service to the user, satisfying an atomic task. Starting from a *task* we generate a *STask* as 3-tuple $\langle TR, TC, TP \rangle$ where $TR = \{tr_i | i = 1, ..., n\}$ is the finite set of services needed to fulfill the *task*; tr_i is a service requirement; $TC = \{tc_i | i = 1, ..., m\}$ represents a finite set of constraints over the set TP to allow the intended behavior of a composed pervasive service; and $TP = \{tp_i | i = 1, ..., q\}$ is a finite set of task constraints managed by the user.

Our objective is to examine possible composition configurations in a user's local environment and to identify the configuration that satisfies all service requirements, adheres to all service constraints, and best caters to the user's preferences. Complicating this process is the fact that agents are autonomous and may be operating under different objectives and behaviors used in making composition decisions. This makes centralized solutions or methods involving high amounts of information sharing unsuitable for solving this problem. An acceptable solution is a distributed one, utilizing minimal information sharing and providing efficient negotiation techniques.

6.3 Service Composition as a DisCSP Problem

Satisfying the objective outlined in the previous section, we model pervasive hardware service composition in the form of a distributed constraint satisfaction problem *disCSP*. A *disCSP* is represented as a 4-tuple $\langle X, D, C, A \rangle$ where $X = \{x_i | i = 1, ..., n\}$ is a finite set of

variables; x_i is a variable corresponding to a service needed to satisfy a particular user requirement; $D = \{D_i | i = 1, ..., n\}$, such that for each x_i there is a service option domain D_i; C is a finite set of constraints $\{C_1, C_2, ..., C_m\}$ of the user requirement; and A is a set of agents over which the variables and constraints are distributed. At any agent, the set of possible values for variable x_i is its current domain D_i. Figure 6.1 shows a visual abstraction. The matrix *disCSP* is constituted by agents α_i and required services x_j; each cell represents a service that each agent α_i can provide and its relevance.

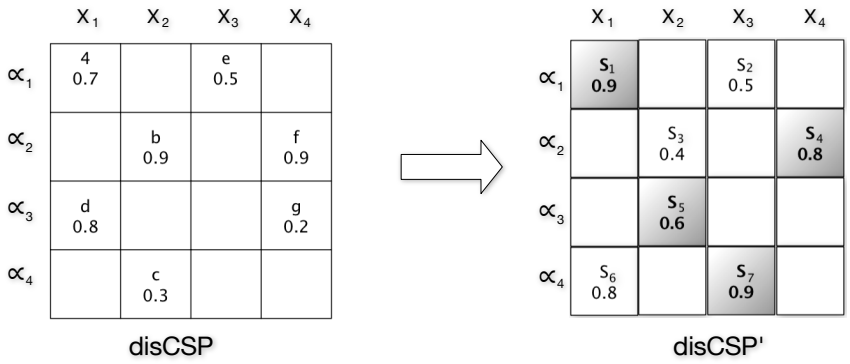

Figure 6.1: Matrix representation of a disCSP

Having identified our constraint satisfaction problem, we apply an algorithm based on the asynchronous backtracking algorithm [65] to solve the pervasive hardware service composition problem. That is, we transform the matrix *disCSP* into *disCSP'*).

6.4 Pervasive Hardware Service Composition

6.4.1 *Dynamic-disCSP* Framework

From our perspective, pervasive hardware services must be user-centered, with the services remaining available despite user mobility through heterogeneous environments. Our objective is to examine possible pervasive service configurations in the user's vicinity and to identify a configuration that satisfies user requirements, and fulfills task constraints and user preferences. Figure 6.2 shows the conceptual model of an ecosystem constituted by devices (fixed and/or mobile) modeled by agents, connected by an ad hoc network and providing one or more services (each of these services can have constraints) that may be requested by peer agents, that is, with the aim of fulfilling one or more tasks (each task can have requirements and preferences).

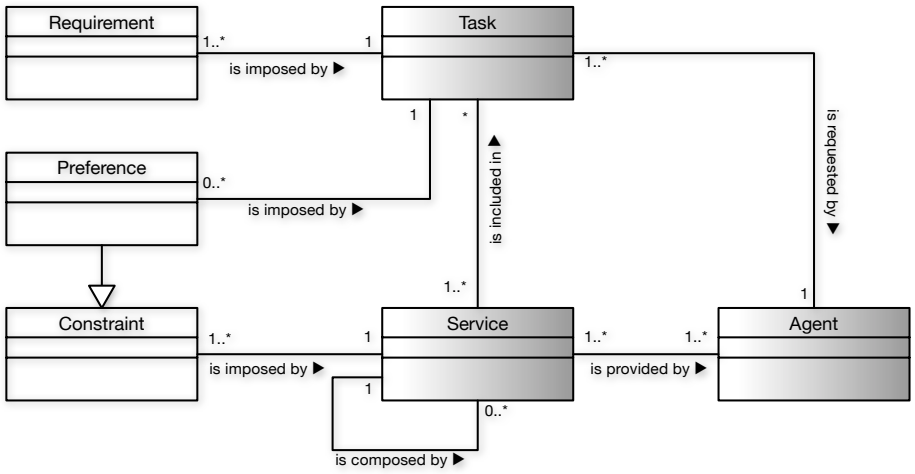

Figure 6.2: Conceptual model of a pervasive service ecosystem context

Below we describe a protocol for the composition of pervasive hardware services. This protocol comprises two phases: candidate agent formation and service composition.

6.4.2 Formation of Candidate Agents

The objective of the formation phase is to identify potential participant agents for the service composition in the user local's environment. Figure 6.3 shows an abstract scenario modeled by agents devices. At this stage it is worth mentioning the existence of a maximum transmission range of the initiator (i.e. the agent that requires a service composition). All agents that are within this range are possible participants in the construction of the solution. However, it must be considered that both the initiator and the participating agents can move, so the pool of potential agents that can contribute to the solution may change. Therefore, the solution must be adaptable (adaptation is discussed in future sections).

When a user requirement becomes active in the user device (for example, in figure 6.3 a user requirement becomes active in the device managed by agent α_0), the agent sends a task request (*TaskRequest*) message containing the *STask* description to systems in the user's environment. The *STask* is used by receiving agents to decide whether they can contribute at least one service to fulfill the *STask*. Agents that may contribute send a *TaskReply* message to the system initiator. With the *TaskReply* messages the agent initiator builds a candidate agent list, where each item of the list has its address and its relevance factor to fulfill the *STask*. Once the list is made, the initiator agent sends it to all participating agents. Figure 6.4 shows a possible message sequence example where a user requirement

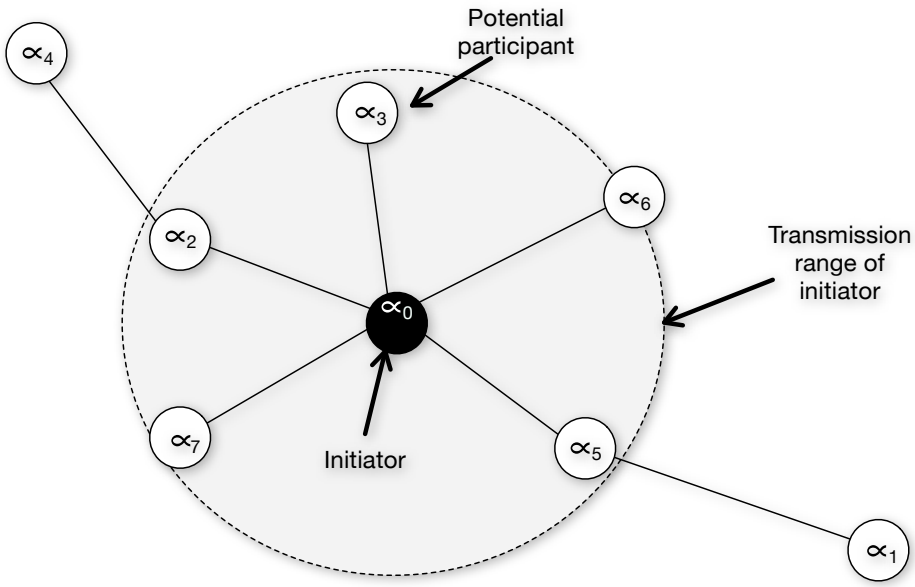

Figure 6.3: Agents in the local environment of the α_0 agent (considering one hop)

becomes activated at agent α_0.

The relevance $RL_i(\alpha_i)$ means the appropriateness of a service provided by an agent. The relevance is determined by each agent α_i, using evaluation function 6.1 to determine the relevance of services provided by agents that model non-mobile devices, and evaluation function 6.2 for services provided by agents that model mobile devices.

$$RL_1(\alpha_i) = K_1 \cdot R(\alpha_i) + K_2 \cdot \frac{1}{WL(\alpha_i)} \tag{6.1}$$

$$RL_2(\alpha_i) = K_1 \cdot \frac{1}{S(\alpha_i)} + K_2 \cdot R(\alpha_i) + K_3 \cdot \frac{1}{WL(\alpha_i)} \tag{6.2}$$

$K_j \in \mathbb{R}$ is a constant for tuning the relevance factors based on specific applications; its value is limited by $0 < K_j < 1$. $S(\alpha_i)$ is the device's speed (in the case of mobile devices) and it is inversely proportional to its relevance factor (i.e. devices with high speed have low relevance for the service composition). $R(\alpha_i)$ represents the amount of the agent's resources (depending on a specific application, each kind of resource may have a different relevance), and $WL(\alpha_i)$ means the current work load of α_i; it is inversely proportional to its relevance factor. The relevance factors can vary based on the application domain. The

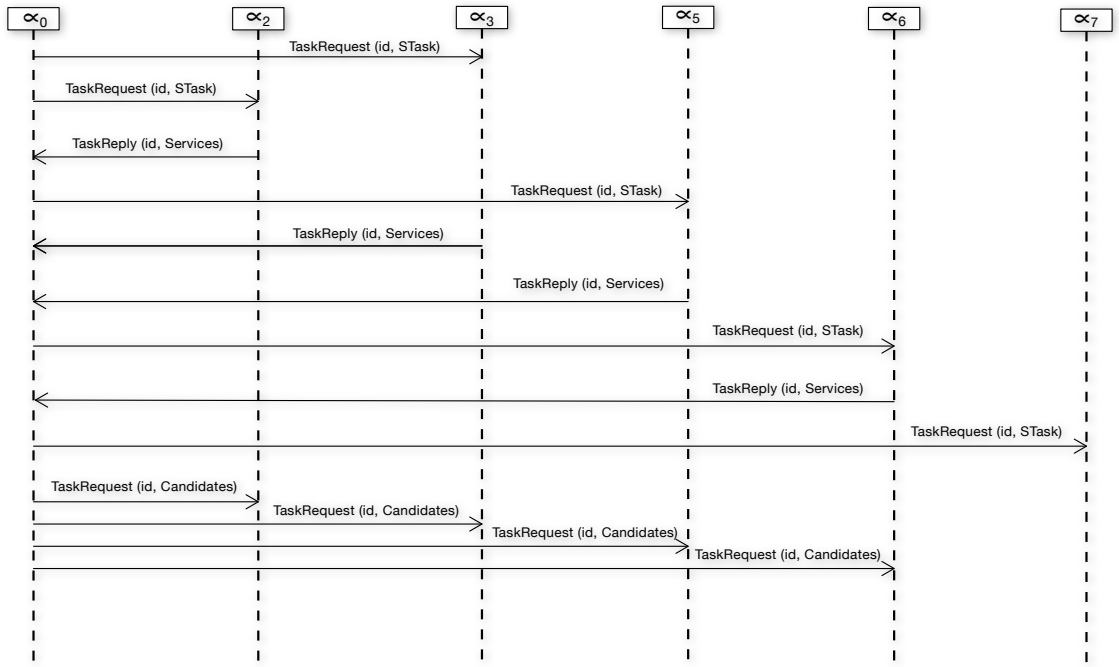

Figure 6.4: Candidate agent formation

relevance factor $RL(\alpha_i)$ has two purposes: to improve the participant selection and to build the agent order that will be required in the composition and adaptation phases.

6.4.3 Pervasive Hardware Service Composition Algorithm

Once the initiator agent has the list of participating agents, we apply the algorithm for pervasive hybrid service composition (PHSC). This is based on the ABT algorithm used for resolving distributed constraint satisfaction problems [114] [66]. Like several algorithms for solving DisCSP, it requires a total ordering of the relevance of participating agents. For each $\alpha_i \subseteq A$, agent α_j has a higher relevance than α_i if it appears before α_i in the total ordering (on the candidate agent list). On the other hand, α_j has a lower relevance than α_i if it appears after α_j in the total ordering (on the participating agent list). So, the total order classifies all neighboring participating agents of α_i, $N(\alpha_i)$, into higher relevance neighbors, $N^+(\alpha_i)$, and lower priority neighbors, $N^-(\alpha_i)$. In the real world, communication among devices is not necessarily FIFO; therefore we used a time-stamp that is incremented only if α_i changes its assignments (thus each assignment has a label).

In order to solve a DisCSP, agents α_i generate locally consistent assignments and exchange their new proposals with their neighbors $N^-(\alpha_i)$ to achieve a global consistency. As

in the ABT algorithm, each α_i stores assignments received from its neighbors in its agent view and a list of no-goods. α_i stores in its agent view the most up-to-date assignments of its higher priority neighbors. α_i stores in its no-good list no-goods justifying the removal of values.

The main elements of the PHSC protocol that implement the idea set forth above are described. In the initial procedure *phsc*(), each α_i assigns a value to its variable and informs its lower neighboring agents. Then, it loops in order to process the received messages.

```
procedure phsc(participantAgents)    do
  end ← false, a_i ← null, t_i ← 0;
  checkAgentView();
  while (!solution) do
      msg ← getMessage();
      switch (msg.type) do
          chk:   processAssign(msg);    stp: end ← true;
          ngd:   resolve(msg)
  end while
end procedure
```

Procedure *checkAgentView* checks whether the current value (a_i) is consistent with the *AgentView*. If a_i is inconsistent with assignments of higher priority neighbors, α_i tries to select a consistent value. During this process, some values from $D(x_i)$ may appear as inconsistent. Thus, no-goods justifying their removal are added to the no-good list of α_i. When two no-goods are possible for the same value, α_i selects the best no-good. If a consistent value exists, it is returned and then assigned to a_i. Next, α_i informs all agents in $N^-(\alpha_i)$ about its new assignments through *chk* messages.

```
procedure ckeckAgentView()    do
  if isConsistent(AgentView, a_i )!=true then
      a_i ← newAssignment();
      if a_i ≠ null then
          for each (s_k ∈ N⁻(s_i)) do
              sendMessage(chk, Assignment⟨ s_i, a_i, t ⟩, s_k);
          end for
      else
          backTrack();
      end if
  end if
end procedure
```

Otherwise, α_i has to backtrack (using the *backTrack*() procedure). Whenever α_i receives a *chk* message, it processes it by calling for procedure *processAssign*(*msg*). The *AgentView* of α_i is updated (*updateAgentView*) only if the received message contains an as-

signment that is more up-to-date than the one already stored for the sender, and all *nogoods* become non-compatible when the *AgentView* of α_i is removed.

Then, a consistent value for α_i is sought after the change in the *AgentView* (*checkAgentView*). When every value of α_i is forbidden by its *noGoodList*, procedure $backTrack()$ is called for. In procedure $backTrack()$, α_i resolves its no-goods, deriving a new no-good, *newNoGood*. If *newNoGood* is empty, the problem has no solution. α_i broadcasts the stp messages to all agents and terminates the execution. Otherwise, the new no-good is sent in an *ngd* message to the agent, say α_j, owning the variable appearing in its lrl. Then, the assignment of α_j is deleted from the *AgentView* ($updateAgentView$). Finally, a new consistent value is selected (*checkAgentView*).

```
procedure backTrack() do
    newNoGood ← solve(myNoGoodList);
    if !newNoGood then
        end ← true;
        sendMessage(stp, participantAgents);
    else
        sendMessage(ngd, newNoGood, s_j );
        updateAgentView(a_j ← null);
        checkAgentView();
    end if
end procedure
```

Whenever α_i receives an *ngd* message, procedure *resolve* is called for resolve the conflict. The no-good included in the *ngd* message is accepted only if its *hrl* is compatible with assignments on the *AgentView* of α_i. Next, the no-good is stored, acting as justification for removing the value on its lrl. A new consistent value for α_i is then sought (*checkAgentView*) if the current value was removed by the received no-good. If the no-good is not compatible with the *AgentView*, it is discarded because it is obsolete. However, if the value of a_i was correct in the received no-good, α_i resends its assignment to the no-good sender by way of a *chk* message.

Afterward, α_i sends its assignment through a *chk* message to the sender of the request if its value is different from the one included in the received message.

6.5 Adaptation from a Pervasive System Perspective

In previous sections, we provide a disCSP model to hardware service composition and a distributed service composition protocol suitable for pervasive hardware service composition. From a pervasive service perspective, hardware services should function in highly

```
procedure processAssign(msg) do
   updateAgentView(msg.Assignments);
   checkAgentView();
end procedure
procedure resolve(msg) do
   if conflict(lrl(msg.noGood), AgentView) then
      myNoGoodList.add (msg.noGood);
      checkAgentView();
   else
      if hrl(msg.noGood).value == a_i) then
         sendMessage(chk, myAssignment, msg.sender );
      end if
   end if
end procedure
```

dynamic environments occurring throughout a user's daily encounters. The assumption that a composed hardware service will remain static in nature once generated is false, as the services and resources available in the user's environment will be unpredictable. Normally, most of existing schemes for infrastructure-less hardware service composition and management do not consider the need for adaptation. Some others consider the adaptation of services based on a composition from scratch (i.e., it is every time a change occurs in the environment). We present a heuristic influenced by the service relevance level model for the adaptation, allowing the reuse of previous solutions, and through simulation we show its performance.

The dynamic adaptation of hardware services enables systems to expand their functionality by means leveraging resources in the user's vicinity; this serves to extend the availability of services in time and space. Most adaptation approaches based on ad hoc networks use service adaptation from scratch; that is, each time the environment changes, the system starts the composition process all over again, resulting in the depletion of system resources and network capacity. In real scenarios devices managed by agents can leave or enter, service availability in the ecosystem can change along with the environmental conditions.

Adaptation has been an important topic in diverse fields, for example: simulation [115], robotics [116] and graphics [117] [118]. In context of service-oriented systems [77] [119] [120] [121] [122]. Research in this field follows several trends. One way to characterize these trends is to organize them along several dimensions, where each dimension represents one or more facets of the service-adaptation problem. In the literature addressing this issue, there are several characterizations such as [123] [124] [125]. However, most of these characterizations are focused on SOA (Service-Oriented Architecture) and do not

address issues involving the system's environment and participants. Taking a perspective of pervasive service as the starting point, and considering the dimensions proposed by Cardellini in [125], we characterized the adaptation problem (figure 6.5 shows a taxonomy of adaptation from a pervasive service perspective) by answering the following questions:

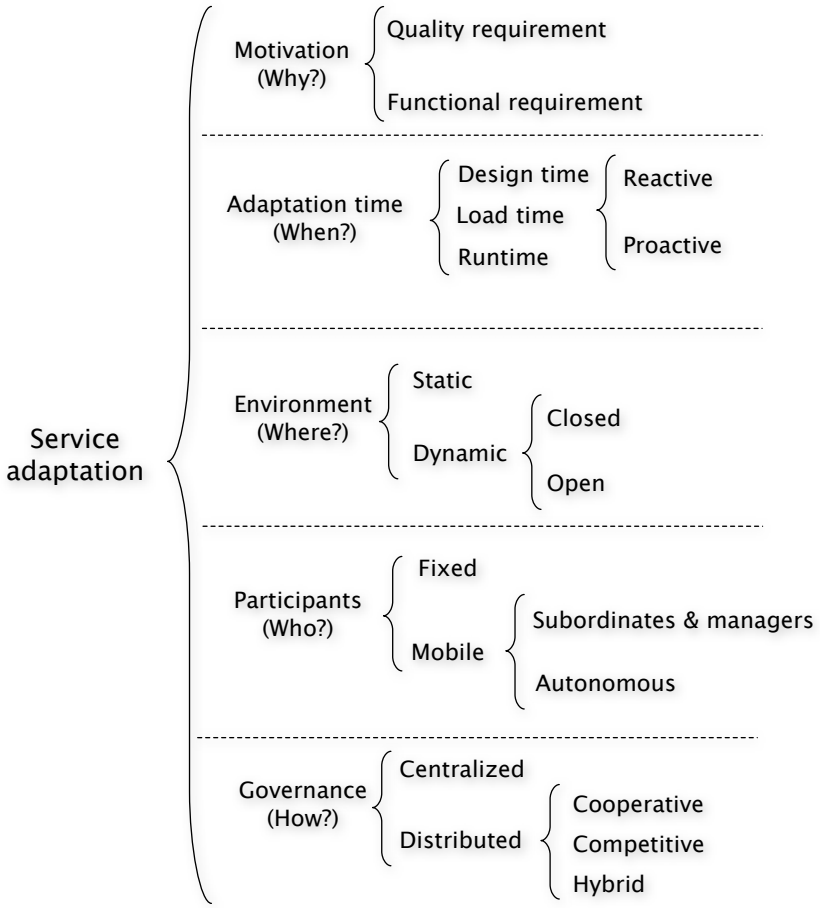

Figure 6.5: Service adaptation taxonomy

- **Why should a hardware service be adapted?** The primary objective of pervasive hardware service is to provide services that are required by the user, despite variations in its environment; this is achieved by means of adaptation. Adaptation aims to make the pervasive services able to fulfill functional and/or QoS requirements in spite of changes in their environment [126] [127]. Our focus is on functional requirements concerning the availability of services to fulfill user needs.

- **When must adaptation actions be executed?** Adaptation must be carried out at

different stages in a hardware service's life cycle. Existing approaches can be placed between design time and run-time stages [128] [129]. In the pervasive service domain, there is a special interest in performing the service adaptation during run-time [130] [131] [132] [133]. Within this stage, we can distinguish between reactive and proactive approaches. In the proactive approach, systems predict possible future changes in order to perform the adaptation. In the reactive approach, systems are adapted after the detection of changes. Currently our approach adopts a reactive mode placed in run-time.

- **Where does the adaptation occur?** Broadly speaking, pervasive services may be deployed in several kinds of environments: static or dynamic. In the pervasive service domain, there is a growing emphasis on building services tolerant of unexpected changes in their environment [134] [135] [136]. Within this type of dynamic environments, we may further distinguish between closed and open environments. In closed environments, all participant systems are known at the stage of system design, while in open environments, the participants may move from one environment to another. The majority of adaptation approaches are designed for closed environments, using servers for service registration, discovery, composition and adaptation, and assuming reliable participanting systems, for example [125] [137] [138]. These approaches often involve preconfigured adaptation managers. They do not cater to highly pervasive, mobile and ad hoc environments, leaving themselves brittle to issues such as central point of failure, mobility, and fault management. This paper adopts an adaptation approach for dynamic environments.

- **Who participates in the adaptation process?** This question has to do with the kind of systems that participate during the adaptation process (for example; autonomous participants (agents), or subordinate and manager systems), that is, the regime that manages the service-adaptation process. In the case of multiple systems, their adaptation can be under the control of a single authority or under the control of multiple authorities (they can be cooperative or/and competitive). We are interested in cooperative autonomous participants, that is, devices managed by agents that need to cooperate to achieve their objectives, while preserving their autonomy (i.e., each agent is the only one that can decide whether to cooperate or not). In our approach, each agent is motivated to cooperate with others due to its need for other agents to cooperate with it.

Diverse approaches have been proposed for service adaptation, for example dynamic service selection, dynamic coordination pattern selection, etc. However, most authors assume that resources are always available and they only consider the dynamic degradation of service quality. In this respect, this paper aims to achieve adaptation of hardware services based on a mutual cooperation approach for a distributed selection of participant agents and their services while preserving the principle of autonomy (i.e. any agent in the user environment can be forced to be part of a hardware service composition and only each participating system may decide whether or not to provide a service).

6.6 *Dynamic-disCSP* Framework

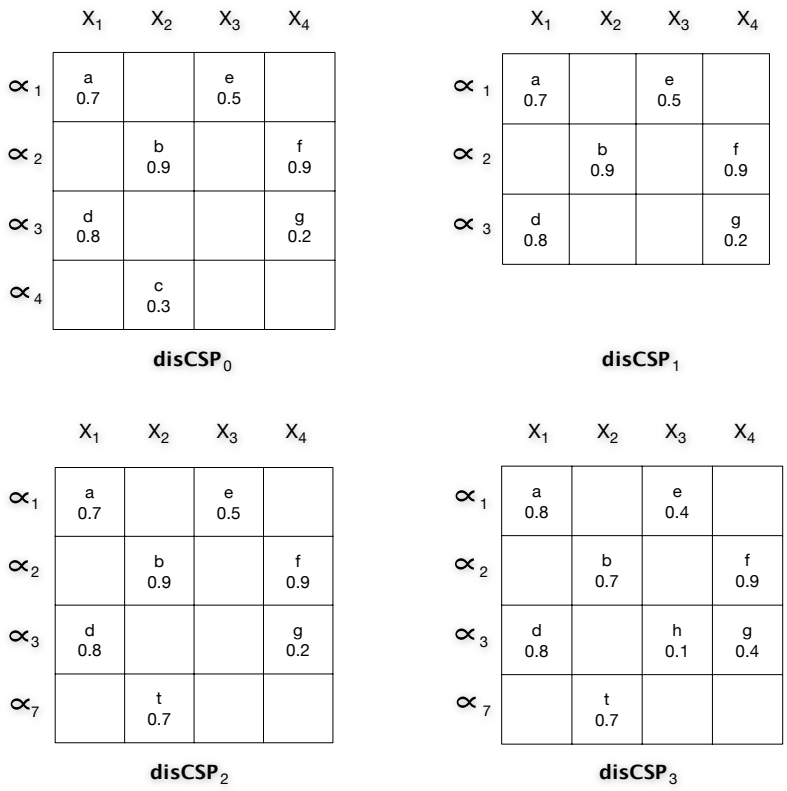

Figure 6.6: Matrix representation of a Dynamic disCSP

In order to meet the hardware service adaptation outlined in the previous section, we model the service adaptation problem as a *dynamic-disCSP*. A *dynamic-disCSP* is a sequence of *disCSPs* $\langle disCSP_0, disCSP_1, ..., disCSP_n \rangle$. A *disCSP* is represented as a 4-tuple

$\langle X, D, C, SYS \rangle$ where $X = \{x_i | i = 1, ..., n\}$ is a finite set of variables; x_i is a variable corresponding to a service needed by an ecosystem member (i.e., an agent α) to satisfy a particular user requirement; $D = \{D_i | i = 1, ..., n\}$, such that for each x_i there is a service option domain D_i; C is a finite set of constraints $\{C_1, C_2, ..., C_m\}$ of the user requirement; and A is a set of autonomous agents over which the variables and constraints are distributed. At any system, the set of possible values for variable x_i is its current domain D_i. Each $distCSP_i$ results from a change in the previous one, $disCSP_{i-1}$, and represents new situations in the dynamic environment. Figure 6.6 shows a visual abstraction (each matrix is constituted by participant agents α_i and required services x_j; each cell represents a service that each participant can provide and its relevance) of a sequence of four disCSPs $\langle disCSP_0, disCSP_1, disCSP_2, disCSP_3 \rangle$. In the sequence various changes can be observed, such as:

- In $disCSP_1$, α_4 has left the group of participating agents (AG) with regard to $disCSP_0$).
- In $disCSP_2$, sys_7 has joined the group of participating agents (AG) with regard to $disCSP_1$).
- In $disCSP_3$, the relevance of the services provided by α_1, α_2 and α_3 has changed with regard to $disCSP_2$.

These kinds of changes may affect the components in the problem definition: variables (additions or removals), domains (changes in the intentional definition, value additions or removals in the case of extensional definition), constraints (additions or removals), constraint scopes (variable additions or removals), or constraint definitions (changes in the intentional definition, tuple additions or removals in the case of extensional definition). As a result of such changes, the set of solutions of each $disCSP_i$ can potentially decrease (a restriction of the $disCSP_i$) or increase (a relaxation of the $disCSP_i$).

6.6.1 A Service Adaptation Heuristic

Starting from the fact that the environment has changed (for example, a service has become unavailable or a participant α has left the Agent Group (AG)), the AG may need to adapt its topology and functionality to meet user needs. In order for the AG to be able to withstand service adaptation, we propose a heuristic named Dynamic Partial Solution (DPS) that is based on the dynamic addition of new participating agents α to the AG.

For this, an identification phase begins to detect potential participant agents α for the AG in the user's vicinity. When an adaptation requirement becomes activated in the AG, an initiator agent sends a task request (TaskRequest) message, containing the STask description with the latest partial solution, to agents in the user's environment that are not participating in the current solution. The STask is used by receiving agents to decide whether they can contribute at least one service to fulfill the STask. Agents that may contribute send a TaskReply message to the system initiator. With the TaskReply messages the system initiator updates the candidate system list, where each new item on the list is assigned the address and most important factor in relation to candidates that contributed to the previous solution. Once the list is made, the initiator agent sends it to all participating agents. Figure 6.7 shows an example where an adaptation requirement becomes activated at agent α_1.

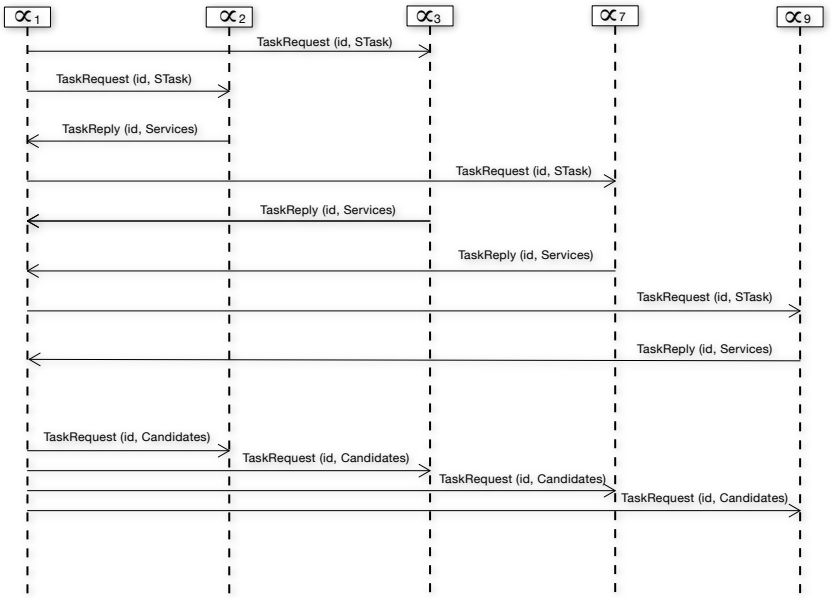

Figure 6.7: New candidate choice for the service adaptation

This heuristic allows AG to adapt services without starting from scratch. That is, with the heuristic DPS, the algorithm PHSC can reuse the latest service composition knowledge. Figure 6.8 shows an abstraction of an initial service composition ($disCSP_i$) and the service composition state after changes in the environment ($disCSP_i'$) that can be used in the service adaptation process. In this example α_4 left the AG.

In order to achieve the adaptation, we start from the $disCSP_0$ that models the initial state of the AG. The moment a service fault is detected, the initiator agent starts the adaptation

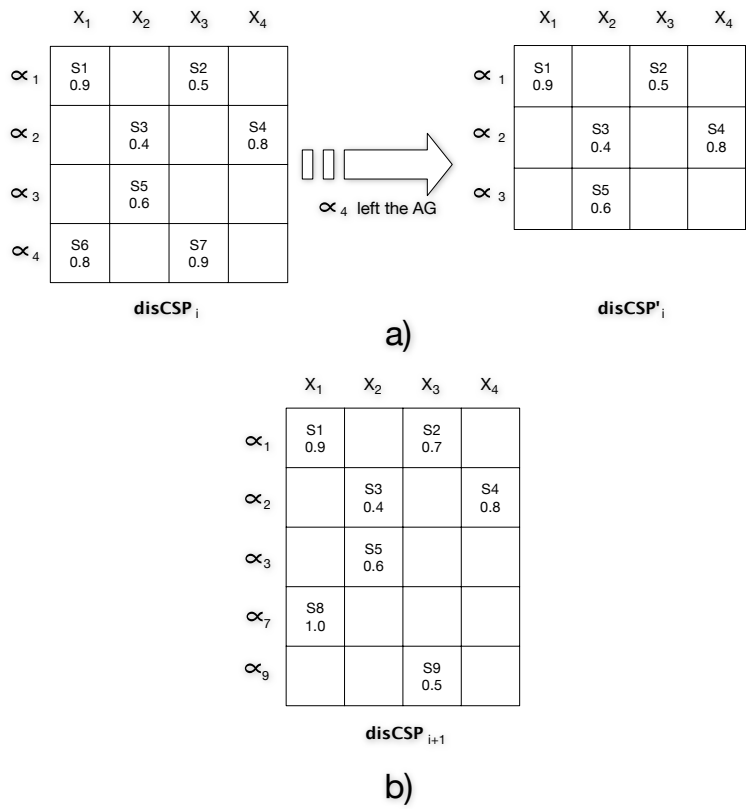

Figure 6.8: Example of a possible service adaptation

process. First, the AG attempts to reallocate the required services for the available participating agent α. If it is not possible to find an assignment without violating constraints, then the initiator agent sends a message *TaskRequest* message to all agents found in its neighborhood in order to find new participating agent capable of contributing to the service adaptation. Once the initiator agent α has located the potential agents δ, the DPS heuristic is applied. This heuristic is based on the idea that current members of the AG are unable to satisfy user requirements (i.e., it is not possible to find an assignment without violation of constraints); therefore, it is necessary to give higher relevancy to the new participating agents. In practical terms, this idea implies that the initial partial solution (partial composition of a service) should be adjusted according to the contributions of new participating agents.

6.7 Conclusion

In this chapter we presented a dynamic *DisCSP* model for pervasive service composition in dynamic environments, and a heuristic for adapting partial solutions using an asynchronous backtracking algorithm for solving the dynamic *DisCSP*. The model provides a technique for adapting services without restarting the service composition process each time that a participating agent is unavailable or leaves the *AG*.

Most solutions designed for the composition of adaptive pervasive services are based on dedicated infrastructure. These solutions use notions such as central servers, stable nodes and reliable communications channels. These include proposals such as [139] [140] [141]. Most of these approaches involve preconfigured composition mechanisms residing on dedicated machines with high resources. Some authors such as [136], [142] only consider the initial composition of pervasive services. Other authors have proposed protocols and frameworks for the adaptation of pervasive services in slightly dynamic environments; for example Karmouch and Nayak [143] proposed a *DisCSP* model for service composition. They used a *QoS*-based approach to adapt the service composition (to determine the quality level, bandwidth, delay, loss and jitter were used in the network). However, the framework proposed by Karmouch assumes that all the components of the instance under consideration, such as variables, domains and constraints, are completely known before modeling and solving it, and do not change either during or after modeling and solving. However, it has been observed for a long time that such assumptions do not hold true in many situations.

Multihop composition and adaptation was not considered in this research because one hop is enough to discover autonomous devices in the user's environment. However, even though the resources needed to provide a service may not be in the user's vicinity, the proposal could use them through a directed multihop broadcast.

Chapter 7

Implementation

A solution on the Multiagent Software Hardware Simulator (MASH) platform was designed so that service ecosystems can be built based on an agent federation. The solution allows us to build ecosystems of pervasive hybrid services. The overall solution consists broadly of four main elements: an agent group formation protocol, a candidate formation protocol, a service composition protocol and a service adaptation protocol. For this we have implemented the obligation-based ACL (described in chapter 5) to support agent interaction based on obligations and a package to model problems of constraint satisfaction. This chapter introduces the MASH platform and provides an overview of our implemented solution.

7.1 MASH: A Tool to Tune Design and Deployment

There are several tools for simulations such as ns3[144], tinyOs[145] and J-Sim [146]. However, most of these tools were designed for network simulations and do not provide support for implementing multiagent systems (at least in a simple and transparent manner). MASH, on the other hand, is a tool that allows developers to simulate and execute embedded multiagent systems including real-world software/hardware agents [147]. This section provides the key features of this tool, and includes a short explanation of the MASH architecture and the basic aspects of initial setup, through scenario creation and agent setup.

7.1.1 Overview of the MASH Architecture

MASH enables developers to simulate several agent nodes interacting with an environmental component through an abstraction layer. Figure 7.1 shows the basic architecture of MASH.

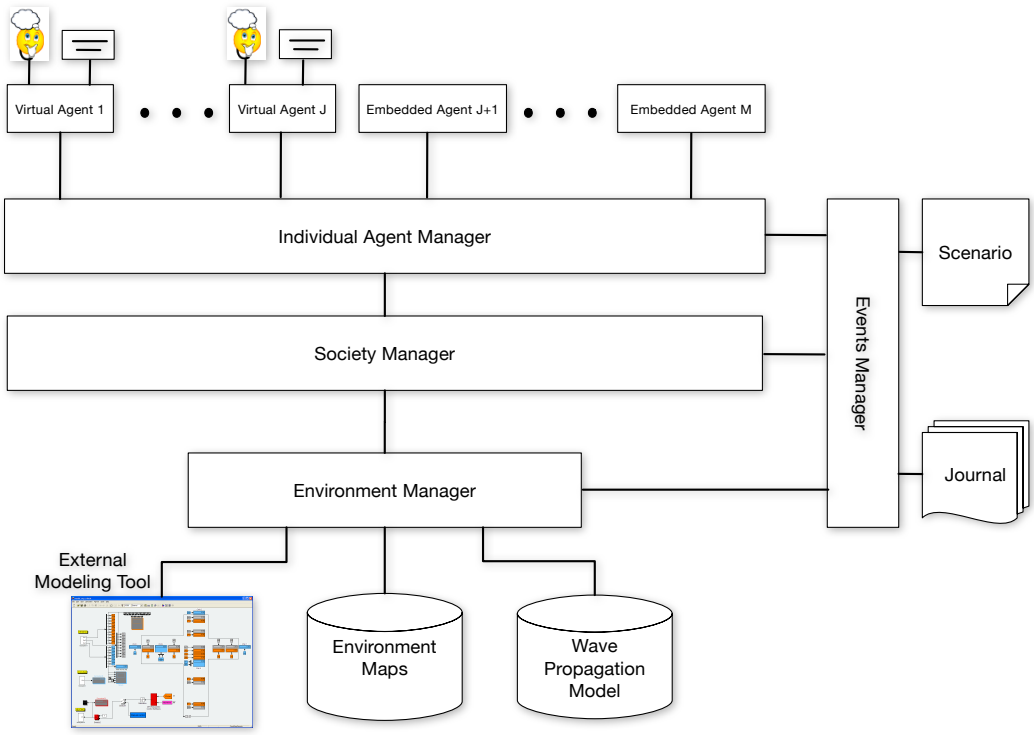

Figure 7.1: Simplified architecture of MASH

MASH supports the following four types of agents:

1. Software agents are traditional agents that can perceive and act on virtual environments.

2. Virtual agents are a type of software agent used to simulate the behavior of real-world embedded agents.

3. Embedded agents are agents embedded in the real world that can perceive and act on the physical environment. They are often constituted by a software part and a hardware part. Soccer robots, autonomous vehicles and intelligent sensors are embedded agents.

4. Avatar agents represent embedded agents in virtual societies. They enable embedded agents to interact with software agents. They link the simulated MAS with behaviors computed on physical devices.

Virtual agents and embedded agents are abstracted by an Individual Agent Manager (IAM). The IAM enables the integration of virtual and real world agents in the simulation. An agent can be implemented by a software agent (such as a java class) or its behavior can be computed in a real-world embedded agent. In this case, an avatar translates the logical call of methods and exchanged messages to its wrapped embedded agent. The avatar make it possible to give a graphic representation of the real-world embedded agent in MAS representation.

The *Behavior* component simulates the execution of software on a single device. It processes messages received from the other agents. The Environment Manager and the Society Manager enable agents to interact together and with their environment.

7.1.2 A Toy Problem

In this section we will to use a simple toy problem to describe the use of MASH. The example is one extension of the predator-prey pursuit problem. In our case, predators are replaced by police and the prey by a thief. The police must search for the thief and the thief must run away from the police. (Figure 7.2).

7.1.3 Building a Solution

In MASH a solution is the abstraction of the project that is being simulated. It specifies the agents that our project will manage, in order to ensure the cleanliness of the solution, and is should follow a specific order. This section describes how to set up a solution correctly.

The first step in implementing a solution is to create the solution item. This item must to be placed in *simulation.solutions.custom*, extend from the *SolutionItem* class and invoke the method *setSolution* so that it can be displayed. In our Toy Problem it should be:

```
public class PoliceThiefCitizenSolution extends SolutionItem{
  public PoliceThiefCitizenSolution() {
    super();
    try{
    Vector<Class> agents = new Vector<Class>();
```

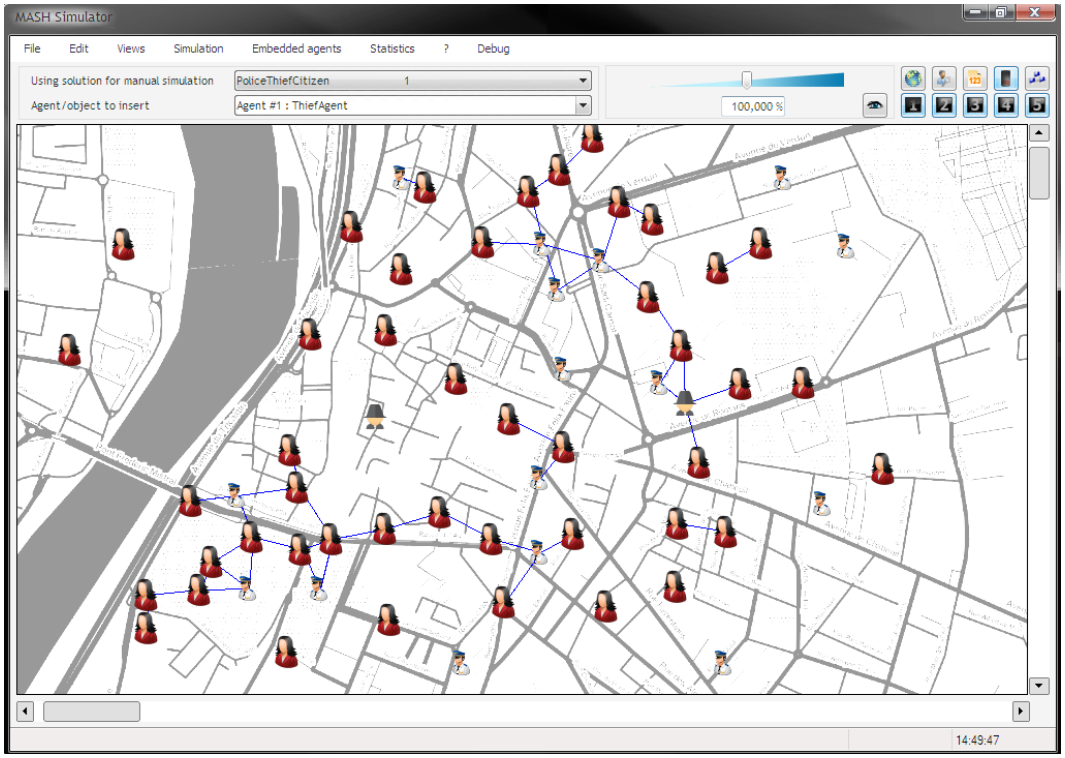

Figure 7.2: Toy problem example

```
agents.add(Class.forName(
    "simulation.solutions.custom.PoliceThiefCitizen.ThiefAgent"));
agents.add(Class.forName(
    "simulation.solutions.custom.PoliceThiefCitizen.PoliceAgent"));
agents.add(Class.forName(
    "simulation.solutions.custom.PoliceThiefCitizen.CitizenAgent"));
super.setSolution(
    new Solution(
        new SolutionDescriber(
            "Francisco Cervantes",
            "francisco.cervantes@lcis.grenoble-inp.fr",
            "University of Grenoble / CINVESTAV",
            "Valence",
            "France",
            "PoliceThiefCitizen",
            "1",
            "Implementation of Police-Thief-Citizen model"),
        agents
```

```
        )
    );
    }catch(SolutionException e){
      e.printStackTrace();
    }catch(SolutionDescriberException e){
      e.printStackTrace();
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
}
```

The next step is to configure *MASH* to recognize the new solution. Select *Start MASH Edit -> Preferences -> Existing Solutions* and write the name of the solution item, in our case *PolizeThieveCitizenSolution* (Figure 7.3).

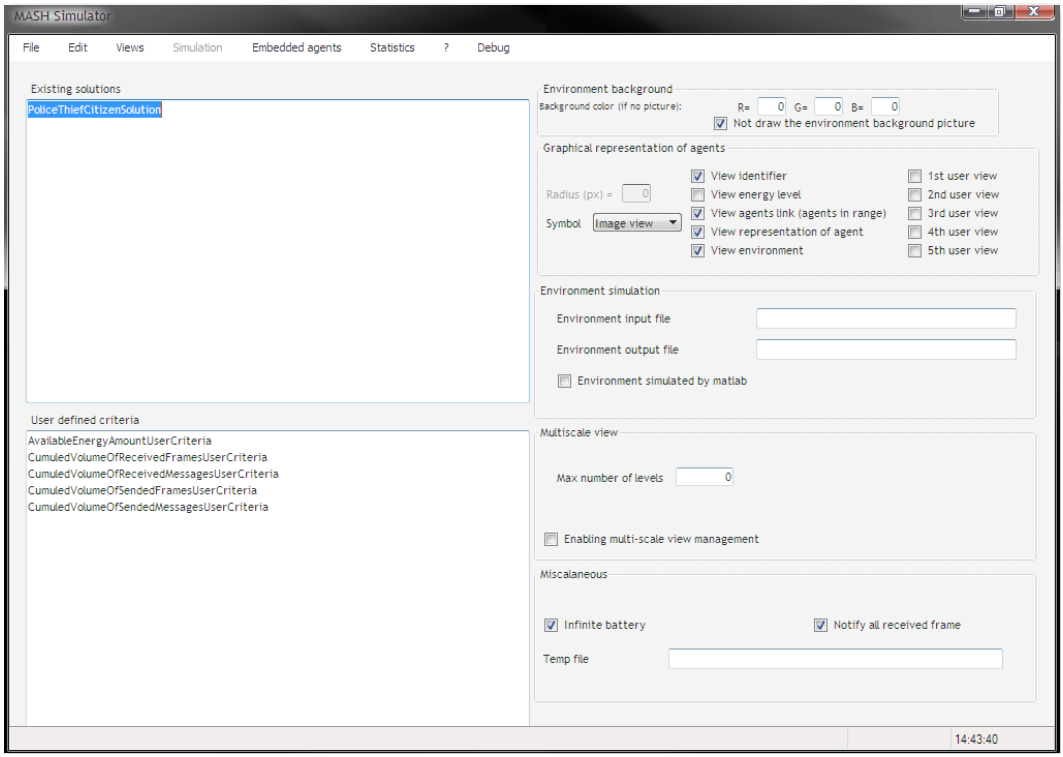

Figure 7.3: Setting MASH Preferences

7.1.4 Creating an Agent

In accordance with the MASH package organization we must place the agents in a new package *simulation.solutions.custom.NEWPACKAGE*. In the case of our toy problem it must be *simulation.solutions.custom.PoliceThiefCitizen*.

To create an agent, MASH provides an *Agent* class that our objects must extend and creates a constructor (*MAS*, *Integer id*, *Float energy*, *Integer range*). Each *Agent* will be run as a new thread, so we must override the *run*() method. In order to give a little time lapse until the construction of all the agents, it is recommended that our first line in the run method be a sleep. Here is an example of the *PoliceAgent* structure:

```
public class PoliceAgent extends Agent implements
    ObjectAbleToSendMessageInterface{

  public PoliceAgent(MAS mas, int id, Integer range) {
    this(mas, id, (float)1,range);
  }

  public void run() {
    try{Thread.sleep(SLEEP_TIME_SLOT);}catch(Exception e){}
        while(!isKilling() && !isStopping())
    {
      while(((isSuspending()) && (!isKilling() && !isStopping()))){
      ...
      }
    }
}
```

It is important to remember that each *Agent* class (in our example, PoliceAgent, ThiefAgent and CitizenAgent) must be included in the *Solution Item* implementation. At this moment our agents are ready to be added on the scenario (see Figure 7.4); however, they are not able to interact with other agents yet, which brings us to the next section.

7.1.5 Agent Interaction in MASH

The *MASH* defines an interaction framework based on *Messages* and *Frames*. They represent the third and second layer of the *OSI* model. Frames will contain messages; therefore

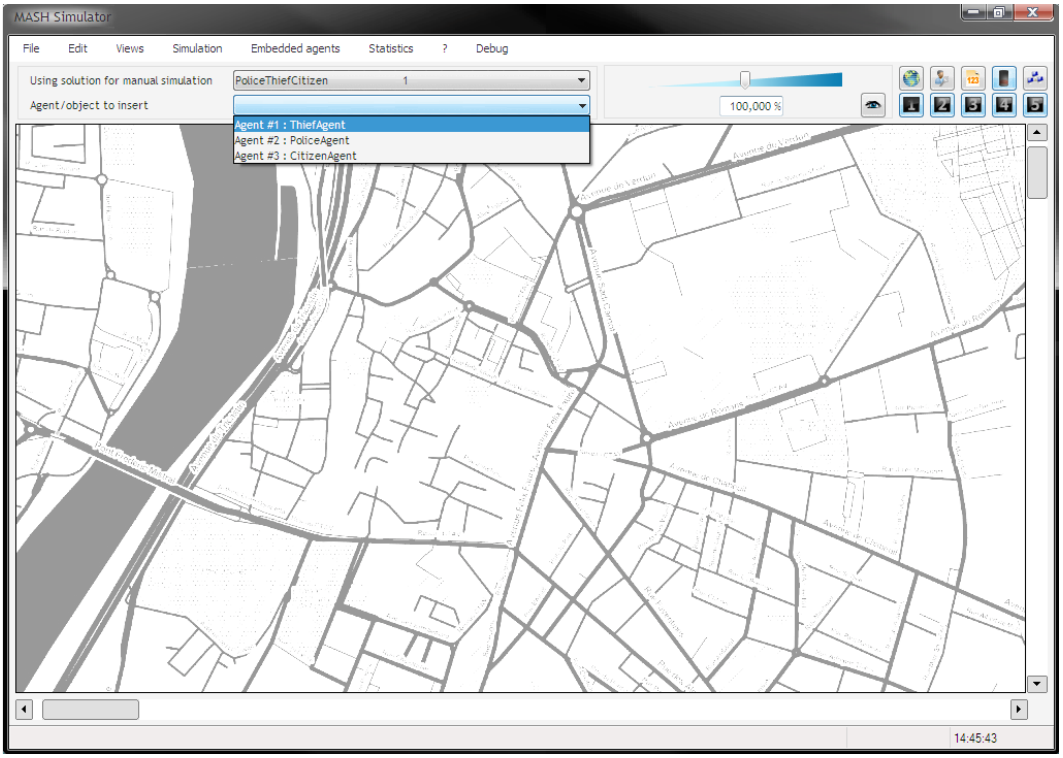

Figure 7.4: MASH Solution

the *Message* class provides a method to be encapsulated into a byte sequence, and the *Frame* class provides a method to unencapsulate messages so that they can be used in a hybrid simulation. Here a correct implementation of *Messages* and *Frames* in *MASH* is shown. According to the *MASH* package organization we must place our interaction classes in *simulation.solutions.custom.PoliceThiefCitizen.Messages*

The interaction between agents in our toy problem is simple. The police look for a thief sending a *HOWL* message every time he moves (this message contains his position). If a thief hears the *HOWL*, he *TURNS* and runs in the opposite direction. The police hears the *TURN* with the thief's position and runs in that direction. When the police are very close to the thief, they send a *CATCH* message and finish their execution. The thief receives this message and finishes his execution too.

Here is the code of the *PoliceThiefCitizen Message* and *Frame* classes:

```
public class PoliceThiefCitizenMessage extends Message{

  private static final long serialVersionUID = 1L;
  private int senderID;
  private int receiverID;
```

```
  private IntegerPosition pos;

  public byte type;

  public static final byte HOWl=0;
  public static final byte TURN=1;
  public static final byte CATCH=2;

  public PoliceThiefCitizenMessage(int sender, int receiver, byte
      type,IntegerPosition pos){
    senderID=sender;
    this.receiverID=receiver;
    this.type=type;
    this.pos = pos;
  }

  public int getReceiver() {return receiverID;}
  public int getSender() {return senderID;}

  public IntegerPosition getPosition(){ return pos;}

  public byte[] toByteSequence(){
    if(type==CATH)
      return
          ByteBuffer.allocate(19).put(type).putInt(this.senderID).putInt(receiverID).putInt(0
    else
      return
          ByteBuffer.allocate(19).put(type).putInt(this.senderID).putInt(receiverID).putInt(p
  }
}
```

```
public class PoliceThiefCitizenFrame extends Frame {
  private static final long serialVersionUID = 1L;
  public PoliceThiefCitizenFrame(int sender, int receiver,byte[] data){
    super(sender,receiver,data);
  }
```

```
  public PoliceThiefCitizenFrame(int sender, int receiver,Message msg){
    super(sender,receiver,msg);
  }

  public Message getMessage(){
    ByteBuffer buffer = ByteBuffer.wrap(this.getData());
    byte type= buffer.get();
    if(type == PoliceThiefCitizenMessage.CATCH)
      return new PoliceThiefCitizenMessage(buffer.getInt(), buffer.getInt(),
          type, null);
    else if(type == PoliceThiefCitizenMessage.TURN)
      return new PoliceThiefCitizenMessage(buffer.getInt(), buffer.getInt(),
          type, new IntegerPosition(buffer.getInt(), buffer.getInt()));
    else if(type == PoliceThiefCitizenMessage.HOWl)
      return new PoliceThiefCitizenMessage(buffer.getInt(), buffer.getInt(),
          type, new IntegerPosition(buffer.getInt(), buffer.getInt()));
    System.out.println("Error in the encapsulation of the frame data");
    return null;
  }
}
```

7.2 Implementation of the Solution

7.2.1 Overview of the Solution: Service Ecosystem based on a Federation of Agents

Considering the issues raised for the deployment of pervasive services, we have developed a solution for creating pervasive ecosystem services. Our proposal is based on a social ecosystem metaphor, the interaction between agents is based on obligations, and service composition is seen as a constraint satisfaction problem (distributed and dynamic). Figure 7.5 shows the expected behavior of the ecosystem through the major stages involved in the deployment of pervasive services.

According to the ecosystem description provided in 2, the main elements of an ecosys-

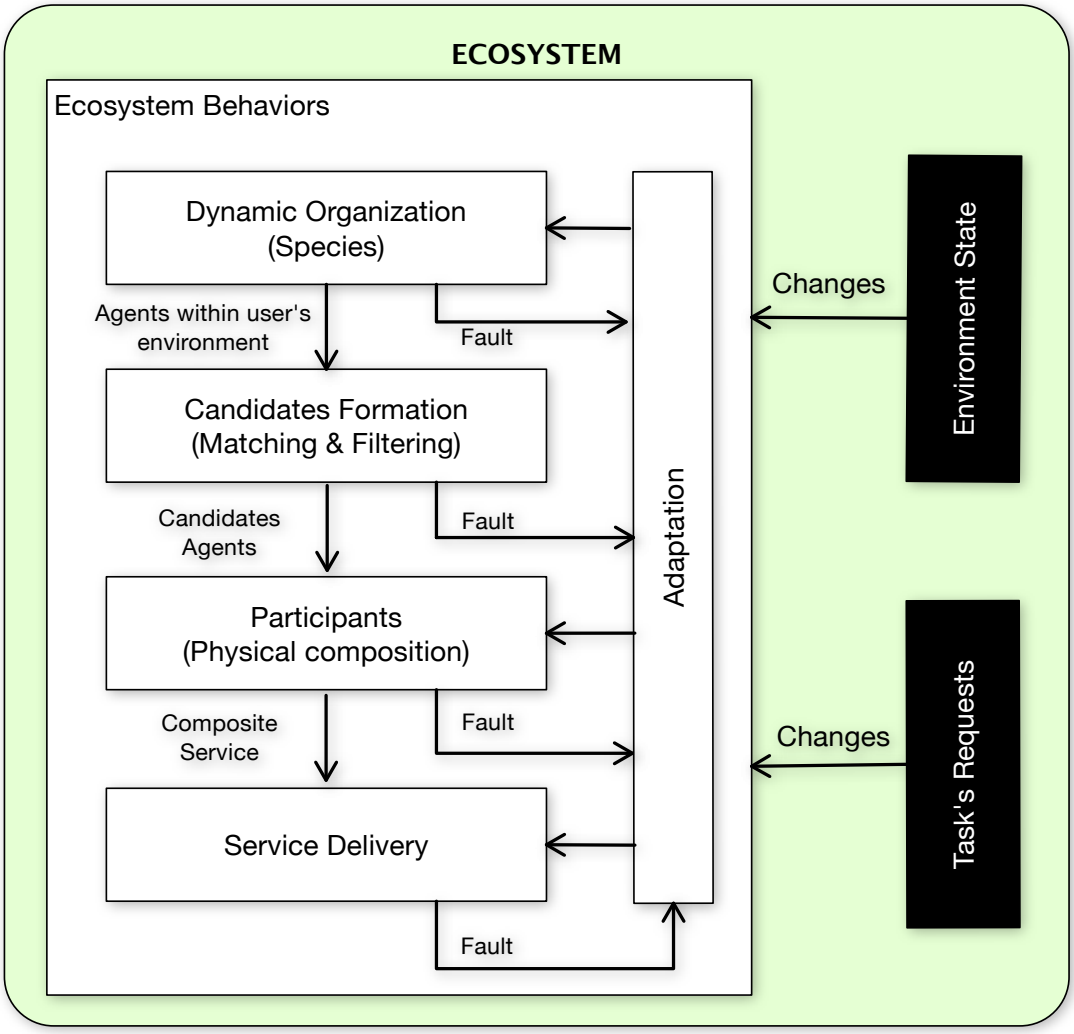

Figure 7.5: Main behaviors of our service ecosystem

tem are the environment, the members (in our case modeled by agents) and the laws (i.e. interaction mechanisms). It bears mentioning that these behaviors must emerge from the local behavior of ecosystem member agents and the way interact with each other. Therefore the behaviors observed in figure 7.5 – dynamic organization, candidates formation, participants election, service delivery, and adaptation – are all implemented in each member agent of the ecosystem as distributed and asynchronous mechanisms. In the rest of this chapter, the main elements of our implementation are described.

The input into the system is a task requested by the user through a member of the ecosystem and the output is the service (atomic or composed) provided by the ecosystem members in order to fulfill the user request. As was described before, members of the ecosystem are agents. Figure 7.6 shows a class view of the agent implemented in our

solution.

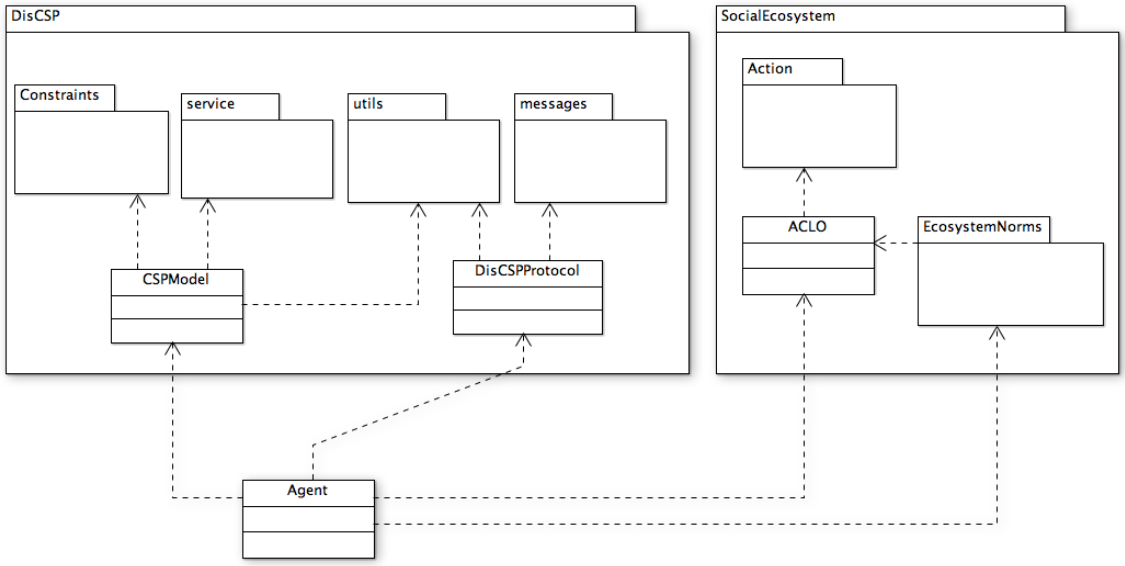

Figure 7.6: Class view of the agent implementation

As shown in Figure 7.6, agent definition uses classes such as CSPModel, DisCSPProtocol, ACLO and EcosystemNorms in order to support agent interaction based on social norms, to represent constraint satisfaction problems, and to provide the composition / adaptation services through protocols that search for solutions to problems of satisfaction restrictions (ABT-based algorithm). In the MASH context (Figure 7.7) our solution is stored in the subpackage "solution.customer".

7.2.2 Formation of Candidate Agents

The formation of candidate agents is driven by messages and oriented to task requests. Each agent, when receiving a TaskRequest message, must answer the following question: Could I make a contribution for the requested task?. The following code is a part of the implementation for the formation of agent candidates.

```
if(taskRequests.size()>0){
  if(isCandidate(taskRequests.get(0))==true){
    agentTask=taskRequests.get(0);
    taskCandidates=new TaskCandidates(1,
        taskRequests.get(0).getAgentOwnerId());
    sendMessage(taskRequests.get(0).getAgentOwnerId(),"TaskReply");
```

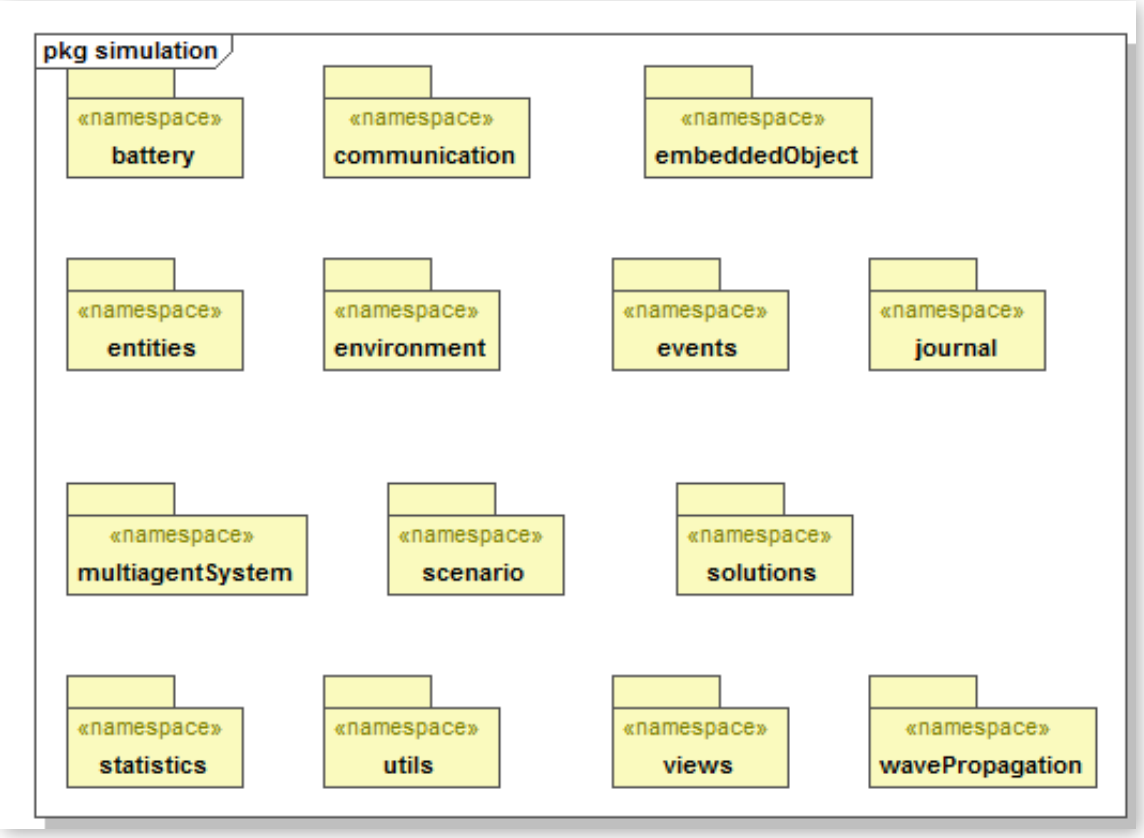

Figure 7.7: Package view of the MASH Context

```
  }
  taskRequests.remove(0);
}

public boolean isCandidate(AgentTask taskRequest){
  boolean candidate=false;
  for(Service s: agentServices){
    for(Service r: taskRequest.getRequirements()){
      if(s.getName().equals(r.getName())==true)
        return true;
    }
  }
  return candidate;
}
```

```
if(searchingCandidates==true){
  searchingCandidates=false;
  for (TaskReply tReply: taskReplies){
    taskCandidates.addAgentCandidate(new AgentCandidate(tReply.getAgentId(),
        tReply.getAgentRelevance()));
  }
  for(int index=0; index<taskCandidates.getSize();index++){
    System.out.println("Agent " +
        taskCandidates.getAgentCandidates().get(index).getAgentId());
  }
  readySolution=false;
  timer.start();
  timer.setON();
  for (TaskReply tReply: taskReplies){
    sendMessage(tReply.getAgentId(), "ACT");
  }
  taskReplies.clear();
}
```

7.2.3 Composition and Adaptation of Services

Composition and adaptation algorithms are driven by messages. After processing the acts at the level of social obligations, the following partial code processes each command of the protocols for composition and adaptation.

```
if(msg.type==ABTMessage.TaskRequest){
  this.notifyEvent(new ABTMessageEvent(this.getSystemId(),"TaskRequest"));
  int senderID=frame.getSender();
  ByteBuffer buffer = ByteBuffer.wrap(frame.getData());
  byte type= buffer.get();
  int senderId=buffer.getInt();
  int receiverId=buffer.getInt();;
  byte[] b=new byte[buffer.remaining()];
  buffer.get(b);
  try {
    if (agentTask==null){
```

```
      agentTask = (AgentTask)Serializer.deserialize(b);
      taskRequests.add(agentTask);
    }
  } catch (ClassNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  }

}

if(msg.type==ABTMessage.TaskReply){
  this.notifyEvent(new ABTMessageEvent(this.getSystemId(),"TaskReply"));
  int senderID=frame.getSender();
  ByteBuffer buffer = ByteBuffer.wrap(frame.getData());
  byte type= buffer.get();
  int senderId=buffer.getInt();
  int receiverId=buffer.getInt();
  double relevance=buffer.getDouble();
  taskReplies.add(new TaskReply(senderId, relevance));
}

if(msg.type==ABTMessage.ACT){
  this.notifyEvent(new ABTMessageEvent(this.getSystemId(),"ACT"));
  int senderID=frame.getSender();
  ByteBuffer buffer = ByteBuffer.wrap(frame.getData());
  byte type= buffer.get();
  int senderId=buffer.getInt();
  int receiverId=buffer.getInt();;
  byte[] b=new byte[buffer.remaining()];
  buffer.get(b);
  try {
    taskCandidates = (TaskCandidates)Serializer.deserialize(b);
  } catch (ClassNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
```

```
    e.printStackTrace();
  }
  initialContribution();
}

if(msg.type==ABTMessage.CHK){
  this.notifyEvent(new ABTMessageEvent(this.getSystemId(),"CHK"));
  int senderID=frame.getSender();
  ByteBuffer buffer = ByteBuffer.wrap(frame.getData());
  byte type= buffer.get();
  int senderId=buffer.getInt();
  int receiverId=buffer.getInt();;
  byte[] b=new byte[buffer.remaining()];
  buffer.get(b);
  try {
    newAssignments = (AgentAssignments)Serializer.deserialize(b);
    processAssignments(newAssignments);
  } catch (ClassNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  }
}

if(msg.type==ABTMessage.NGD){
  this.notifyEvent(new ABTMessageEvent(this.getSystemId(),"NGD"));
  int senderID=frame.getSender();
  ByteBuffer buffer = ByteBuffer.wrap(frame.getData());
  byte type= buffer.get();
  int senderId=buffer.getInt();
  int receiverId=buffer.getInt();;
  byte[] b=new byte[buffer.remaining()];
  buffer.get(b);
  try {
    noGoods.add((NoGood)Serializer.deserialize(b));
  } catch (ClassNotFoundException e) {
```

```
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  }
}

if(msg.type==ABTMessage.NOSOLUTION){
  this.notifyEvent(new ABTMessageEvent(this.getSystemId(),"NOSOLUTION"));
  if(agentTask.getAgentOwnerId()==getUserId()){
    readySolution=false;
  }
  else
    clearTaskMemory();
  }
  if(msg.type==ABTMessage.SOLUTION){
    this.notifyEvent(new ABTMessageEvent(this.getSystemId(),"SOLUTION"));
    if(agentTask.getAgentOwnerId()==getUserId()){
      this.notifyEvent(new ABTSolutionEvent(this.getSystemId(),"ABTSolution
          (" + System.currentTimeMillis() + ")"));
      readySolution=true;
      timer.setOFF();
      for(AgentCandidate candidate: taskCandidates.getAgentCandidates()){
        sendMessage(candidate.getAgentId(), "STP");
      }
    }
  }
  if(msg.type==ABTMessage.STP){
    clearTaskMemory();
    this.notifyEvent(new ABTMessageEvent(this.getSystemId(),"STP"));
  }
}
```

7.2.4 Conclusion

In this chapter, we present the MASH platform that allows us to simulate and evaluate our proposal. Additionally we provide an overview of our implemented solution for deploying

ecosystems of pervasive services. Our proposal describes the expected behaviors of the ecosystem that must emerge from agent's local behaviors and the way they interact with each other. Through these behaviors, we cover the four stages usually considered by other authors for service composition. Furthermore, we provide support for changes in the environmental state and task requests, i.e., we have considered that the real world is dynamic and services cannot remain static after their composition. In addition, the proposed solution also contemplates that there may be failures during the execution of each behavior. The next chapter presents the evaluation of our proposal and the results obtained.

Chapter 8

Evaluation

This chapter shows the performance of our Pervasive Hardware Service Composition (PHSC) protocol when system scale, mobility, service density, and composition order are varied. Our experimental scenario consists of agent-managed mobile devices that are able to provide one or more services. These devices are connected to each other using an ad hoc network. We considered compositions of several orders (in terms of number of the services needed in order to fulfill the task). We compare our results with those of Karmouch and Nayak [143]. The aim of the comparison is to identify the improvement of the performance in message utilization and composition time using our PHSC protocol. At the same time we want to determine the effects of varying service density, composition order and scale on the same metrics.

8.1 Pervasive Hybrid Service Composition

8.1.1 Scenario Description

We built a pervasive hybrid service based on an ad hoc network and implemented both PHSC and Virtual Device Constraint Satisfaction Protocol (VDCSP) protocols using MASH. Simulations were carried out over a set of agents in a previously delimited area, following a random-way-point mobility model [148]. All broadcasts had a bounded hop count of one for the PHSC and VDCSP protocols. The simulation was done for service composition orders of three, five, seven and nine, and service densities from 10 to 100 percent (i.e., percentage of nodes that have one service required to fulfill a requested task). For each simulation, we identified the amount of time and number of messages consumed to achieve

the composition (values were the average of 100 simulations). In order to focus on the performance of the protocols, we utilized predetermined similarity values (for comparing services with each others) and random relevance levels for the services provided by each agent. At this moment, we consider the changes to network conditions as noise in the measurement of the performance of the protocols. Therefore, we do not consider the effect of network conditions on the performance of the two protocols. However, it is possible to consider the network conditions as factors to determine the relevance level of services.

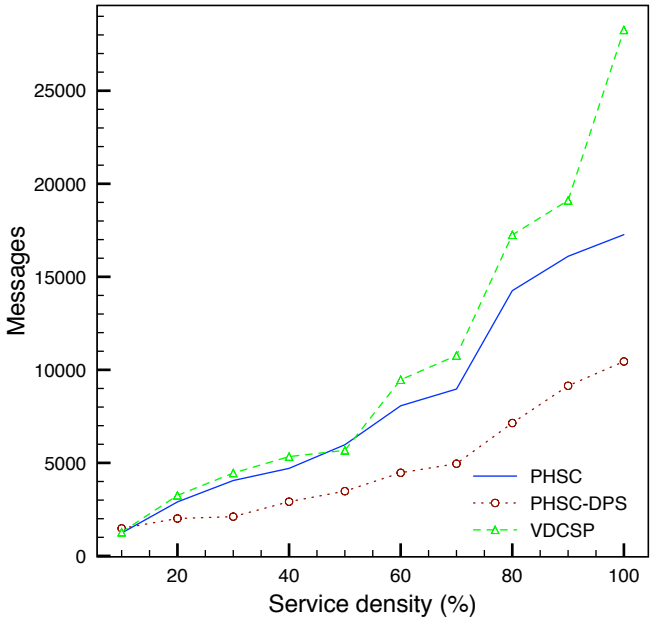

Figure 8.1: Number of messages with respect to service density

8.1.2 Effects of Service Density

We used the concept of service density as the percentage of agents that have least one of the required services to fulfill a requested task. In order to analyze the effects of service density on the performance of each protocol, we used a scenario with 50 static devices (this is because the mobility of the device may introduce noise into the effect of the service density), and a composition length of five services. The simulation was carried out using service densities of 10 - 100 percent in a spatial area of 50 m^2. For each simulation, we identified the number of messages consumed and the amount of time used to achieve the service composition. Results indicated that per composition, the number of messages consumed by PHSC (Figure 8.1) was an average of 2,132 messages (20.33 percent) lower

than VDCSP, and the time used for composition decreased by 0.92 percent compared to VDCSP (Figure 8.2).

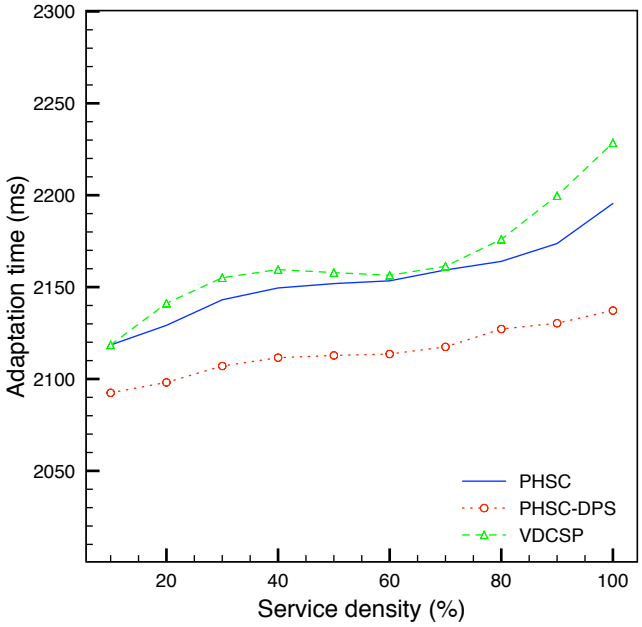

Figure 8.2: Composition time with respect to service density

The performance of both protocols, PHSC and VDCSP, was affected by the service density. Although a high service density provides robustness to the ecosystem, it also has a negative effect: as the density of services increases, the number of participants for the composition also increases. This implies a greater number of messages required to find a solution. However, figures 8.1 and 8.2 show that the PHSC (in time and consumed messages) performed better than the protocol VDCSP.

8.1.3 Effects of Scaling

In order to study the effect of scaling on the performance of protocols, we set the service density at 50 percent, with a composition order of five services. Additionally, we progressively increased the number of agents involved in a spatial area of 50 m^2. For each simulation, after the service composition we identified the number of messages consumed and the amount of time used to achieve the service composition. Results show that the performance of both protocols was affected by scalability issues. As the number of agents increased and the spatial environment area remained fixed, the time and number of messages required to achieve a service composition increased (figures 8.4 and 8.3). Simulations indicated that

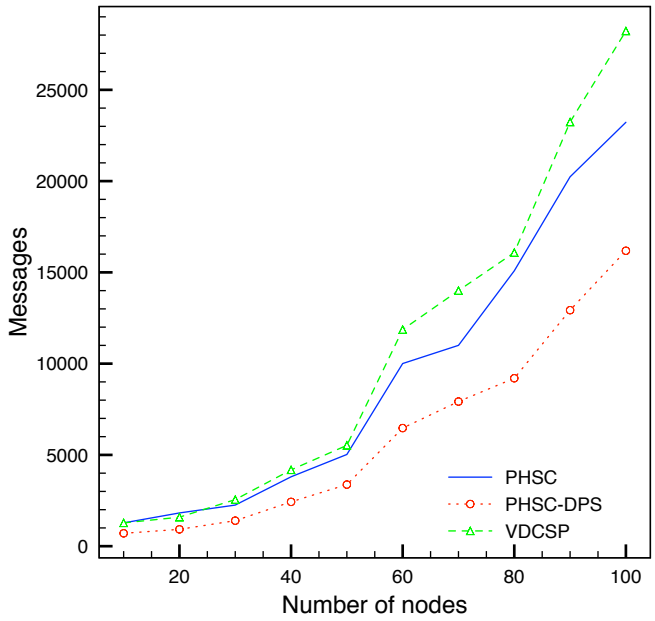

Figure 8.3: Number of messages with respect to number of nodes

per composition, the number of messages consumed by PHSC was an average of 9318.6 messages (14.13 percent) lower than VDCSP, and the time used for composition decreased by 0.39 percent compared to VDCSP.

8.1.4 Effects of Composition Order

To examine the effects of composition order on the performance of the protocols, we set the service density at 50 percent, the number of agents at 50, and the composition order was progressively increased from three to five, to seven and to nine in a spatial area of 50 m^2. For each simulation, after the service composition we identified the number of messages consumed and the amount of time used to achieve the service composition. Figures 8.5 and 8.6 show the negative effect of the order of composition over time and the number of messages required to compose a service. Results show that the PHSC protocol performed better than the VDCSP protocol (in terms of messages and time).

8.2 Pervasive Hybrid Service Adaptation

This section evaluates the performance of our PHSC with the DPS heuristic by varying scale, mobility and service density, as in the previous section, and instead of the order of

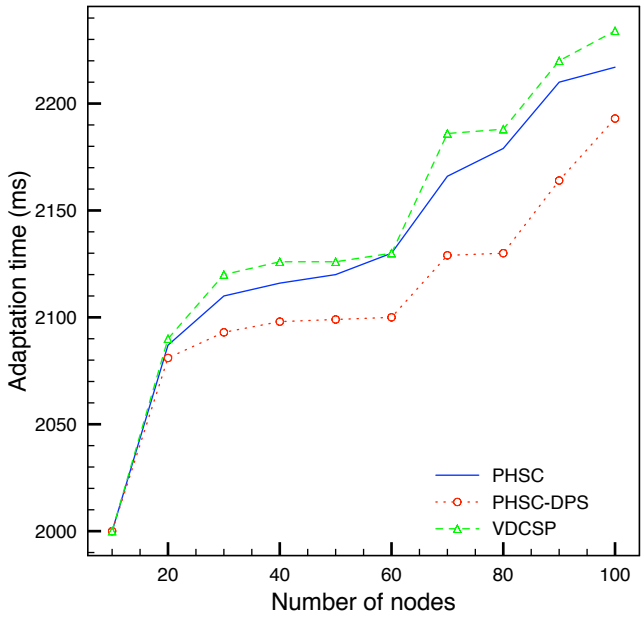

Figure 8.4: Composition time with respect to number of nodes

composition we will focus on the adaptation order. Service adaptation order means the number of elements of the composite service to be adapted. This section also evaluates the AG's ability to adapt in order to deal with changes in the environment due to the agents' mobility. Our experimental scenario consisted of agent-managed mobile devices that were able to provide one or more services in a dynamic environment. These devices were connected to each other using an ad hoc network. We considered adaptations of several orders (in terms of the number of services that the AG needs to adapt in order to fulfill the task).

We compared the performance of our DPS heuristic in adapting services with the results of using the PHSC and VDCSP algorithms. The aim of the comparison is to identify the improvement of the performance in message utilization, composition time and adaptation time when our PHSC protocol and DPS heuristic is used (in dynamic environments). At the same time we want to determine the effects of varying the service density, adaptation order, scale and mobility on the same metrics.

8.2.1 Scenario Description

We use the same scenario described previously in section 8.1. However, this time simulation was done for a service composition length of five with adaptation orders from one to four. For each simulation, we identified the amount of time taken and the number of

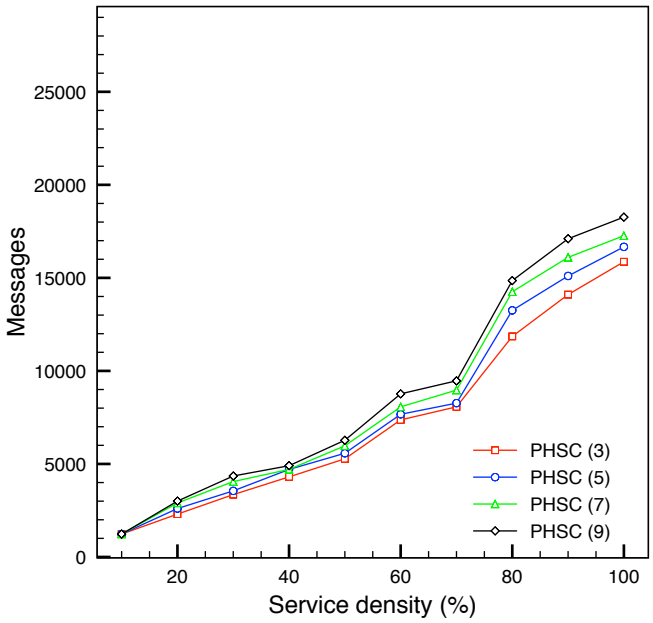

Figure 8.5: Number of messages with respect to composition length

messages consumed to achieve the adaptation (values are the average of 100 simulations). Again, as in the previous section 8.1, we utilized predetermined similarity values and did not consider the effect of network conditions on the performance of the three protocols.

8.2.2 Effects of Service Density

In order to analyze effects of service density on the performance of each protocol, we used a scenario with 50 static devices, a composition length of five services and an adaptation order of one. For each simulation, after the service composition we turned off participating agents that contributed to the composite service; then we identified the number of messages consumed and the amount of time used to achieve the service adaptation. The simulation was carried out using service densities of 10 - 100 percent in a spatial area of 50 m^2. Results indicated that per adaptation, the number of messages consumed by PHSC-DPS (figure 8.7) was an average of 5,669 messages (54 percent) lower than PHSC, and the time used for adaptation decreased by 2.1173 ms (2.27 percent) compared to PHSC (figure 8.8). The reduction of consumed messages is because PHSC needs to restart the composition process each time a participant agent is no longer available to provide its service (i.e. it seeks the solution from scratch).

Although a high service density provides robustness to the ecosystem, it also has a

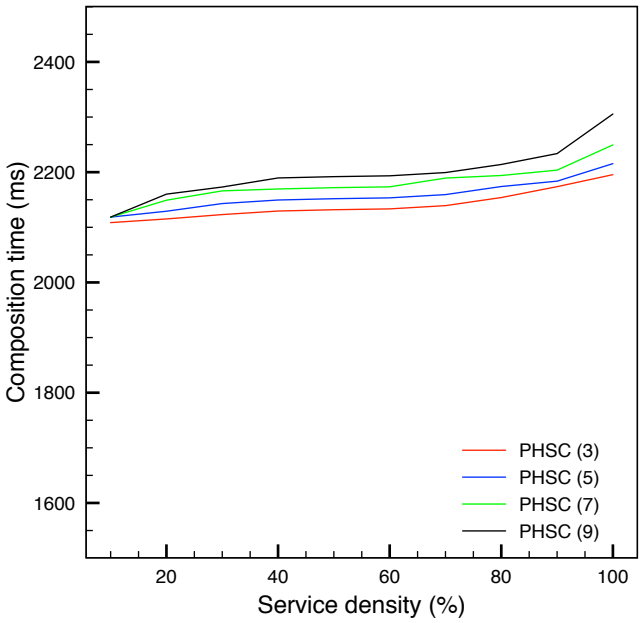

Figure 8.6: Composition time with respect to composition length

negative effect: as the density of services increases, the number of participating agents for the composition and adaptation also increases. This implies a greater number of messages required to find a solution. However, figure 8.7 and figure 8.8 show that the PHSC-DPS (in time and consumed messages) performed better than the protocols PHSC and VDCSP when the service density was high.

8.2.3 Effects of Scaling

In order to study the effect of scaling on the performance of adaptation, we set the service density at 50 percent, a composition order of five services, and an adaptation order of one service. For each simulation, after the service composition we turned off a participating agent that contributed to the composite service; then we identified the number of messages consumed and the amount of time used to achieve the service adaptation. In addition, we progressively increased the number of autonomous devices involved in a spatial area of 50 m^2.

Results show that as the number of autonomous agents increased and the spatial environment area remained fixed, the time and number of messages required to achieve service composition and adaptation increased (figure 8.10 and 8.9), i.e., the performance the performance of the three protocols was affected by scalability issues.

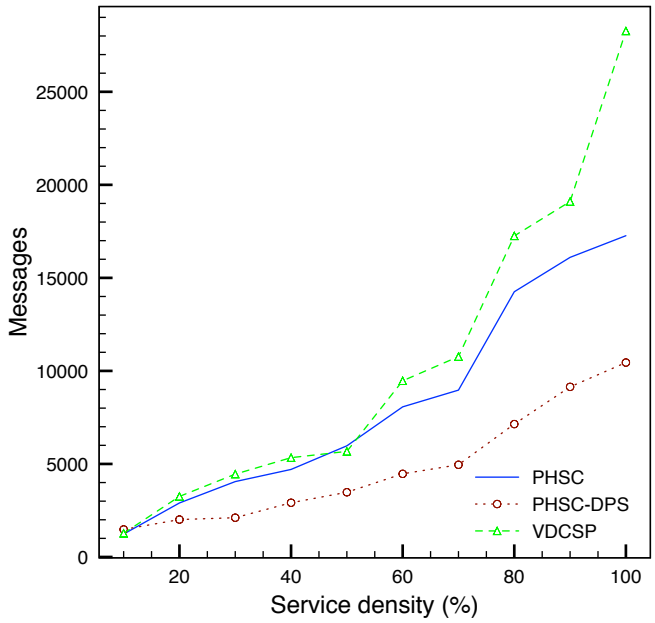

Figure 8.7: Number of messages with respect to service density

8.2.4 Effects of Adaptation Order

To examine the effects of adaptation order on the performance of the protocols, we set the service density at 50 percent, the number of autonomous devices at 50, and the composition order at five services. In addition, we progressively increased the adaptation order from one to four in a spatial area of 50 m^2. For each simulation, after the service composition we turned off a participating agent that contributed to the composite service; then we identified the number of messages consumed and the amount of time used to achieve the service adaptation. We repeated the last procedure, varying it by turning off two, three, and four participants.

Figures 8.12 and 8.11 show the negative effect of the order of adaptation on time and the number of messages required to adapt a service to changes in the environment. Results show that the PHSC-DPS protocol performed better than the PHSC and VDCSP protocols (in terms of messages and time). However, as the adaptation order went higher, the performance of the PHSC-DPS protocol decreased. The performance of the VDCSP and PHSC protocols remained almost the same because they did not include any strategy to adapt the composed service when a participating agent became unavailable. Therefore, in order to adapt the composed service, they restarted the service composition process from

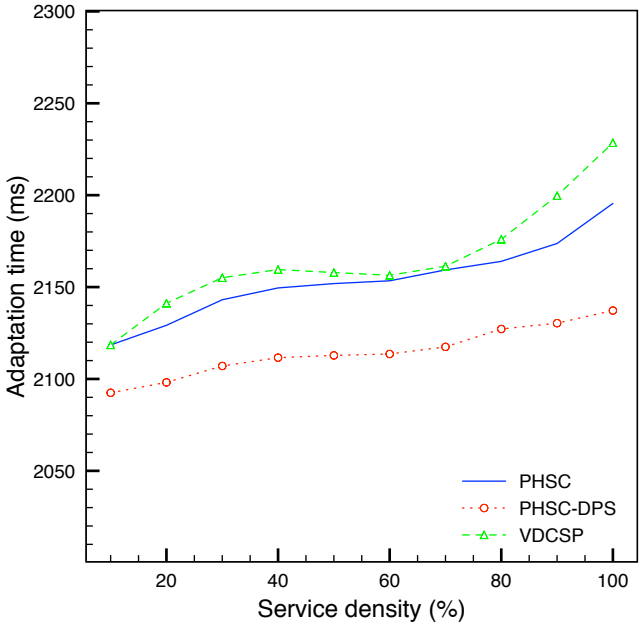

Figure 8.8: Adaptation time with respect to service density

scratch.

8.2.5 Effects of Mobility

In order to analyze the effects of mobility on the two protocols, we set the service density at 50 percent, the number of autonomous devices at 50, the composition order at five services and a random way-point mobility model. In this case the network topology and adaptation order could change (from one to five), and an autonomous agent randomly moved in an area of 150 x 500 m. For the sake of simplicity and in order to have control over the scenario, we configured the environment with 50 fixed devices and one mobile device. The position of the mobile devices was randomly selected within a fixed area of 150 x 500 m and then moved lineally to the selected position with a consistent random speed. For each simulation we identified the percentage of time during which the service stayed available (from the service composition until the end of the simulation). We repeated the experiments for different travel speeds (1 - 10 m/s).

The results showed that mobility at high speed had an adverse effect (figure 8.14) on the availability of the service, because the time during which the participating agents were in the transmission range was small, while the number of messages required to continuously

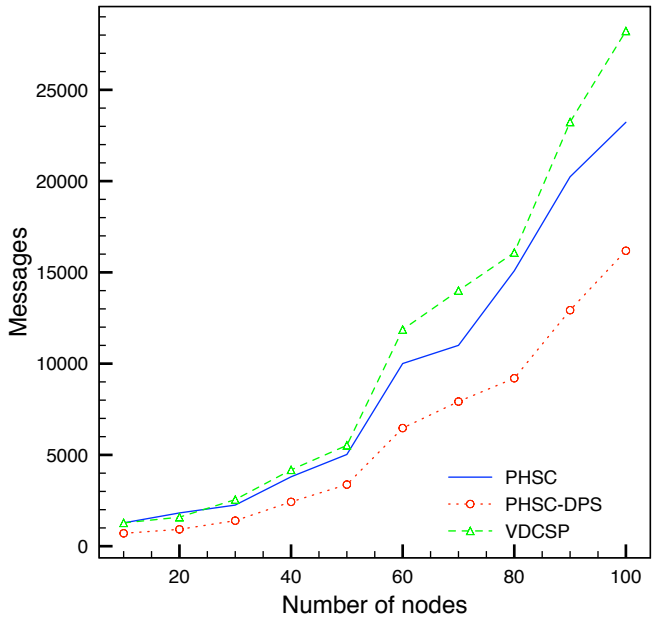

Figure 8.9: Number of messages with respect to number of nodes

adapt the composed service and maintain the service availability varied (figure 8.13) due to random displacement of the autonomous devices in their environment. One issue concerning the use of the random way-point mobility model for representing human mobility behavior has to do with the sharp turn [149]. Sharp turn occurs whenever there is a direction change in the range of 90 - 180 degrees. This problem can be eliminated by allowing past direction to affect future direction. The Gauss-Markov mobility model [150] solves this problem by achieving more realistic movement of autonomous devices.

8.2.6 Conclusion

In this chapter we presented a dynamic DisCSP model for pervasive service composition in dynamic environments, and a new heuristic useful for adapting partial solutions using an asynchronous backtracking algorithm to solve the dynamic DisCSP. The model provides a technique for adapting services without restarting the service composition process each time that a participating agent is unavailable or leaves the AG. Through simulation, we have shown that our protocol is able to maintain service availability despite the dynamism of the environment. We have also discussed some drawbacks of the proposed protocol in the context of network segmentation.

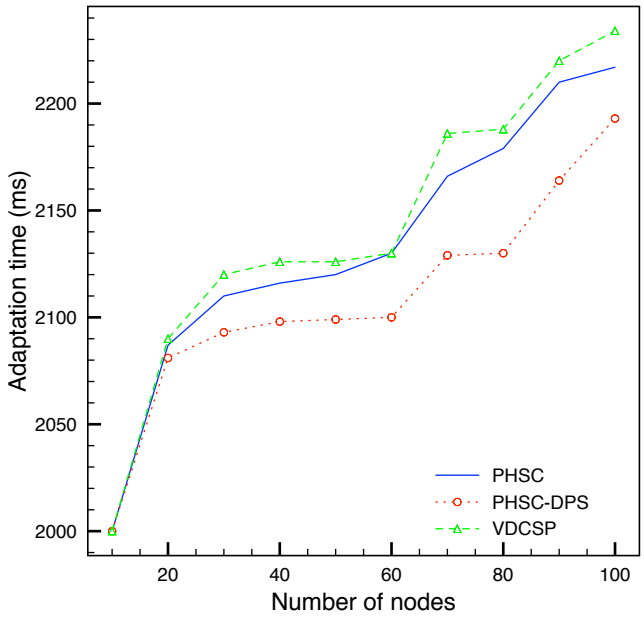

Figure 8.10: Adaptation time with respect to number of nodes

Our current research includes the design of composition techniques based on learning with multihop broadcasting. Multihop composition was not considered in this paper because one hop is enough to discover autonomous devices in the user's environment. However, even though the resources needed to provide a service may not be in the user's vicinity, the proposal can exploit them by using a directed multihop broadcast protocol. Moreover, the knowledge acquired may be applied as we experiment with autonomous devices in the real world.

Most solutions designed for the composition of adaptive pervasive services are based on dedicated infrastructure. These solutions use notions such as central servers, stable nodes and reliable communications channels, and include proposals as [139] [140] [141]. Most of these approaches involve preconfigured composition mechanisms residing on dedicated machines with high resources. Furthermore, some authors, such as [136], [142], only consider the initial composition of pervasive services. Some other authors have proposed protocols and frameworks for the adaptation of pervasive services in slightly dynamic environments; Karmouch and Nayak [143], for example, have proposed a disCSP model for service composition. They used a Quality of Service (QOS)-based approach to adapt the service composition (in order to determine the quality level, bandwidth, delay, loss and jitter were used in the network). However, the framework proposed by Karmouch assumes

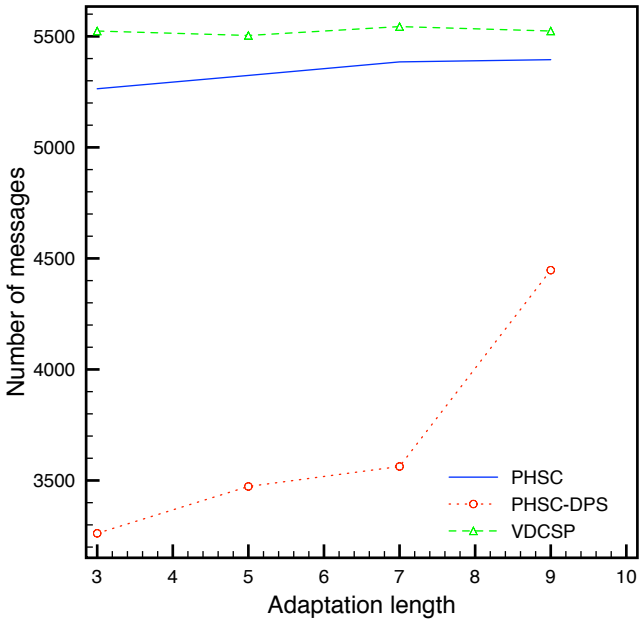

Figure 8.11: Number of messages with respect to adaptation length

that all the components of the instance under consideration, such as variables, domains and constraints, are completely known before it is modeled and solved, and do not change either during or after the modeling and solving. However, it has long been observed that such assumptions do not hold true in many situations.

In this paper we presented a dynamic DisCSP model for pervasive service composition in dynamic environments, and a new heuristic useful for adapting partial solutions using an asynchronous backtracking algorithm for solving the dynamic DisCSP. The model provides a technique for adapting services without restarting the service composition process each time that a participating agent is unavailable or leaves the AG. Through simulation, we have shown that our protocol is able to maintain service availability despite the dynamism of the environment. We have also discussed some drawbacks of the proposed protocol in the context of network segmentation. Moreover, the knowledge acquired may be applied as we experiment with real devices in the real world.

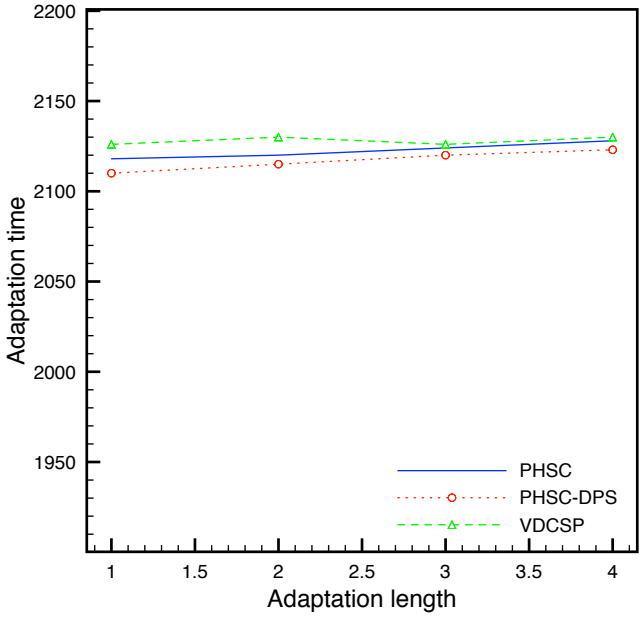

Figure 8.12: Adaptation time with respect to adaptation length

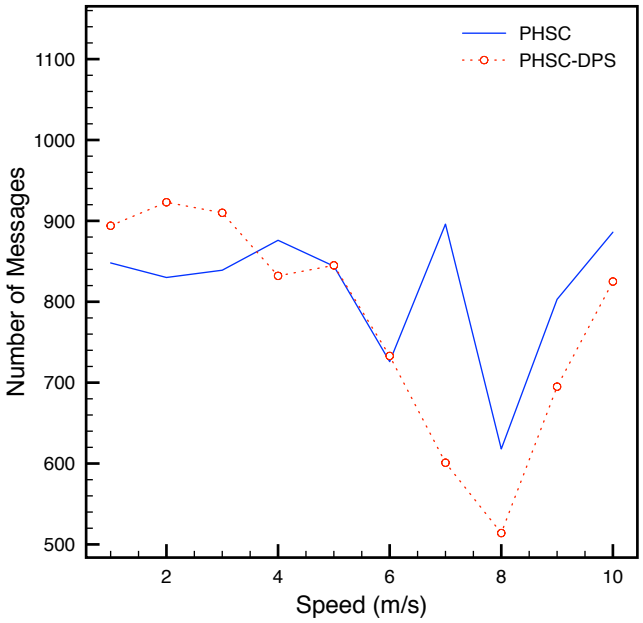

Figure 8.13: Number of messages with respect to mobility

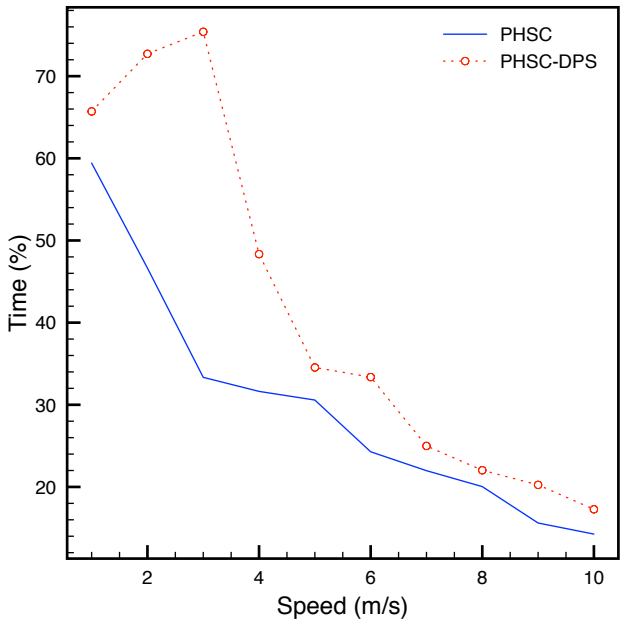

Figure 8.14: Service availability with respect to adaptation length

Chapter 9

Conclusions and Future Work

9.1 Work Context

The origin of this thesis is a relevant interest in problems within user-centered pervasive systems and the composition of their hybrid services. Motivations include the work of collaborative autonomous systems in the context of such booming areas as smart cities and the internet of things, with applications such as security, target monitoring, traffic control and health care. From the intersection of these two areas, two important issues emerge.The first is a suitable technique for the dynamic composition of hybrid services in these user-centered pervasive systems. SOA-based proposals, which are effectively addressed, do not provide suitable mechanisms for dealing with services involving physical dependencies in the user's environment; they are limited mainly to software-only systems. The second issue concerns the need for suitable architectures. Considering the physical elements of pervasive systems as autonomous cooperating nodes of ad hoc networks is an attractive approach that can lead to the decentralization and mobility of pervasive environments. Then we need to provide an approach to provide suitable mechanisms for hybrid service adaptation in dynamic environments.

9.2 Contributions

Concerning the first and second issues, this research has developed a model for the dynamic composition and adaptation of hybrid services. As the pervasive hybrid service notion is not yet well defined in the research literature, the first step towards this focus included a literature study of service-oriented approaches and the various aspects of hybrid services.

More precisely, we aimed to address two principal issues: what are the requirements of pervasive hybrid service composition and adaptation, and why it is difficult to satisfy these composition and adaptation requirements with current approaches. The result of this phase has been a survey of the main research directions in the area of service-oriented pervasive computing, service composition and service adaptation. In view of the limitations identified in current research, a novel approach to designing ecosystems of pervasive hybrid services has been proposed. A summary of the main contributions of our research is given below.

- A novel model of a service ecosystem based on a social metaphor, formed as groups of distributed devices, modeled by autonomous agents, all working towards the composition and adaptation of pervasive hybrid services. This model includes the basis for an agent communication language based on social obligations that provides support in the design of interaction mechanisms suitable for flexible organizations such as ecosystems.

- A distributed constraint satisfaction problem model for hybrid service composition in mobile and ad hoc networks, and a distributed protocol for service composition (i.e., for solving the respective distributed constraint satisfaction problem) utilizing an asynchronous backtracking algorithm for solving. Simulation results show the performance of our method in the composition of hybrid services.

- An extension to the distributed constraint satisfaction problem model that becomes dynamic in mobile and ad hoc networks and a respective heuristic for adapting the solution of the particular dynamic and distributed constraint satisfaction problem. Capability adaptation is a requirement: the assumption that a composed hardware service can remain static once generated is false, as the composed hardware service needs to be reconfigured along with the resources available in the user's environment. Simulation results show the performance of the solution.

9.3 Perspectives

Research work is still needed to make our proposal a practically usable tool for the development and deployment of pervasive hybrid services. Some of the most critical open research issues we have identified include:

- Social Ecosystem Metaphor. Despite the promises of the ecosystem approach, the way towards the deployment of usable and effective ecosystems of pervasive hybrid services still requires answers to several challenging questions. How can an ecosystem's members and the social obligations lead to suitable, useful, and controllable forms of organization? How can their dynamics be controlled to ensure continuous service availability in open environments? Should ecosystem members have the ability to learn? What shape should an actual software infrastructure have in order to support learning by the ecosystem's members? All of these, and many further questions we may have overlooked, open up fascinating areas of research.

- Composing Pervasive Hybrid Services. Although in this thesis we provide a solution for the composition of hybrid services based on resources within a user's environment, no consideration was given to situations where there are available resources and services but agents do not know how to interact to perform the service composition. Further research is required accounting for scenarios where agents need to learn how to interact with each other in order to achieve some specific objective.

- Adapting Pervasive Hybrid Services. Although the solutions provided in this thesis account for the adaptation of hybrid services based on resources within a user's environment, no consideration was given to the problem of environment characterization to achieve the most suitable adaptation. Further research is required to account for scenarios where devices face new problems in unknown environments and need to learn how to perform the adaptation.

The objective of this work was to propose the use of agent federations to provide hybrid services in an ecosystem of pervasive services. The potential of multiagent systems and ecosystem-inspired approaches to develop and deploy pervasive hybrid services is very large and the topic is now open for achievements.

References

[1] A. Dohr, R. Modre-Opsrian, M. Drobics, D. Hayn, and G. Schreier, "The internet of things for ambient assisted living," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pp. 804–809, April 2010.

[2] C. Fischer and H. Gellersen, "Location and navigation support for emergency responders: A survey," *Pervasive Computing, IEEE*, vol. 9, pp. 38–47, Jan 2010.

[3] Y. Chon and H. Cha, "Lifemap: A smartphone-based context provider for location-based services," *Pervasive Computing, IEEE*, vol. 10, pp. 58–67, April 2011.

[4] H. Cai, C. Peng, L. Jiang, and Y. Zhang, "A novel self-adaptive fault-tolerant mechanism and its application for a dynamic pervasive computing environment," in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2012 15th IEEE International Symposium on*, pp. 48–52, April 2012.

[5] M. Fahad, O. Boissier, P. Maret, N. Moalla, and C. Gravier, "Smart places: Multi-agent based smart mobile virtual community management system," *Applied Intelligence*, vol. 41, no. 4, pp. 1024–1042, 2014.

[6] M. Musolesi and C. Mascolo, "Car: Context-aware adaptive routing for delay-tolerant mobile networks," *Mobile Computing, IEEE Transactions on*, vol. 8, pp. 246–260, Feb 2009.

[7] D. Amendola, F. De Rango, K. Massri, and A. Vitaletti, "Neighbor discovery in delay tolerant networking using resource-constraint devices," in *Wireless Days (WD), 2013 IFIP*, pp. 1–3, Nov 2013.

[8] A. Manzalini, N. Brgulja, C. Moiso, and R. Minerva, "Autonomic nature-inspired eco-systems," in *Transactions on Computational Science XV* (M. Gavrilova, C. Tan, and C.-V. Phan, eds.), vol. 7050 of *Lecture Notes in Computer Science*, pp. 158–191, Springer Berlin Heidelberg, 2012.

[9] S. Jones, "Toward an acceptable definition of service [service-oriented architecture]," *Software, IEEE*, vol. 22, pp. 87–93, May 2005.

[10] M. Viroli, D. Pianini, S. Montagna, and G. Stevenson, "Pervasive ecosystems: A coordination model based on semantic chemistry," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, (Trento, Italy), pp. 295–302, ACM, 2012.

[11] S. Kalasapur and M. Kumar, "Resource adaptive hierarchical organization in pervasive environments," in *Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International*, pp. 1–8, January 2009.

[12] X. Gu, H. Shi, and J. Ye, "A hierarchical service discovery framework for ubiquitous computing," in *Pervasive Computing and Applications, 2008. ICPCA 2008. Third International Conference on*, vol. 1, pp. 307–312, October 2008.

[13] M. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," in *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pp. 3–12, Dec 2003.

[14] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, pp. 38–45, Nov 2007.

[15] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng, "Composing adaptive software," *Computer*, vol. 37, pp. 56–64, July 2004.

[16] J. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw, "Task-based adaptation for ubiquitous computing," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 36, pp. 328–240, May 2006.

[17] S.-F. Chang and A. Vetro, "Video adaptation: Concepts, technologies, and open issues," *Proceedings of the IEEE*, vol. 93, pp. 148–158, Jan 2005.

[18] D. Georgakoulos and M. P. Papazoglu., *Service-Oriented Computing*. Mas- Masachusetts Institute of Technology, 2009.

[19] T. Erl, *SOA Principles of Service Design*. Prentice Hall, 2008.

[20] A. Rostampour, A. Kazemi, F. Shams, P. Jamshidi, and A. Azizkandi, "Measures of structural complexity and service autonomy," in *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pp. 1462–1467, Feb 2011.

[21] Y. Dai, Y. Feng, Y. Zhao, and Y. Huang, "A method of uddi service subscription implementation," in *Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on*, pp. 661–666, June 2014.

[22] L. Wu, Y. He, D. Wu, and J. Cui, "A novel interoperable model of distributed uddi," in *Networking, Architecture, and Storage, 2008. NAS '08. International Conference on*, pp. 153–154, June 2008.

[23] N. Yulin, S. Huayou, L. Weiping, and C. Zhong, "Pdus: P2p-based distributed uddi service discovery approach," in *Service Sciences (ICSS), 2010 International Conference on*, pp. 3–8, May 2010.

[24] A. Haseeb, M. Matskin, and P. Kungas, "Distributed web services discovery middleware for edges of internet," in *Web Services (ICWS), 2010 IEEE International Conference on*, pp. 680–682, July 2010.

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

[26] C. L.-P. Caroline Funk, Carolin Ehm, "Support of stateful services in pervasive environments," in *Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops(PerComW'07)*, Computer Society, IEEE, 2007.

[27] U. Wajid, C. Marín, and N. Mehandjiev, "Optimizing service ecosystems in the cloud," in *The Future Internet* (A. Galis and A. Gavras, eds.), vol. 7858 of *Lecture Notes in Computer Science*, pp. 115–126, Springer Berlin Heidelberg, 2013.

[28] W.-M. Shen, P. Will, A. Galstyan, and C.-M. Chuong, "Hormone-inspired self-organization and distributed control of robotic swarms," *Autonomous Robots*, vol. 17, no. 1, pp. 93–105, 2004.

[29] R. Quitadamo, F. Zambonelli, and G. Cabri, "The service ecosystem: Dynamic self-aggregation of pervasive communication services," in *In SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, (Washington, DC, USA,), p. 1, IEEE Computer Society, 2007.

[30] A. Barros and M. Dumas, "The rise of web service ecosystems," *IT Professional*, vol. 8, pp. 31–37, Sept 2006.

[31] S. T. A. Pickett and M. L. Cadenasso, "The ecosystem as a multidimensional concept: Meaning, model, and metaphor," *Ecosystems*, vol. 5, no. 1, pp. 1–10, 2002.

[32] A. Omicini, "Nature-inspired coordination for complex distributed systems," in *Intelligent Distributed Computing VI* (G. Fortino, C. Badica, M. Malgeri, and R. Unland, eds.), vol. 446 of *Studies in Computational Intelligence*, pp. 1–6, Springer Berlin Heidelberg, 2013.

[33] J. Crowcroft, "Toward a network architecture that does everything," *Commun. ACM*, vol. 51, pp. 74–77, January 2008.

[34] M. Mamei and F. Zambonelli, *Field-Based Coordination for Pervasive Multiagent Systems*. Springer Series on Agent Technology, Springer-Verlag New York, Inc., 2005.

[35] C. Di Napoli, M. Giordano, Z. Németh, and N. Tonellotto, "Adaptive instantiation of service workflows using a chemical approach," in *Euro-Par 2010 Parallel Processing Workshops* (M. Guarracino, F. Vivien, J. Träff, M. Cannatoro, M. Danelutto, A. Hast,

F. Perla, A. Knupfer, B. Di Martino, and M. Alexander, eds.), vol. 6586 of *Lecture Notes in Computer Science*, pp. 247–255, Springer Berlin Heidelberg, 2011.

[36] M. Giordano and C. Di Napoli, "A chemical evolutionary mechanism for instantiating service-based applications," in *Parallel Architectures and Bioinspired Algorithms* (F. Fernández de Vega, J. I. Hidalgo Pérez, and J. Lanchares, eds.), vol. 415 of *Studies in Computational Intelligence*, pp. 267–286, Springer Berlin Heidelberg, 2012.

[37] F. De Angelis, J. Fernandez-Marquez, and G. Di Marzo Serugendo, "Self-composition of services in pervasive systems: A chemical-inspired approach," in *Agent and Multi-Agent Systems: Technologies and Applications* (G. Jezic, M. Kusek, I. Lovrek, R. J. Howlett, and L. C. Jain, eds.), vol. 296 of *Advances in Intelligent Systems and Computing*, pp. 37–46, Springer International Publishing, 2014.

[38] D. Coulter and E. Ehlers, "Biologically inspired obsolescence management in mobile agent systems: A dynamic, service oriented approach," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pp. 2063–2070, June 2011.

[39] G. Briscoe, S. Sadedin, and P. De Wilde, "Digital ecosystems: Ecosystem-oriented architectures," *Natural Computing*, vol. 10, no. 3, pp. 1143–1194, 2011.

[40] M. D. Peysakhov, R. N. Lass, and W. C. Regli, "Stability and control of agent ecosystems," in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, (New York, NY, USA), pp. 1143–1144, ACM, ACM, 2005.

[41] C. Villalba, M. Mamei, and F. Zambonelli, "A self-organizing architecture for pervasive ecosystems," in *Self-Organizing Architectures* (D. eyns, S. Malek, R. de Lemos, and J. Andersson, eds.), vol. 6090 of *Lecture Notes in Computer Science*, pp. 275–300, Springer Berlin Heidelberg, 2010.

[42] C. Villalba and F. Zambonelli, "Towards nature-inspired pervasive service ecosystems: Concepts and simulation experiences," *J. Network and Computer Applications*, vol. 34, no. 2, pp. 589–602, 2011.

[43] M. Mamei and F. Zambonelli, "Programming pervasive and mobile computing applications with the tota middleware," in *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, pp. 263–273, March 2004.

[44] J.-P. Banâtre, P. Fradet, and Y. Radenac, "Generalised multisets for chemical programming," *Mathematical. Structures in Comp. Sci.*, vol. 16, pp. 557–580, aug 2006.

[45] O. Flórez-Choque and E. Cuadros-Vargas, "A biologically motivated computational architecture inspired in the human immunological system to quantify abnormal behaviors to detect presence of intruders," in *Biologically Inspired Cooperative Computing* (Y. Pan, F. Rammig, H. Schmeck, and M. Solar, eds.), vol. 216 of *IFIP International Federation for Information Processing*, pp. 95–106, Springer US, 2006.

[46] L. N. de Castro and J. Timmis, "Artificial immune systems: A new computational intelligence approach," *Springer*, 2002.

[47] M. Wooldridge, *An Introduction to MultiAgent Systems*. John Wiley and Sons Ltd, 2nd ed., 2009.

[48] D. Weyns and M. Georgeff, "Self-adaptation using multiagent systems," *Software, IEEE*, vol. 27, pp. 86–91, Jan 2010.

[49] J. Rao and X. Su, "A survey of automated web service composition methods," in *Proceedings of the First International Conference on Semantic Web Services and Web Process Composition*, no. 12 in SWSWPC'04, (Berlin, Heidelberg), pp. 43–54, Springer-Verlag, 2004.

[50] R. Thiagarajan, A. Srivastava, A. Pujari, and V. Bulusu, "Bpml : a process modeling language for dynamic business models," in *Advanced Issues of E-Commerce and Web-Based Information Systems, 2002. (WECWIS 2002). Proceedings. Fourth IEEE International Workshop on*, pp. 222–224, June 2002.

[51] J. Pasley, "How bpel and soa are changing web services development," *Internet Computing, IEEE*, vol. 9, pp. 60–67, May 2005.

[52] S. Kalasapur, M. Kumar, and B. Shirazi, "Dynamic service composition in pervasive computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, pp. 907–918, July 2007.

[53] D. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, "Timed i/o automata: a mathematical framework for modeling and analyzing real-time systems," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pp. 166–177, Dec 2003.

[54] M. Capiluppi, L. Schreiter, P. Fiorini, J. Raczkowsky, and H. Woern, "Modeling and verification of a robotic surgical system using hybrid input/output automata," in *Control Conference (ECC), 2013 European*, pp. 4238–4243, July 2013.

[55] S. Mitra and S. Sastry, "Hybrid input output automata for composable conveyor systems," in *Automation Science and Engineering, 2009. CASE 2009. IEEE International Conference on*, pp. 29–29, August 2009.

[56] C. Norström, A. Wall, and W. Yi, "Timed automata as task models for event-driven systems," in *Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on*, pp. 182–189, 1999.

[57] M. Quottrup, T. Bak, and R. Zamanabadi, "Multi-robot planning : a timed automata approach," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 5, pp. 4417–4422, April 2004.

[58] M. Sharafi, "Extending team automata to evaluate software architectural design," in *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pp. 393–400, July 2008.

[59] C. Herrero and J. Oliver, "Extended cooperating automata," in *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, vol. 1, pp. 402–408, Oct 2003.

[60] R. Kazhamiakin, P. Pandya, and M. Pistore, "Timed modelling and analysis in web service compositions," in *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, p. 7 pp., April 2006.

[61] T. Murata, "Petri nets: properties, analysis and applications.," in *Proceedings IEEE*, vol. 77(4), pp. 541–580, 1989.

[62] S. Narayanan and S. Mcllraith, "Simulation, verification and automated composition of web services," in *In Proceedings of the 11th International World Wide Web Conference*, (New York, NY, USA), pp. 77–88, ACM Press, 2002.

[63] X. Yi and K. Kochut, "A cp-nets-based design and verification framework for web services composition," in *Proceedings of the IEEE International Conference on Web Services* (J. H., L. L., and Z. L.-J., eds.), (San Diego, CA.), pp. 756–760, Computer Society Press, 2004.

[64] G. Salaün, L. Bordeaux, and M. Schaerf., "Describing and reasoning on web services using process algebra," in *Proceedings of the IEEE International Conference on Web Services*, (Los Alamitos, C.A.), pp. 43–50, IEEE Computer Society Press, 2004.

[65] M. Yokoo, *Distributed Constraint Satisfaction: Foundations of Cooperation in MultiAgent Systems*. Springer Series on Agent Technology, Berlin: Springer, 2001.

[66] K. Ghédira, *Constraint Satisfaction Problems: CSP Formalisms and Techniques*. John Wiley, 2013.

[67] C. Yang and M.-H. Yang, "Constraint networks: a survey," in *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, vol. 2, pp. 1930–1935, Oct 1997.

[68] C. Castro, "Binary csp solving as an inference process," in *Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on*, pp. 462–463, Nov 1996.

[69] B. Nadel, "Representation selection for constraint satisfaction: a case study using n-queens," *IEEE Expert*, vol. 5, pp. 16–23, June 1990.

[70] A. Petcu, *A Class of Algorithms for Distributed Constraint Optimization*. IOS Press, 2009.

[71] R. Wallace and E. C. Freuder, "Stable solutions for dynamic constraint satisfaction problems.," in *In Proceedings of CP98*, 1998.

[72] G. Verfaillie and T. Schiex., "Solution reuse in dynamic constraint satisfaction problems," in *In Proceedings of the National Conference on Artificial Intelligence, AAAI-94*, 1994.

[73] C. Bessiere., "Arc-consistency for non-binary dynamic csps.," in *In Proc. of the 10th ECAI*, 1992.

[74] A. Fukunaga, "An improved search algorithm for minperturbation," in *Lecture Notes in Computer Science*, Springer, 2013.

[75] A. Horvath and D. Varro, "Dynamic constraint satisfaction problems over models," *Software Systems Model*, 2012.

[76] N. Nicoleta, *Constraint Satisfaction Techniques for Agent-Based Reasoning*. Springer, 2005.

[77] W. Sun, Z. Zhang, W. Chen, B. Peng, and Y. Xu, "Decentralized execution of composite service in manets," in *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE*, pp. 355–360, Dec 2008.

[78] Z. Ruttkay, "Fuzzy constraint satisfaction," in *Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the Third IEEE Conference on*, vol. 2, pp. 1263–1268, Jun 1994.

[79] A. Schmid, S. Padmanabhuni, and A. Schroeder, "A soft constraints-based approach for reconciliation of non-functional requirements in web services-based multi-agent systems," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pp. 711–718, July 2007.

[80] R. Young, R. Giachetti, and D. Ress, "A fuzzy constraint satisfaction system for design and manufacturing," in *Fuzzy Systems, 1996., Proceedings of the Fifth IEEE International Conference on*, vol. 2, pp. 1106–1112, Sep 1996.

[81] U. Montanari, "Networks of constraints: fundamental properties and applications to picture processing," *Information Sciences*, vol. 7, pp. 95–132, 1974.

[82] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems," *Artificial Intelligence*, vol. 58, no. 1-3, pp. 161–205, 1992.

[83] P. Morris, "The breakout method for pscaping from local minima.," in *In Proceedings of the Eleventh National Conference on Artificial Intelligence*, 1993.

[84] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Weley, 1984.

[85] C. Bessiere, I. Brito, A. Maestre, and P. Meseguer, "Asynchronous backtracking without adding links: A new member in the abt family," *Artificial Intelligence*, vol. 161, pp. 7–24, January 2005.

[86] M. Yokoo, "Asynchronous weak-commitment search for solving distributed constraint satisfaction problems," in *Principles and Practice of Constraint Programming — CP '95* (U. Montanari and F. Rossi, eds.), vol. 976 of *Lecture Notes in Computer Science*, pp. 88–102, Springer Berlin Heidelberg, 1995.

[87] O. Boissier, M. Colombetti, M. Luck, J. C. Meyer, and A. Polleres, "Norms, organizations, and semantics," *Knowledge Engineering Review*, vol. 28, no. 1, pp. 107–116, 2013.

[88] F. Cervantes, M. Occello, F. Ramos, and J.-P. Jamont, "Toward self-adaptive ecosystems of services in dynamic environments," in *Advances in Systems Science - Proceedings of the International Conference on Systems Science 2013* (J. Swiatek, A. Grzech, P. Swiatek, and J. M. Tomczak, eds.), vol. 240, pp. 671–680, Springer, 2013.

[89] M. A. Arbib, "Review of 'theory of automata' (salomaa, a.; 1969).," *IEEE Transactions on Information Theory*, vol. 16, no. 5, pp. 652–653, 1970.

[90] M. Barbuceanu and M. S. Fox, "Cool - a language for describing coordination in multi-agent systems," *In Proceedings of the First International Conference on Multiagent Systems*, pp. 17–24, 1995.

[91] A. Fukada, A. Nakata, J. Kitamichi, T. Higashinoz, and A. Cavalli, "A conformance testing method for communication protocols modeled as concurrent dfsms. treatment of non-observable non-determinism," in *Information Networking, 2001. Proceedings. 15th International Conference on*, pp. 155–162, 2001.

[92] R. Fernández and U. Endriss, "Abstract models for dialogue protocols," *Journal of Logic, Language and Information*, vol. 16, no. 2, pp. 121–140, 2007.

[93] A. K. Chopra and M. P. Singh, "Contextualizing commitment protocol," in *5th International Joint Conference on Autonomous Agents and Multiagent Systems AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, pp. 1345–1352, 2006.

[94] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. London, UK, UK: Springer-Verlag, 1995.

[95] H. Mazouzi, A. E. F. Seghrouchni, and S. Haddad, "Open protocol design for complex interactions in multi-agent systems," in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, AAMAS '02, (New York, NY, USA), pp. 517–526, ACM, 2002.

[96] J.-L. KONING, "Operational semantics rules as a computational coordination mechanism in multi-agent systems," *International Journal of Intelligent Control and Systems*, vol. 12, pp. 167–178, June 2007.

[97] S. Paurobally, J. Cunningham, and N. R. Jennings, "Developing agent interaction protocols using graphical and logical methodologies," in *PROMAS* (M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, eds.), vol. 3067 of *Lecture Notes in Computer Science*, pp. 149–168, Springer, 2003.

[98] P. D. O'Brien and R. C. Nicol, "Fipa - towards a standard for software agents," *BT Technology Journal*, vol. 16, pp. 51–59, jul 1998.

[99] FIPA, "Fipa communicative act library specification."

[100] M. P. Singh, "Agent communication languages: Rethinking the principles," *Computer*, vol. 31, pp. 40–47, dec 1998.

[101] M. P. Singh, "A social semantics for agent communication languages," *Issues in Agent Communication*, pp. 31–45, 2000.

[102] B. Gaudou, A. Herzig, D. Longin, and M. Nickles, "A new semantics for the fipa agent communication language based on social attitudes," *Proceedings of the 2006 Conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva Del Garda, Italy*, pp. 245–249, 2006.

[103] O. Gutiérrez, "Multiagent systems interaction through social norms," *Disertation*, 2009.

[104] O. Boissier, M. Colombetti, M. Luck, J.-J. Meyer, and A. Polleres, "Norms, organizations, and semantics," *The Knowledge Engineering Review*, vol. 28, no. 1, pp. 107–116, 2013.

[105] M. Rob and M. Shanahan, "The event calculus in classical logic - alternative axiomatisations," *Electron. Trans. Artif. Intell. 3(A)*, pp. 77–105, 1999.

[106] J. O. Gutierrez-Garcia, F. F. Ramos-Corchado, and J.-L. Koning, "From obligations to organizational structures in multi-agent systems," *Proceedings of the 11th Pacific Rim international Conference on Multi-Agents: intelligent Agents and Multi-Agent Systems.*, vol. 5357, pp. 206–213, 2008.

[107] M. Shanahan, "The event calculus explained," *Artificial Intelligence Today*, pp. 409–430, 1999.

[108] R. Miller and M. Shanahan, "Some alternative formulations of the event calculus, in computational logic," *Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski, Part II, A.C. Kakas and F. Sadri, Eds., Lecture Notes in Computer Science*, vol. 2408, pp. 452–490, 2002.

[109] F. Lopez and M. Luck, "Modelling norms for autonomous agents," in *Proceedings of Fourth Mexican International Conference on Computer Science*, pp. 238 – 245, IEEE Computer Society, 2003.

[110] J. F. Hübner, O. Boissier, and R. H. Bordini, "A normative programming language for multi-agent organisations," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 1-2, pp. 27–53, 2011.

[111] J. R. Searle and D. Vanderveken, *Foundations of Illocutionary Logic*. Cambridge University Press, Cambridge, 1985.

[112] N. Fornara and M. Colombetti, "Operational specification of a commitment-based agent communication language," *In Proceedings of the First international Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, pp. 536–542, 2002.

[113] M. P. Singh and M. N. Huhns, *Service-Oriented Computing; Semantics, Processes, Agents*. John Wiley and Sons Ltd, 2005.

[114] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: formalization and algorithms," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 10, pp. 673–685, Sep 1998.

[115] D. Chen, L. Wang, A. Y. Zomaya, M. Dou, J. Chen, Z. Deng, and S. Hariri, "Parallel simulation of complex evacuation scenarios with adaptive agent models," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, pp. 847–857, March 2015.

[116] N. M. Ceriani, A. M. Zanchettin, P. Rocco, A. Stolt, and A. Robertsson, "Reactive task adaptation based on hierarchical constraints classification for safe industrial robots," *IEEE/ASME Transactions on Mechatronics*, vol. 20, pp. 2935–2949, Dec 2015.

[117] H. Kim, Y. Yoon, and H. Park, "Adaptation method for level of detail (lod) of 3d contents," in *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pp. 879–884, Sept 2007.

[118] L. F. G. Preciado, S. E. Vargas, J. F. C. Alvarez, G. T. Blanco, and F. F. R. Corchado, "Simplifying the design of interactive simulations of deformable objects," *IEEE Latin America Transactions*, vol. 14, pp. 391–397, Jan 2016.

[119] W. Chen, Y. Jing, J. Wu, and W. Sun, "A dynamic execution path selection approach for composite services in manets," in *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pp. 1–4, Oct 2008.

[120] W. Chen, Z. He, G. Ren, and W. Sun, "Service recovery for composite service in manets," in *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pp. 1–4, Oct 2008.

[121] S. Han and Y. Zhang, "Design and implementation of service composition protocol based on dsr," in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2010 International Conference on*, pp. 323–328, Dec 2010.

[122] P. Choudhury, P. Dutta, S. Nandi, and N. Debnath, "Mobility aware distributed service composition framework in soa based manet application," in *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on* (July, ed.), pp. 1016–1021, 2012.

[123] J. Andersson, R. de Lemos, S. Malek, and D. Weyns, "Modeling dimensions of self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems* (B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, eds.), vol. 5525 of *Lecture Notes in Computer Science*, pp. 27–47, Springer Berlin Heidelberg, 2009.

[124] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel, "Towards a taxonomy of software change: Research articles," *J. Softw. Maint. Evol.*, vol. 17, pp. 309–332, sep 2005.

[125] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. Lo Presti, and R. Mirandola, "Moses: A framework for qos driven runtime adaptation of service-oriented systems," *Software Engineering, IEEE Transactions on*, vol. 38, pp. 1138–1159, Sept 2012.

[126] N. Taylor, P. Robertson, B. Farshchian, K. Doolin, I. Roussaki, L. Marshall, R. Mullins, S. Drüsedow, and K. Dolinar, "Pervasive computing in daidalos," *Pervasive Computing, IEEE*, vol. 10, pp. 74–81, Jan 2011.

[127] C. Baladron, J. Aguiar, B. Carro, L. Calavia, A. Cadenas, and A. Sanchez-Esguevillas, "Framework for intelligent service adaptation to user's context in next generation networks," *Communications Magazine, IEEE*, vol. 50, pp. 18–25, March 2012.

[128] V. W.-M. Kwan, F. C.-M. Lau, and C.-L. Wang, "Functionality adaptation: a context-aware service code adaptation for pervasive computing environments," in *Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference on*, pp. 358–364, Oct 2003.

[129] V. Schwartze, "An interactive user interface adaptation process," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pp. 546–547, March 2012.

[130] E. de Lara, Y. Chopra, R. Kumar, N. Vaghela, D. Wallach, and W. Zwaenepoel, "Iterative adaptation for mobile clients using existing apis," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, pp. 966–981, Oct 2005.

[131] M. Williams, Y. Yang, N. Taylor, S. McBurney, E. Papadopoulou, F. Mahon, and M. Crotty, "Personalized dynamic composition of services and resources in a wireless pervasive computing environment," in *Wireless Pervasive Computing, 2006 1st International Symposium on*, pp. 1–6, Jan 2006.

[132] C. Funk, A. Schultheis, C. Linnhoff-Popien, J. Mitic, and C. Kuhmunch, "Adaptation of composite services in pervasive computing environments," in *Pervasive Services, IEEE International Conference on*, pp. 242–249, July 2007.

[133] P.-A. Avouac, P. Lalanda, and L. Nigay, "Adaptable multimodal interfaces in pervasive environments," in *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pp. 544–548, Jan 2012.

[134] S. Hagen and A. Kemper, "Facing the unpredictable: Automated adaption of it change plans for unpredictable management domains," in *Network and Service Management (CNSM), 2010 International Conference on*, pp. 33–40, Oct 2010.

[135] J. Cervino, E. Kalyvianaki, J. Salvachua, and P. Pietzuch, "Adaptive provisioning of stream processing systems in the cloud," in *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pp. 295–301, April 2012.

[136] Y. Maurel, S. Chollet, V. Lestideau, J. Bardin, P. Lalanda, and A. Bottaro, "fanfare: Autonomic framework for service-based pervasive environment," in *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, pp. 65–72, June 2012.

[137] S. Komorita, M. Ito, H. Yokota, C. Makaya, B. Falchuk, D. Chee, and S. Das, "Loosely coupled service composition for deployment of next generation service overlay networks," *Communications Magazine, IEEE*, vol. 50, pp. 62–72, January 2012.

[138] K. Zielinnski, T. Szydlo, R. Szymacha, J. Kosinski, J. Kosinska, and M. Jarzab, "Adaptive soa solution stack," *Services Computing, IEEE Transactions on*, vol. 5, pp. 149–163, April 2012.

[139] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. chien Shan, "Adaptive and dynamic service composition in eflow," in *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, CAiSE '00, (London, UK, UK), pp. 13–31, Springer-Verlag, 2000.

[140] F. Cicirelli, A. Furfaro, and L. Nigro, "Integration and interoperability between jini services and web services," in *Services Computing, 2007. SCC 2007. IEEE International Conference on*, pp. 278–285, July 2007.

[141] K. Rajaram and C. Babu, "Template based soa framework for dynamic and adaptive composition of web services," in *Networking and Information Technology (ICNIT), 2010 International Conference on*, pp. 49–53, June 2010.

[142] W. Baldwin, T. Ben-Zvi, and B. Sauser, "Formation of collaborative system of systems through belonging choice mechanisms," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 42, pp. 793–801, July 2012.

[143] E. Karmouch and A. Nayak, "A distributed constraint satisfaction problem approach to virtual device composition," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, pp. 1997–2009, Nov 2012.

[144] G. Riley and T. Henderson, "The ns-3 network simulator," in *Modeling and Tools for Network Simulation* (K. Wehrle, M. Güneş, and J. Gross, eds.), pp. 15–34, Springer Berlin Heidelberg, 2010.

[145] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "Tinyos: An operating system for sensor networks," in *Ambient Intelligence* (W. Weber, J. Rabaey, and E. Aarts, eds.), pp. 115–148, Springer Berlin Heidelberg, 2005.

[146] A. Sobeih, W.-P. Chen, J. Hou, L.-C. Kung, N. Li, H. Lim, H. ying Tyan, and H. Zhang, "J-sim: a simulation environment for wireless sensor networks," in *Simulation Symposium, 2005. Proceedings. 38th Annual*, pp. 175–187, April 2005.

[147] J.-P. Jamont, M. Occello, and E. Mendes, "Decentralized intelligent real world embedded systems: A tool to tune design and deployment," in *Advances on Practical Applications of Agents and Multi-Agent Systems* (Y. Demazeau, T. Ishida, J. Corchado, and J. Bajo, eds.), vol. 7879 of *Lecture Notes in Computer Science*, pp. 133–144, Springer Berlin Heidelberg, 2013.

[148] C. Bettstetter, G. Resta, and P. Santi, "The node distribution of the random waypoint mobility model for wireless ad hoc networks," *Mobile Computing, IEEE Transactions on*, vol. 2, pp. 257–269, July 2003.

[149] J. Ariyakhajorn, P. Wannawilai, and C. Sathitwiriyawong, "A comparative study of random waypoint and gauss-markov mobility models in the performance evaluation of manet," in *Communications and Information Technologies, 2006. ISCIT '06. International Symposium on*, Oct 2006.

[150] B. Liang and Z. Haas, "Predictive distance-based mobility management for pcs networks," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1377–1384, Mar 1999.

Acronyms

ABT	Asynchronous Back Tracking. 22, 56, 57
ACL	Agent Communication Language. 34, 36, 49, 68
ACO	Ant Colony Optimization. 23
AG	Agent Group. 64, 65, 89, 96, 98
AWCS	Asynchronous Weak-Commitment Search. 22
BDI	Beliefs, Desires and Intentions. 36
BPEL	Business Process Execution Language. 17–20
BPML	Business Process Modeling Language. 17
CCS	Calculus of Communicating System. 20
CPN	Colored Petri Nets. 35
CSP	Constraint Satisfaction Problem. 16, 20, 21
DBA	Distributed Breakout Algorithm. 22
DFSM	Deterministic Finite State Machine. 35
DisCSP	Distributed Constraint Satisfaction Problem. v, 21, 22, 51–53, 56, 57, 96, 98
DPS	Dynamic Partial Solution. 64, 65, 89, 91, 93, 94
ERA	Environment, Reactive Rules and Agents. 22
FIPA	Foundation for Intelligent Physical Agents. 36
IAM	Individual Agent Manager. 70
MANET	Mobile and Ad Hoc Network. 9, 52
MAS	Multiagent System. 14, 35, 70
MASH	Multiagent Software Hardware Simulator. 68–70, 73, 85
PHSC	pervasive hybrid service composition. 56, 57, 64, 85, 87–89, 91, 93, 94
PN	Petri Nets. 35

PROMELA	Process Meta Language. 19
PSO	Particle Swarm Optimization. 23
QOS	Quality of Service. 98
SOA	Service Oriented Architecture. 3, 4, 7, 8, 14
UDDI	Universal Description, Discovery, and Integration. 8
UML	Unified Modeling Language. 35
VDCSP	Virtual Device Constraint Satisfaction Protocol. 85, 87–89, 91, 93, 94
XLANG	XML Language. 20
XML	Extensible Markup Language. 18
XPath	XML Path Language. 19

Annexes

Publications of the author

Articles

1. **Context Sensitive Ecosystem of Intelligent Environments**
   Francisco Cervantes, Rodolfo Ostos, Félix Ramos, *8th International Conference on Digital Intelligent Environments*, **México**, IEEE, 2012.

2. **Toward Self-Adaptive Ecosystems of Services in Dynamic Environments**
   Francisco Cervantes, Michel Occello, Félix Ramos, Jean-Paul, *Advances in Intelligent Systems and Computing*, **Polone**, Springer International Publishing, 2013.

3. **Designing the Web of Things as a Society of Autonomous Real/Virtual Hybrid Entities**
   Francisco Cervantes, Michel Occello, Félix Ramos, Jean-Paul, *International Workshop on Web Intelligence and Smart Sensing*, **France**, ACM, 2014.

4. **Simplifying the Design of Interactive Simulations of Deformable Objects**
   L. F. Gutierrez, F. Cervantes, S. Vargas, G. Torres, and Félix Ramos, *IEEE Latin America Transactions, Vol. 14, No. 1*, Jan. 2016.

Book chapter

1. **Coordinación Distribuida en Espacios Inteligentes Heterogéneos**
   Rodolfo Ostos, Francisco Cervantes, Félix Ramos, *La medicina como arte, ciencia, humanismo e investigación: XXXII Jornada Médica ISSSTE-UAEM*, **Universidad Autónoma del Estado de México**, 2014.