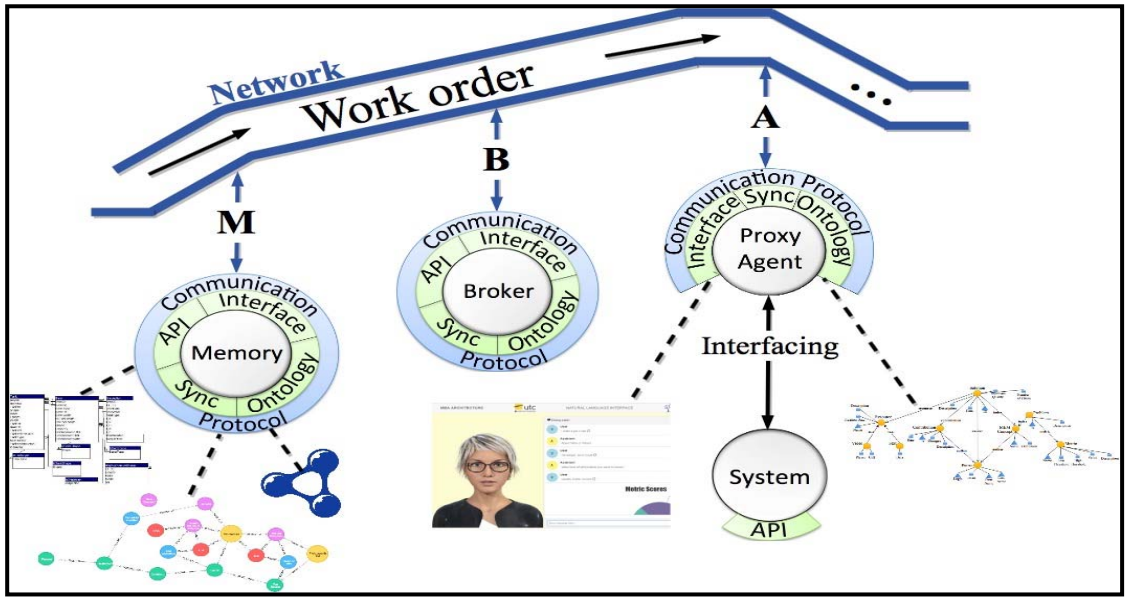
A framework for facilitating the development of systems of systems

Gregory Moro Puppi Wanderley

Par **Gregory MORO PUPPI WANDERLEY**

A framework for facilitating the development of systems of systems

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 27 juin 2018
Spécialité : Sciences et Technologies de l'Information et des Systèmes : Unité de recherche Heudyasic (UMR-7253)

D2425

Thèse

présentée pour l'obtention du grade de

Docteur de l'Université de Technologie de Compiègne

Spécialité : Sciences et technologies de l'information et des systèmes

Laboratoire Heudiasyc - UMR CNRS 7253

A Framework for Facilitating the Development of Systems of Systems

par

Gregory Moro Puppi Wanderley

Juin 2018

A Framework for Facilitating the Development of Systems of Systems

Thèse soutenue le 27 Juin 2018 devant le jury composé de :

M. Philippe BONNIFAIT	Professeur des universités Laboratoire Heudiasyc, UTC ***Président***
M. Jean-Paul A. BARTHÈS	Professeur émérite Laboratoire Heudiasyc, UTC ***Co-directeur de thèse***
M^{me} Marie-Hélène ABEL	Professeur des universités Laboratoire Heudiasyc, UTC ***Co-directrice de thèse***
M. Emerson C. PARAISO	Professeur Laboratoire PPGIa, PUCPR ***Co-directeur de thèse***
M^{me} Emmanuelle GRISLIN-LE STRUGEON	Maître de Conférences Laboratoire LAMIH, Univ. de Valenciennes ***Rapporteur***
M. Jean-Paul JAMONT	Maître de Conférences Laboratoire LCIS, Univ. Grenoble Alpes ***Rapporteur***
M. Milton P. RAMOS	Professeur Laboratoire de Systèmes Intelligents, TECPAR ***Rapporteur***
M^{me} Elsa NEGRE	Maître de Conférences Laboratoire LAMSADE, Univ. Paris-Dauphine
M. Claude MOULIN	Maître de Conférences Laboratoire Heudiasyc, UTC

À mes parents, Jérvis et Luzimeri

Acknowledgements

First and foremost I would like to thank my advisors. I am eternally grateful to Prof. Jean-Paul A. Barthès for accepting me as your student. I am very honored to have had such a fantastic and inspiring advisor. I could benefit from his unique insight, wisdom, dedication, and kindness. Thank you for your precious advice and guidance during this thesis. I will take all your lessons with me, every day.

Also, I am profoundly grateful to my co-advisor Prof. Marie-Hélène Abel. Thank you for your invaluable advice and extremely pertinent ideas along this entire thesis. I would like to show my gratitude for all the enthusiasm and encouragement to publish, and the opportunities for participating in various scientific events.

Furthermore, I am very thankful also to my co-advisor Prof. Emerson Cabrera Paraiso. Thank you for supporting me throughout my research path since I was a young undergraduate student. I wish to acknowledge all the advice, and the serenity that kept me on the right track to realize successfully this work.

I would like to thank my thesis committee members: Prof. Philippe Bonnifait, Prof. Emmanuelle Grislin-Le Strugeon, Prof. Jean-Paul Jamont, Prof. Milton P. Ramos, Prof. Elsa Negre, and Prof. Claude Moulin. Thank you for all of their guidance through this process; your discussion, pertinent ideas, and feedback have been absolutely invaluable.

My sincere thanks to Prof. Laurent Heurley and also to his student Elodie Vandenbergh, from the Université de Picardie Jules Verne, for the opportunity to cooperate in the CONSIGNELA project.

I also acknowledge the CNPq (National Council for Scientific and Technological Development) from Brazil for funding this thesis.

Last but surely not least, I wish to thank my family. To my parents, Jérvis and Luzimeri, I express my deepest gratitude for all their dedication, effort, unconditional support, encouragement and love, provided throughout my life. My special thanks also to my sister Madeleine for all her support.

Résumé

Le développement de Systèmes de Systèmes a pris de l'ampleur dans de nombreux domaines. Aujourd'hui, les applications complexes nécessitent que plusieurs systèmes développés indépendamment coopèrent ensemble, ce qui conduit au concept de Systèmes de Systèmes. Malgré une telle popularité, aucun consensus n'a pas encore pu être atteint sur une définition précise de ce que sont les Systèmes de Systèmes. De plus, le nœud du problème est que la plupart des applications sont encore construites à la main et développées de manière ad hoc, c'est-à-dire, sans contraintes et sans être guidées par une structure prédéfinie. Développer un système de systèmes à la main est une tâche herculéenne pour un architecte informatique, en lui demandant de créer un entrelacement de connexions entre les systèmes composant du Système de Systèmes pour qu'ils puissent coopérer. En raison d'un tel entrelas, la complexité et le couplage serré augmentent, et l'évolution des Systèmes de Systèmes devient plus difficile, nécessitant des efforts substantiels. Pour trancher le nœud gordien auquel font face les architectes de Systèmes de Systèmes, nous proposons dans cette recherche un "framework" générique pour faciliter le développement de Systèmes de Systèmes dans le cadre de l'ingénierie des systèmes. Notre approche introduit une nouvelle architecture que nous appelons MBA pour Memory-Broker-Agent. Pour tester notre framework, nous avons construit un système de systèmes dans le domaine du développement collaboratif de logiciel. Les résultats montrent que notre approche réduit la difficulté et l'effort de développement. Sur la base de ces résultats, nous avons créé une méthode originale pour construire un système de systèmes à travers notre framework. Nous avons testé le potentiel de notre méthode ainsi que les caractéristiques génériques de notre framework, en construisant avec succès et avec plus de précision un nouveau système de systèmes dans le domaine de la Santé.

Mots-Clés : Systèmes de Systèmes, Architecture, Systèmes Multi-agents, Gestion des Connaissances

Abstract

Building Systems of Systems (SoS) has gained momentum in various domains. Today, complex applications require to let several systems developed independently cooperate, leading to the moniker of SoS. Despite such popularity, no consensus has yet been reached about a precise definition of what SoS are. Moreover, the crux of the matter is that most applications are still handcrafted, being developed in an ad hoc fashion, i.e., freely and without being constrained by a predefined structure. Handcrafting SoS is an Herculean task for architects, requiring them to create an interwoven set of connections among SoS constituent systems for allowing cooperation. Because of the large number of interconnections, the complexity and tight coupling increase in SoS, and their evolution becomes more difficult, requiring substantial efforts from architects. To sever the Gordian knot faced by SoS architects, we propose in this research a generic framework for facilitating the development of SoS from a systems engineering perspective. Our approach is based on a novel architecture we call MBA for Memory-Broker-Agent. To test our framework we built an SoS for developing software collaboratively. Results show that our approach reduces the difficulty and effort for developing an SoS. Based on such results, we created an original method for building an SoS using our framework. We tested the potential of our method along with the generic features of our framework, by building a new SoS in the Health Care domain successfully and more accurately.

Keywords: Systems of Systems, Architecture, Multi-agent system, Knowledge Management

Contents

List of Figures

List of Tables

List of Acronyms

ACE4SD	Advanced Collaborative Environment for Software Development
AI	Artificial Intelligence
API	Application Programming Interface
APU	Auxiliary Power Unit
CONSIGNELA	Adapted electronical instructions
COTS	Commercial-Off-the-Shelf
CoQAS	Code Quality Analysis System
DB	Database
DoDAF	Department of Defense Architecture Framework
EclipA^2	Eclipse Augmented by Agent
EBNF	Extended Backus-Naur Form
ExAS	Extended Abstract Statements
FEDER	European Regional Development Fund
FTP	File Transfer Protocol
GEO	Global Earth Observation
GST	General System Theory
Guide4MBA	Guidelines for developing MBA SoS
HTTP	Hyper Text Transfer Protocol
HTML	Hypertext Markup Language
IDE	Integrated development environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
INCOSE	International Council on Systems Engineering
IP	Internet Protocol
ISO	International Organization for Standardization
JSON	JavaScript Object Notation

KM	Knowledge Management
KQML	Knowledge Query and Manipulation Language
LAN	Local Area Network
MAS	Multi-agent system
MBA	Memory-Broker-Agent
MeTRo	Method for Testing Robustness
MI	Maintainability Index
MMI	Man–Machine Interface
MODAF	Ministry of Defense Architecture Framework
NL	Natural Language
OMAS	Open Multi-Agent System
P2P	Peer-to-Peer
PA	Personal Assistant
PeWeS^2	Personalized Web Search System
ProPA	Proactive Personal Assistant
RDF	Resource Description Framework
SA	Service Agent
SDI	Strategic Defense Initiative
SOA	Service-oriented architecture
SoS	Systems of Systems
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UM	User Model
UML	Unified Modeling Language
VRE	Virtual Research Environment
XA	Transfer Agent
XML	Extensible Markup Language
WIMP	Windows, icons, menus, pointer

Chapter 1

Introduction

Ludwig von Bertalanffy conducted primordial studies culminating in the General System Theory (GST) (Bertalanffy, 1969). Since then, the need of developing complex systems successfully has led to the emergence of the field of systems engineering, and to the creation of the International Council on Systems Engineering (INCOSE). Currently, an increasing number of applications require letting several systems built independently work together, leading to the concept of *Systems of Systems* (SoS).

Building SoS from a systems engineering perspective, focused on connecting independent systems together, gained momentum with the United States Strategic Defense Initiative (SDI) from the late 1980s (Nielsen et al., 2015). Notwithstanding, over the years the growing interest and demand has made SoS quite popular, and nowadays they have been applied in a number of distinct domains, inter alia, to support manufacturing (Ford et al., 2012), to support military activities (Olivier et al., 2014), to improve the quality of the code (Wanderley et al., 2018), to help developing particle accelerator facilities (Friedrich et al., 2017), to improve understanding of medication prescriptions (Wanderley et al., 2017b).

Despite such a popularity, no consensus has yet been reached about a precise definition of what SoS are (Jamshidi, 2017). Moreover, the crux of the matter is that most of SoS are still handcrafted being developed in an ad hoc fashion (Gonçalves et al., 2015), i.e., freely without any guidance and without being constrained by a predefined structure.

Developing an SoS through ad hoc approaches is a Herculean task for SoS architects. First, the heterogeneous systems of an SoS are usually connected directly or point-to-point to each other, thus requiring a substantial effort. At the same time, one creates an interwoven set of connections among SoS constituent systems, augmenting complexity and tightening the coupling among them. As a consequence, cascading effects may occur and spread out through the entire SoS, causing a massive revision need for the architect. Moreover, evolving (i.e., changing) the SoS is also more difficult, for instance, when

adding a new system, the architect is required to create or update references across all the constituents impacted by such a change.

In this research, we aim to sever the Gordian knot faced by SoS architects. To try to understand and determine how we could support them, we began our studies by building a first experimental SoS. Because we had some past research experience in the domain of collaborative software development to improve code quality, we built the SoS in such a context. For supporting us to develop this first SoS, we experimented some well-studied approaches in our laboratory such as multi-agent systems and knowledge management. Our experiments culminated in the design of the approach we propose in this research which is an architecture called MBA for Memory-Broker-Agent, aiming at facilitating the development of an SoS from a systems engineering perspective.

When developing an SoS with the MBA approach, say our first SoS focused on code quality, an architect interfaces agents with systems like code quality analyzers or databases for letting them work together. SoS users such as software developers or managers are interfaced through Personal Assistant agents for requesting and receiving code quality information. Results show that the MBA architecture reduces the difficulty and effort for building an SoS. We detail the development of our first SoS and such obtained results in this thesis.

The experience of building our first SoS has also brought valuable insights which were used to improve our architecture. One of the main discoveries was that we could generalize it creating an MBA framework along with a method for building Systems of Systems. This has led us to test the potential of our approach to be applied to different domains. We ended by building a brand new SoS in the domain of Health Care which we also detail in this thesis.

1.1 Goals

- The main goal of this research is to design and to study a domain-independent framework for building Systems of Systems (SoS) from a systems engineering perspective. We call it MBA for Memory-Broker-Agent.

1.2 Hypothesis

We hypothesize that:

- *A framework based on a Memory-Broker-Agent architecture could facilitate the development of an SoS, regarding the encountered difficulties and the efforts needed for connecting the SoS constituent systems.*

1.3 Contributions

The main contributions of this research are:

- A domain-independent framework for facilitating the development of Systems of Systems (SoS) from a systems engineering perspective. Our framework is based on a novel architecture we call MBA (Memory-Broker-Agent), consisting of agents, brokers and memories.
- An original method for building Systems of Systems. The method consists of key points describing general guidelines for supporting SoS architects to develop MBA SoS accurately.

1.4 Document outline

This thesis is organized in nine chapters. This is the Chapter 1 and describes the main context and motivation, the goals, the hypothesis, and the contributions of this research.

Chapter 2 - Systems of Systems

The Chapter 2 approaches the central topic of this research which are Systems of Systems (SoS). In the chapter we present an overview of the fundamental concepts and definitions needed for understanding SoS. Moreover, we also present the state of the art for developing SoS related to the scope of this research. We finish it by making a discussion to show the main differences and relevance of the approach proposed by this research, compared to the state of the art.

Chapter 3 - Multi-agent systems

In Chapter 3 we provide an overview of the concepts and definitions of Multi-Agent Systems (MAS) related to the scope of this research. The goal is to provide the background needed for understanding MAS which is a key element in the approach proposed by this research. Readers with a reasonable knowledge in MAS can go directly to the end of this chapter, in the Section 3.2 (*Discussion: Why use MAS for supporting SoS?*). In this section, we give the reason for using an MAS in our approach, and how an MAS differs from an SoS.

Chapter 4 - An architecture for facilitating the development of Systems of Systems

In Chapter 4 we present the Memory-Broker-Agent (MBA) architecture which is the core of the approach proposed in this research. First, we provide an overview and explain the choice for each element of our architecture. Then, we present a formal definition of the model used by the MBA architecture, and then we describe each of its component elements. After that, we describe the crucial role ontologies play in our approach since they are the backbone of our architecture. Next, we present the *Work Order* which is the communication protocol provided by the MBA architecture. We finish by highlighting the advantages of the MBA architecture and it elements.

Chapter 5 - Extended functionalities provided by the MBA architecture

The Chapter 5 presents extended functionalities provided by the MBA architecture, and composing the framework of this research. In this chapter we describe in detail the *Proactive interfaces* which are the user interfaces provided by our approach for supporting the SoS user dimension. Then, we give a brief introduction of a *synchronization method* we provide for supporting SoS robustness, letting constituent systems remain coherent when they are restarted after going down. The method is fully detailed in the Appendix B.

Chapter 6 - The ACE4SD System of Systems

In Chapter 6 we describe a functional prototype, named *ACE4SD*, which was the first SoS we built in this research. The role of the ACE4SD was pivotal in our research, as it acted as a basis for designing and improving the MBA architecture, as well as for supporting us to move it towards a framework. Moreover, the ACE4SD SoS also aimed to validate the approach proposed in this research. First, we present the ACE4SD context and goals. Then, we detail its constituent systems describing their roles, and giving examples of interactions between them. After that, we detail the realization and implementation of our prototype.

Chapter 7 - Evaluation

The Chapter 7 describes the evaluation realized in the approach proposed by this research. First, we present the design of the experiments. Then, we describe the metrics

used to evaluate our approach. We finish by showing and discussing the results we obtained.

Chapter 8 - General guidelines for developing MBA SoS

In Chapter 8 we present a method describing general guidelines for building MBA SoS. The goal of our guidelines is to support SoS architects with a logical sequence of steps to follow when developing MBA SoS. First, we detail the method describing our guidelines. After that, we show the results of using our guidelines for developing a brand new MBA SoS. Our goal is to validate the generic features of our framework, testing its potential to be applied to different domains.

Chapter 9 - Conclusion and future research

Ultimately, in Chapter 9 we detail our conclusions, highlight the main contributions and the results obtained by this research, and present possible directions for future work.

Chapter 2

Systems of Systems

This chapter presents the main concepts and definitions necessary for understanding Systems of Systems (SoS) in the scope of this research. The goal is to provide an overview of SoS presenting its main concepts, definitions, characteristics and challenges. In this chapter we also present the state of the art for developing SoS related to the scope of this research, comparing the works of the literature and highlighting the main differences and relevance of the approach proposed in this research.

First, we provide a short historical outline of Systems of Systems highlighting the motivation for developing them. Then, we present the main characteristics and the definition of SoS we adopt in this research. After that, we show the main challenges faced when developing SoS related to this research. We finished this chapter by presenting the state of art for developing SoS related to the scope of this research, and then by making a discussion to show the main differences and relevance of the approach proposed by this research.

2.1 Overview

To understand how the concept of Systems of Systems (SoS) has emerged, we first need to look at the elements constituting them, i.e., the *systems*. The contemporary notion of *system* can be traced back to the 1940s when Ludwig von Bertalanffy wrote about General System Theory (GST). In his work, he described some foundations to try to unify the notion of system across domains. The foundations were system-environment, boundary, input, output, process, state, hierarchy, goal-directedness, and information (Bertalanffy, 1969). According to von Bertalanffy, the GST is a discipline concerned with the general properties and laws of systems belonging to any domain. In such a domain-independent context, von Bertalanffy has defined a system as a complex set of components in interaction, or by some similar proposition (Von Bertalanffy, 1967).

As GST advanced, it became a trans-disciplinary study of the abstract organization of phenomena, independent of a given system substance, type, or spatial or temporal scale of existence (Audi, 1999). Thus according to such definitions, systems can range, for instance, from microbiological systems (Tumini et al., 2017) to databases (Eckstein and Schultz, 2018).

The attempt to analyze and understand the life cycle of systems, for instance, to be aware of possible problems or potential solutions, came through Systems Thinking. Such an understanding is characterized by visualizing, conceptualizing, assessing, formulating, developing, synthesizing, evaluating, selecting, optimizing, assimilating, modeling and analyzing a given system (Wasson, 2015). When the understanding concerns specifically systems that are man-made, created and utilized to provide products or services in defined environments for the benefit of users and other stakeholders, the analyses take a perspective of Systems Engineering (SE) (Walden et al., 2015).

Schlager states that the need for SE was first perceived when it came to the mind that satisfactory components do not necessarily combine to create a satisfactory system (Schlager, 1956). The author says that the Bell Telephone Laboratories was probably the first organization to use true systems engineering.

According to the International Council on Systems Engineering (INCOSE),

> Systems engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance, training and support, test, manufacturing, and disposal. Systems engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation. Systems engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs (Walden et al., 2015).

Now, it is important to state that in this research we follow the same line of system as used by the INCOSE, meaning that our notion of system is focused on SE, i.e., on man-made systems. Thus, in this research we adopt the definition of system provided by the standard ISO/IEC/IEEE 15288 and recommended by the INCOSE, meaning that, a *system* is a "combination of interacting elements organized to achieve one or more stated purposes" (ISO/IEC/IEEE 15288, 2015).

Currently, the need for collecting real-time information, integrating business partners to reduce end-to-end delivery times, and capitalizing on globalization, has let organizations

to increasingly rely on interconnected systems to provide more capabilities not found in single (monolithic) systems (Lewis et al., 2011). Such an interconnection forms *Systems of Systems*, consisting of existing or legacy systems, Commercial-Off-the-Shelf (COTS) products, and new systems, providing unique capabilities that are not present in its constituents when working alone, in order to support the needs of an organization.

Although the concept appears to be recent, the notion of Systems of Systems (SoS) is not new. The early ideas come from several different sources. Kenneth Boulding, an economist, uses in his paper on general systems theory the term "system of systems" to describe a "Gestalt," i.e., an organized whole that is perceived as more than the sum of its parts, of theoretical constructs (Boulding, 1956). Ackoff has defined an SoS as an "organization" that is a purposeful system containing at least two purposeful "subset" systems that have a relationship and a common purpose. There is division of labor, the distinct subset systems of the organization can respond to the behavior of others through observation or communication, and the subset systems can express their own will (Ackoff, 1971). In the sense of biological systems, Jacob described an SoS as "every object that biology studies" Jacob (1974). In (Jackson and Keys, 1984), the authors recommended using "systems of systems methodologies" as inter-relationship among different systems-based methodologies in the field of operations research.

However, the notion of SoS from a systems engineering perspective, focused on connecting independent systems together, gained momentum with the United States Strategic Defense Initiative (SDI) from the late 1980s (Nielsen et al., 2015). After that, SoS research has increased in both academic and industry, and since than it has attracted a lot of attention and interest. Nevertheless, it took another 15 years for SoS Engineering (SoSE) to develop as a recognized discipline, and by the early 21st century was still regarded as being in its infancy (Keating et al. (2003); Nielsen et al. (2015)).

Nowadays, SoS has been applied and developed in a number of different domains such as approaches for supporting manufacturing (Ford et al., 2012), tourism to support its domain challenges (Bai, 2013), several approaches for supporting the military activities (Hershey et al. (2013); Olivier et al. (2014); Kilian and Schuck (2016)), software industry to improve the quality of the code (Wanderley et al., 2018), SoS to support the development of particle accelerator facilities (Friedrich et al., 2017), Health Care to improve understanding of medication prescriptions (Wanderley et al. (2017a); Wanderley et al. (2017b)), underground mining to maintain accurate mine maps (Borg et al., 2017), among others.

Despite the large number of SoS being developed, one of the key points that still remains without consensus is about its definition. Currently, there is still a lack of agreement and a divergence concerning a clear definition of an SoS (Maier (1998); Dauby and Upholzer (2011); Stary and Wachholder (2016); Jamshidi (2017)). One of the main confusions is to distinguish between a "monolithic" system and an SoS. For instance, according to

Ackoff (1971), cited above, an SoS can be defined as a set of independent systems with own goals cooperating to achieve a higher purpose. DeLaurentis and Callaway (2004) give an example discussing the possible confusion that can be made: "Following this logic, the combination of a set of different systems forms a larger "system-of-systems" that performs a function not performable by a single system alone. So, if an auxiliary power unit (APU) is a system, then is the airplane a "system-of-systems?" In a narrow sense, perhaps, yes, but our answer would be no, because there is a richer meaning than this sort of simplistic "box-inside-a-box" approach."

This lack of consensus in the SoS definition has led researchers to start basing their definitions on SoS characteristics. In the next section, we present the SoS characteristics and the definition we adopt in this research.

2.2 Characteristics and definition

Because of the divergences in defining SoS, authors have started to base their definitions on SoS characteristics. One of the most used and considered set of inherent characteristics of SoS (Nakagawa et al., 2013) is the one proposed by Maier (1996) in a seminal paper. Maier's characteristics are also cited by the INCOSE (Walden et al., 2015) as useful to define what can be an SoS. According to Maier (1996) SoS have the following main characteristics:

- *Operational Independence*: all the constituent systems of an SoS can often deliver their functionalities when not working with other constituents.
- *Managerial Independence*: each constituent system of an SoS is governed by its own rules rather than by others external to the constituent.
- *Evolutionary Development*: functions and purposes of an SoS can dynamically change and constituent systems can be added or removed to fit them.
- *Emergent Behavior*: an SoS is capable of delivering new functionalities that are the result of the constituent systems working together.
- *Geographic Distribution*: constituent systems of an SoS are geographically distributed, meaning that they can readily exchange only information and not substantial quantities of mass or energy.

In the literature, other characteristics have also been widely used to define SoS, for instance, the *heterogeneity* of constituent systems (Baldwin and Sauser (2009); Jamshidi (2017)). This comes from the fact that SoS lie in a heterogeneous environment, consisting usually of existing, legacy or new systems as described in the previous section. The nature of constituent systems in an SoS can vary in terms of the domain of application,

level of independence, ownership, manageability, among others (Foster et al., 2014). Note that, this does not mean that in an SoS we have only heterogeneous constituent systems. It is also possible to have several instances of the same system within the heterogeneous environment.

Another important characteristic of SoS is that they have *users*. In an SoS, the constituent systems are owned or operated by users (Carlock and Fenton (2001); Nielsen et al. (2015); Nomaguchi and Fujita (2017)). Users of constituent systems can use human-system interfaces to request or receive information from the SoS, for instance, to perform desired tasks and achieve the expected results of the SoS goal. An effective SoS promotes collaborative decision-making and shared situation awareness between the systems users (Saunders et al. (2005); DeLaurentis (2007)).

After presenting the main characteristics of SoS, we now present the definition of Systems of Systems adopted by this research. The definition we use goes through and creates a consistence with the SoS history. We built our definition based on the Ackoff notion of Systems of Systems (Ackoff, 1971), supported by the Maier's SoS characteristics (Maier, 1996), and complemented by heterogeneity and users' characteristics described above in this section of this thesis.

Definition 2.1. Systems of Systems (SoS) are a set of independent, distributed and heterogeneous systems with own goals cooperating to achieve a higher purpose. Among the heterogeneous constituent systems of an SoS, it is possible to have several instances of the same system. Each constituent system can have human users operating them.

Because of the characteristics and particularities shown above, SoS face several different challenges to be developed. In the next section, we discuss some of the challenges faced when developing an SoS.

2.3 Challenges

In this section we present the main challenges faced when developing SoS related to the scope of this research. They are challenges concerning interoperability, robustness, knowledge management, user dimension and evolution.

2.3.1 Interoperability

One of the main challenges in SoS concerns interoperability. In an SoS, constituent systems need to exchange information and cooperate to achieve the expected results of the SoS goal or purpose. That is, no SoS is possible without communication (Maier, 1996).

However, because SoS lie in a heterogeneous environment, interoperability becomes crucial specially to connect the several different and independent constituent systems. *Interoperability* can be defined as the ability of distinct systems to share semantically compatible information and then process and manage such an information in semantically compatible ways, enabling users to perform desired tasks (Zeigler et al. (2008); Madni and Sievers (2014)). According to Madni and Sievers (2014), the interoperability between systems needs to reach syntactic and semantic levels. Syntactic interoperability is the ability of two or more systems to communicate and exchange data, for instance, using standards such as Extensible Markup Language (XML). On the other hand, semantic interoperability stands for the ability of two or more systems to understand the meaning of the information being exchanged. In an SoS, two constituent systems have to agree and share a common vocabulary to reach semantic interoperability. Moreover, the interoperability needs also to support SoS characteristics (DiMario, 2006), for example, when handling with SoS evolution such as adding or replacing constituent systems.

Providing interoperability between constituent systems that do not interoperate natively can demand substantial effort and cost (Madni and Sievers, 2014). Several modifications may be necessary concerning, for instance, external interfaces, standards and communication protocols, or procedures for data handling. When interoperability is not handled correctly in an SoS several issues can occur. For instance, the potential for syntactic and semantic mismatches may be "lethal" and affect the expected results of the SoS goal (Madni and Sievers, 2014). Moreover, limited interoperability between constituent systems can also result in high costs concerning system alignment (Curry, 2012).

To support SoS interoperability, communication protocols need to be used, simplifying and managing the connections among the several heterogeneous systems. The clue is to simplify interoperability without increasing complexity, control, or cost. An appropriate interoperability in SoS needs to reduce complexity and ensure constituent systems do not need to be redesigned when other systems are added, removed, modified, or replaced. The solution can be the use of a common interoperability infrastructure in an SoS (Curry, 2012).

2.3.2 Robustness

The emergent behavior of SoS resulting from the interaction between constituent systems provide enhanced capabilities, but may also introduce new vulnerabilities (Tran et al., 2016). Because of the evolutionary nature of SoS, constituent systems can be added, removed or replaced to fulfill the SoS requirements and achieve the expected results of its goal. When such actions are performed, interdependencies and relations between SoS constituents can be created or destroyed, meaning that changes can occur in an unpredictable manner (DeLaurentis, 2005). Due to such emergent and unpredictable behavior failures can cascade throughout the SoS, creating development delays

or operational system failures (Han et al., 2012). Thus, it is paramount to consider robustness in SoS. In the context of SoS, robustness can be defined as the ability to deliver capability in unknown future conditions (Pape and Dagli, 2013).

Thus, it is important that approaches developing SoS take into account robustness. For instance, providing some support when constituent systems go down, or developing the SoS in a loose coupling fashion to reduce possible interdependencies between systems. Strong coupling can create a chain reaction through constituent systems, leading to catastrophic failures in an SoS. The greater is the coupling between constituent systems, more fragile is the SoS (Johnson and Gheorghe, 2013).

2.3.3 Knowledge management

Performing Knowledge Management (KM) is pivotal as SoS change over time. When cooperating, constituent systems and users interact, exchanging information to achieve the expected results of the SoS goal. However, because systems and users may change during SoS life cycle, crucial knowledge such as one that could be used or reused for supporting the expected results of the SoS goal can be lost. According to Siemieniuch and Sinclair (2014) refinements in an SoS can take place at several intervals, due to technical developments, competitive issues, and changes in the SoS and constituent systems goals. For SoS users, there is a churn over time, probably within 10 years of operation 30% of SoS users will have been replaced, and it is likely that all of them will have been replaced within 50 years. There is an important role for formalizing, keeping and disseminating knowledge within an SoS (Siemieniuch and Sinclair, 2012).

Different issues can occur when KM is not taken into account in an SoS. For instance, constituent systems and users have to redo tasks each time they are required, even if the SoS has not been changed. Moreover, when constituent systems or users are no longer part of the SoS, and paramount information usually provided by them is required, either they need to be reintegrated into the SoS or other means need to be found to recover such an information. The fact is that in both cases reworking, time and effort are required, but could probably be avoided. According to Wickramasinghe and Schaffer (2006) knowledge management involves four main steps (i) creating knowledge; (ii) representing or storing knowledge; (iii) accessing, using or reusing knowledge; and (iv) disseminating or transferring knowledge. Thus, it is a key point to provide means for supporting SoS to capitalize knowledge created during operation and to make it available for being reused later by systems and users when needed.

2.3.4 User dimension

Besides being an important characteristic, the user dimension is also a challenge in SoS. Given the fact that system users can interact with an SoS (Section 2.2), for instance, to

request or receive information, then providing appropriate human-system interfaces to SoS users has become mandatory. Different from monolithic systems which users have predefined interfaces, in an SoS users interact through changing interfaces with other systems or users of the SoS (Madni and Sievers, 2014). Such a dynamic behavior occurs mainly because systems can be added, removed or replaced in an SoS. This means that the sources and the information provided in an SoS can change each time it is modified.

Users in an SoS can experience ongoing changes with respect to the systems and collaborating users. Continuous context changing associated with temporary SoS stakeholders (systems and users) can produce excessive cognitive load that can turn into human errors and increased human error rates (Madni (2010); Madni (2011)). Moreover, because of such temporary nature, lack of trust and scarcity of shared of understanding can affect the expected results of the SoS goal (Madni and Sievers, 2014). While there were advances in to integrate humans with systems (Booher, 2003), existing methods, processes, and tools are not appropriate for integrating humans with SoS (Madni (2010); Madni (2011)).

According to Corsello (2008), users may or may not interact with the SoS directly through interfaces in their constituent systems. The author suggests that the interactions can occur instead through interfaces in external systems connected to the SoS such as a Web server or application. Corsello (2008) argues that SoS often have external systems simply consuming the functionalities provided. Such external systems can be considered as clients of the SoS with their own concerns.

Looking more into the future and hypothesizing about SoS users, Siemieniuch and Sinclair (2014) discussed about how users should receive information in an SoS. According to them, information for decision-making should be given on behalf of individuals, i.e., should take a perspective of each user, providing customized information to them perhaps through AI-based systems. This thought is also shared by other authors, for instance, Zhou et al. (2011) arguing that a critical aspect of complexity is that modern systems are becoming more human-centered since services are becoming more customized.

Thus, there is a need for providing more flexible interfaces to support SoS users, perhaps following an AI fashion and also capable of offering customized support.

2.3.5 Evolution

The evolutionary nature of SoS is a challenge that can be partially responsible for other SoS challenges as we have described in the previous sections with interoperability, robustness, knowledge management and user dimension. Evolution means changing an SoS, for instance, by adding, removing, replacing or modifying its constituent systems. According to Agarwal et al. (2015), some of the needs for evolving an SoS can be for: (i) improving SoS performance; (ii) rendering constituent systems interoperable; (iii)

including additional requirements for the SoS goal; (iv) handling evolution in the constituent systems; and (v) adding new functionalities to the constituent systems.

A usual practice recommended to handle the evolution in SoS is to leave constituent systems loosely coupled one to the other (Ingram et al., 2014). When the coupling between SoS elements is not tight it is easier to change the SoS, for instance, adding or removing systems.

Different works have been proposed to try to support the SoS challenges described in the previous sections. In the next section, we present the state of the art related to the scope of this research.

2.4 Developing Systems of Systems: State of the art

There has been much research for supporting the development of Systems of Systems. Different approaches have been proposed such as modeling the SoS behavior and organization to analyze its dynamics (Kumar et al., 2017), simulating an SoS to find effective configuration between constituent systems (Gomez et al., 2015), and methodologies for supporting the process of developing SoS (Lana et al., 2016).

The purpose of this section is to present works related to the scope of this research. Some works explicitly state that they provide architectural support for developing SoS, take into account the user dimension, and consider robustness in SoS.

Section 2.4.1 first presents related work providing architectural support for developing SoS. Then, Section 2.4.2 presents literature that encompasses the user dimension in SoS, and Section 2.4.3 works that consider robustness in SoS. After that, Section 2.4.4 summarizes and discusses our state of the art.

2.4.1 Architectural support for developing for SoS

The work of Varga et al. (2017) implemented a framework for developing and deploying an SoS. The elements of the framework were systems that provide and consume services, and cooperate as an SoS. A system could be the service provider and at the same time a consumer asking for services. The framework was based on a Service-oriented architecture (SOA). The architecture relied on core systems to enable the exchange of information. One of the core systems that played a major role in the approach was the "Service Registry." All systems in the SoS must have to register their services within the Service Registry in order to make them available to consumers. The interoperability between constituent systems is realized through several different protocols, in the case they use natively different technologies. Then, the approach suggested three ways to

realize the interoperability, all of them being based on a central element. Generic examples were given, for instance, by using a central layer shared by all the systems to translate their protocols, or by using translator systems acting as a middleware between consumers and providers. The use of standards was also recommended. According to the authors, semantics related to the communication could be described in a documentation, named "Semantic Profile," which appears to be kept separately from the rest of the constituent systems. The approach was applied to different domains such as home and industrial automation, and virtual markets.

An architecture for Large-Scale Smart Grid SoS was presented by Pérez et al. (2015). The proposed architecture was multi-tier with six layers, based on event-driven service-oriented architecture. Among the layers there were a "Physical Layer," a "Communication Layer," a "Service Layer," and a "Data Layer." The "Physical Layer" was the lower layer in the architecture, being responsible for receiving from and sending information to a given system in the raw format understandable by it. Several different communication protocols were used in this layer. To let systems interoperate the authors recommended the use of different standards. The "Communication Layer" intended to translate the communication protocols that coexisted in the "Physical Layer" into a common technology of communication. This layer provides the communication middleware of the SoS, and the authors recommended to use standards for it also for handling semantics. The "Service Layer" intended to store and make available the services provided by the constituent systems. The authors have used the "Data Layer" to store and retrieve data from the smart grid systems. The approach was specially developed for Large-Scale Smart Grids such as solar farms and power plants.

Ingram et al. (2015) proposed to use architectural patterns for developing an SoS. The goal was to make the patterns as reusable solutions to abstract architectural problems. The authors presented nine patterns, based on experience, to be applied in an SoS, for instance, "Centralized," "Blackboard," "Pipe and Filter" and "Reconfiguration Control" architectural patterns. When describing the patterns, possible structures and rationales were showed. According to the authors, the work is in early stages and the set of patterns is far from complete. Moreover, the approach was not reported to have been applied to case studies, nor tested in practice.

The work of Biran et al. (2017) aimed to allow multiple cloud providers to optimally utilize compute resources. The work proposed a cloud federation system of systems architecture, creating an SoS model of cloud service providers coordinated by a Cloud Federation control plane. The architecture was composed mainly of two centralized elements, named "Clouds-Coordinator" and "Clouds-Broker." The Clouds-Coordinator intended to act as an information registry that stores offers and demand patterns, and the Clouds-Broker to manage the membership of systems, to handle resource allocation and provisioning requests.

Exploring the impact of big data dimensionalities on a Global Earth Observation (GEO) System of Systems was the goal of Nativi et al. (2015). The SoS they developed was based on a broker architecture. The broker was an intermediary separating consumers of services from providers of services. It was also responsible for providing semantic interoperability by translating the several different protocols used by systems. The semantics were handled by using an internal model and consulting semantic assets like taxonomies. In (Nativi et al., 2013), the authors have detailed the construction of the broker used in the architecture. One may see that the approach has used a single broker, named "GEO Discovery and Access Broker" or simply, "GEO DAB." The broker appears to be a central core in the SoS. Besides, another characteristic of the approach was that it could store knowledge such as caching strategy to dataset preview and fast loading.

Wong et al. (2014) presented a system of systems service architecture for social media analytics and data sharing. The architecture consisted of systems such as databases, social data services, and geospatial data services. The interoperability for the cooperation between systems was realized through several different standards, mainly for the Web. There was a centralized transformation between messages from different protocols. A centralized service catalogue was responsible for storing and make available the services provided by the constituent systems. According to the authors, the approach also provided support for storing knowledge from the systems and letting access to it through Web query processing.

The work of Moschoglou et al. (2012) intended to develop a federated System of Systems for creating ubiquitous systems based on service-oriented principles. The federated SoS architecture was composed of different decentralized federations of constituent systems. It was based on a semantic mediation framework for orchestrating activities of publishing system capabilities during the design stage of the SoS, as well as automating capability discovery, selection, and composition at runtime. The semantics and interoperability in the architecture were handled through ontologies. Each constituent system had an ontology containing, for instance, terms and definitions about it. Moreover, a "foundational ontology" was used to create a common terminology between the domains of the ubiquitous systems. All the ontologies were published in a centralized component of the federated architecture, named "Web Service Execution Environment." This centralized element was a mediator responsible for performing a semantic discovery, selection and orchestration of processes when receiving requests from systems.

An approach to Systems of Systems information interoperability aiming to share information on the Web was proposed by Curry (2012). The SoS architecture relied on a single and centralized abstraction, named "linked dataspace," playing the role of a medium for the semantic interoperability between constituent systems. To provide the interoperability the linked dataspace has made use of Semantic Web standards like RDF[1], linking information by entity-centric approach focused on the concepts that exist within the

[1]See `https://www.w3.org/RDF/` for more information.

systems, for instance, business entities like employees, products, and customers. The connection between constituent systems and the linked dataspace was made through wrappers responsible for translating the raw data format used by systems through a "RDFization" process. The authors applied the approach in a case study of enterprise energy management.

Defining an evolutionary systems of systems architecture was the goal of Selberg and Austin (2008). Although, the goal was to define an architecture, the work still appears to be initial proposals. It has mainly focused on providing directions and recommendations to what would be a good architecture for handling the SoS evolutionary nature. According to the authors, a good evolving SoS architecture would consist of standard interfaces, interface layers, and a continual system verification and validation process. To make constituent systems interoperable the authors suggested the used of standard interfaces. Although, it is not explicitly stated, one could infer according to the figures provided in the work, that the interface layer suggested would be a single common abstraction used by all the constituent systems.

Several other works intended to provide architectural support for developing SoS. The work of Oquendo (2016) proposed an architecture description language to formally describe the architecture of an SoS. The language focus more on the early stages of designing an SoS, and it represents SoS architectures in abstract terms, for instance, "systems" are elements defined by intention and with a behavior and "mediators" are elements for enabling the interaction between systems. Other works aimed to provide decision-making to SoS architects. In (Agarwal et al., 2015), the authors aimed to provide decision-making for evolving SoS architectures. They tried to optimize SoS architectures in the face of dynamic requirements and uncertainty in constituent system capabilities and participation. Parameters such as cost and time in an SoS were used as examples for showing the optimization realized.

Besides, some works in the literature have used well-known architecture frameworks, originally developed to the military domain, like DoDAF (DoDAF, 2004) and MODAF (MODAF, 2005) to support their approaches. Such frameworks have been developed to reduce the SoS modeling complexity by using multiple layers called architecture views that represent and describe the system from different perspectives (e.g., operational, functional, and system) (Sanduka and Obermaisser, 2014). An example of work using the DoDAF framework can be of Ge et al. (2014). The authors proposed an executable modeling approach for SoS architecture to create a common and integrated specification of architectural data modeling. The approach provided more flexibility and adaptability to the automated construction of executable models directly from architectural data.

2.4.2 User dimension in SoS

In this section, we present related work that take into account the user dimension in SoS.

Jokinen et al. (2016) intended to provide reconfigurable interfaces for users of SoS consisting of production systems. The interfaces took into account user privileges and data permissions to allow users to monitor and receive data from the systems, presented in dashboards and charts. Despite of the graphics provided, the interfaces were based on the "WIMP" (Windows, icons, menus, pointer) paradigm for the interactions with users. The approach relied on a central component, to distribute and provide information to the interfaces.

Some researchers have exploited the use of Web portals to let users obtain information from the SoS. Nativi et al. (2015) have designed a Web portal for retrieving information from an SoS involving systems for Earth observation, monitoring and information. The goal was to provide visualization for presenting big data to support decision makers to understand and use them. In the Web portal, the visualization of the information adopted a strategy involving filters and zooming. Mazzetti et al. (2017) proposed an SoS for a Virtual Research Environment (VRE), mostly focused on monitoring and retrieving information concerning volcanoes. The approach provides interoperability (through a broker) with the identified heterogeneous systems supplying data and information for the Web portal. The portal designed in this work lets users publish, discover, and access datasets, according to their profile. The Web portals provided by the approaches allowed queries in natural language to search information. After that, the navigation was made, for instance through buttons, pop-ups, and lists.

The work of Vierhauser et al. (2016) intended to provide a flexible framework for developing monitoring solutions for an SoS. The approach was based on an event model for managing and analyzing arbitrary events occurring at runtime in an SoS. To analyze such events, the framework provided interfaces allowing tools to access events and event data, for instance, for visualizing the SoS behavior. Among the provided functionalities, the users could group and filter events or timestamps, and custom-made visualizations. The interaction between users and the interfaces was based on the WIMP paradigm.

Varga et al. (2017) stated that their framework to develop SoS had a "Dashboard MMI" (Man–Machine Interface), in order to allow interaction with the SoS developed. Despite of that, no more details are provided about the dashboard. Thus, some questions stay unanswered, for instance, It it implemented? If not, What are the requirements? How to do that?

The linked dataspace of Curry (2012) for SoS interoperability described in Section 2.4.1 allowed users to interact with it using structured or natural language. The authors detailed the approach in Freitas et al. (2013). The natural language query mechanism was

based on the combination of entity search, a Wikipedia-based semantic relatedness measure and spreading activation. The interaction between users and the linked dataspace followed a query-answer fashion.

2.4.3 Robustness in SoS

In this section, we present related work that consider robustness in SoS.

The work of Pape and Dagli (2013) proposed an approach to assess and analyze the robustness in SoS meta-architectures. The goal was to analyze robustness by removing constituent systems and measuring the remaining performance of the SoS. The meta-architectures with smaller losses of performance were considered more robust. The approach was based on Genetic Algorithms.

Adapting social network analysis tools to analyze and identify potential vulnerabilities in SoS was the goal of Enos et al. (2017). According to the authors, SoS architects might be able to improve the robustness of an SoS by using the analysis performed with their approach. The analysis tools proposed were based on network centrality metrics. The authors applied the approach in a case study of large-scale power outage.

The work of Biran et al. (2017), presented in the previous section, aimed to allow multiple cloud providers to optimally utilize compute resources. Concerning robustness, the authors mentioned the desire for it by doing a continuous integration process in the code used to implement their approach from the moment it is ready for test until it processes production workloads. Besides, they would also plan to achieve robustness by following best practices of software development.

Nomaguchi and Fujita (2017) developed a framework of SoS design to handle uncertain and emergent behavior. The proposed framework adopted a scenario design method, a multi-agent simulation method, and a robust design method. The framework was capable of evaluating robustness based on a lattice point approach. According to Nomaguchi and Fujita (2017), through their framework an SoS architect could evaluate the uncertainty in different scenarios and choose optimal solutions in terms of objective characteristics and robustness. A case study of designing subsidy polices for a distributed generation system was used to apply the proposed approach.

Analyzing the impact of cyberattacks on an SoS was to goal of Guariniello and DeLaurentis (2014). The authors presented a functional dependency analysis tool that modeled the effects of constituent system disruptions, and the effects of cyberattacks on communication links between systems. The tool was able to evaluate robustness in terms of the capability of an SoS to maintain an adequate level of operability following a disruption in communications. According to the authors, the method could be used to

guide decision both in architecting the SoS and in planning updates and modifications, accounting for the impact on the SoS robustness.

The work of Ferrario et al. (2016) proposed a Hierarchical Graph to analyze the robustness in SoS. The robustness of the SoS was evaluated in terms of the product delivered to the demand nodes of the Hierarchical Graph. The quantitative evaluation of the SoS robustness was performed through Monte Carlo simulation. The approach was applied in case studies of gas and electricity, and power distribution domains.

Davendralingam and DeLaurentis (2013) presented a robust optimization framework for architecting an SoS. In the framework constituent systems were modeled as nodes on a network working to fulfill capability objectives. The SoS robust network model was posed as a mathematical programming problem, intending to maximize the expected network performance in fulfilling key objectives whilst satisfying connectivity requirements between constituent systems. The approach was applied to a Littoral Combat Ship platform.

The work of Ingram et al. (2015), presented in the previous section, proposed to use architectural patterns for developing an SoS. According to the authors, some of the patterns presented could make an SoS more robust through redundancy, backup or replacement.

2.4.4 Discussion of the state of the art

In this section we summarize and discuss the related work presented in the previous sections. The goal is to show the main differences between such works and the approach proposed by this research. First, we present a summary of the related work. Then, after that we discuss the related work showing the main differences to our approach.

To facilitate the visualization and comprehension, we summarized the related work using the Table 2.1. The table shows the works according to different criteria such as goals, the interoperability provided by the approaches, domain covered, how they take into account the user dimension, if the approach provides means for Knowledge Management (KM) during SoS operation, if they consider robustness, and if they rely on centralized components. In the table, the works are organized according to the topics described in the previous sections, i.e., architectural support, user dimension and robustness.

Table 2.1: A summary of the related work.

Begin of Table								
Topic	Work	Goal	Centralized Elements?	Interoperability	User dimension	KM?	Consider Robustness?	Domain
Architectural support/ User dimension	(Varga et al., 2017)	Framework for developing SoS	Yes	Several protocols; centralized (layer); semantics (separated doc)	Dashboards (without details)	No	No	Independent
Architectural support	(Pérez et al., 2015)	Architecture for large-scale smart grid SoS	Yes	Centralized (layer); semantics (standards)	Not mentioned	Yes	No	large-scale smart grid
Architectural support-/Robustness	(Ingram et al., 2015)	Architectural patterns for developing SoS	Depends on the pattern	Not mentioned	Not mentioned	No	Depends on the pattern	Independent
Architectural support-/Robustness	(Biran et al., 2017)	Cloud federation SoS to allow multiple cloud providers	Yes	Not mentioned	No	No	Best practices on the implementation	Independent
Architectural support/ User dimension	(Nativi et al., 2015)	Impact of big data on an Earth observation SoS	Yes	Several protocols; Centralized (broker); semantics (own model and taxonomies)	Web portals (point-and-click interfaces)	Yes	No	Earth observation
Architectural support/ User dimension	(Wong et al., 2014)	SoS service architecture for social media analytics	Yes	Several protocols; Centralized (message transformation); Standards	Browser interface (without details)	Yes	No	Social media

Continuation of Table 2.1

Topic	Work	Goal	Centralized Elements?	Interoperability	User dimension	KM?	Consider Robustness?	Domain
Architectural support	(Moschoglou et al., 2012)	Federated SoS for ubiquitous systems	Yes	Decentralized; semantics (ontologies)	No	No	No	Independent
Architectural support/ User dimension	(Curry, 2012)	SoS providing interoperability to share information on the Web	Yes	Decentralized (wrappers); semantics (Semantic Web)	Natural language queries	No	No	Independent
Architectural support	(Selberg and Austin, 2008)	Evolutionary SoS architecture	Yes	Centralized (layer); Standard interfaces	No	No	No	Independent
Architectural support	(Oquendo, 2016)	SoS architecture description language (conceptual level)	-	No	No	No	No	Independent
Architectural support	(Agarwal et al., 2015)	Decision-making for evolving SoS architectures (optimization)	-	No	No	No	No	Independent
Architectural support	(Ge et al., 2014)	Executable modeling approach for SoS architecture (modeling; DoDAF framework)	-	No	No	No	No	Independent
User dimension	(Jokinen et al., 2016)	Reconfigurable interfaces for users of SoS	-	No	User privileges to monitor/receive data in dashboards; "WIMP;" central component	No	No	Production systems

Continuation of Table 2.1

Topic	Work	Goal	Centralized Elements?	Interoperability	User dimension	KM?	Consider Robustness?	Domain
User dimension	(Mazzetti et al., 2017)	SoS for a Virtual Research Environment (VRE)	-	Yes (Broker)	Web portal; queries; "WIMP"	No	No	Geohazard (Volcanoes)
User dimension	(Vierhauser et al., 2016)	Framework for monitoring solutions for an SoS	-	No	Interfaces allowing tools to access event data; group/filter events; "WIMP"	No	No	Independent
Robustness	(Pape and Dagli, 2013)	Assess the robustness in SoS meta-architectures	-	No	No	No	Genetic Algorithms; assess robustness (performance)	Independent
Robustness	(Enos et al., 2017)	Analyze SoS robustness through social network analysis tools	-	No	No	No	Analyzing robustness (network centrality metrics)	Power networks
Robustness	(Nomaguchi and Fujita, 2017)	Framework of SoS design to handle uncertain and emergent behavior	-	No	No	No	Evaluating robustness (lattice point)	Independent
Robustness	(Guariniello and DeLaurentis, 2014)	Analyzing the impact of cyberattacks on an SoS	-	No	No	No	Evaluating robustness (SoS capability to maintain operability)	Independent

Continuation of Table 2.1

Topic	Work	Goal	Centralized Elements?	Interoperability	User dimension	KM?	Consider Robustness?	Domain
Robustness	(Ferrario et al., 2016)	Hierarchical Graph to analyze SoS robustness	-	No	No	No	Evaluating robustness (Monte Carlo simulation)	Independent
Robustness	(Davendralingam and DeLaurentis, 2013)	Optimization framework for architecting an SoS	-	No	No	No	Optimization (SoS was a robust network model)	Independent

End of Table

After presenting a summary of the related work, we now analyze and discuss them based on the criteria used in Table 2.1.

Concerning the architectural support for developing SoS, the most part of the approaches rely on centralized components, for instance, to store and provide the services or functionalities used by systems in the SoS (Moschoglou et al. (2012); Wong et al. (2014); Nativi et al. (2015); Pérez et al. (2015); Biran et al. (2017); Varga et al. (2017)), or to render constituent systems interoperable like handling with semantics and translations (Selberg and Austin (2008); Moschoglou et al. (2012); Wong et al. (2014); Nativi et al. (2015); Pérez et al. (2015); Varga et al. (2017)). However, the use of centralized components makes the architectures for SoS more brittle. First, it creates single points of failure which is even more risky in the case of SoS because of emergent and unpredictable behavior. For instance, when the central element goes down a complete failure can occur in the SoS as functionalities cannot be anymore requested or provided, or the communication between systems fails. Moreover, the data describing the systems and functionalities they provide can be lost during a failure, thus rework and more efforts are required to put the SoS again in operation. Furthermore, the central component can be overloaded with requests, thus the SoS can suffer from bottlenecks which can affect the expected results of its goal. In addition, the use of central elements increases the coupling of constituent systems of an SoS. For example, data and references such as functionalities, IP or ports are kept between the central components and the systems. This is not appropriate in the case of SoS as they evolve and change over time, and thus more effort and rework is required to add, update or remove such references consistently. Moreover, as we stated in Section 2.3.2 the greater is the coupling between constituent systems, more fragile is the SoS.

Regarding the interoperability, the most part of the works use or recommend the use of standards. This is mainly because of the heterogeneous environment of SoS. Some works such as (Wong et al. (2014); (Nativi et al., 2015); Varga et al. (2017)) use several different protocols or standards at the same time to provide the interoperability in the entire SoS. In this case, the translation of the messages is done within central components. This increases the complexity, effort and cost in an SoS (Section 2.3.1), for instance, each time a system using a different protocol is connected to the SoS, new translations need to be created in the central element. In some cases, in order to implement the changes the central element needs to be stopped and so the entire SoS.

Unfortunately, in the literature only few works provide means for knowledge management during SoS operations (Wong et al. (2014); Nativi et al. (2015); Pérez et al. (2015)). Moreover, these are also the works that have proposed approaches designed specifically for a given domain. Their focuses are in large-scale smart grid, Earth observation, and social media domains.

There are also other works that take a more conceptual view when providing architectural support for SoS. There are works proposing languages to formally describe the architecture of an SoS (Oquendo, 2016), or techniques to optimize and provide decision-making about SoS architectures (Agarwal et al., 2015). Other works tried to study architectural patterns for supporting the development of SoS Ingram et al. (2015). Whereas, there are authors like (Ge et al., 2014) using well-known architecture frameworks such as DoDAF (DoDAF, 2004), originally developed for the military domain, to support their approaches. However, such frameworks represent the architecture statically and focus too much on what should be described rather than on the concrete (Hu et al., 2014). All of these works focus mainly on a conceptual level, i.e., describing and documenting the architectures, rather than a practical point of view for supporting the development of SoS.

When the topic concerns user dimension, the most part of the works ((Nativi et al., 2015); Jokinen et al. (2016); Vierhauser et al. (2016); Mazzetti et al. (2017)) try to support SoS users with interfaces consisting mainly of buttons, lists, and pop-ups, or that were designed with the WIMP paradigm (Windows, icons, menus, pointer). These are rigid interfaces and can be difficult to be used in the case of SoS as information are provided to and come from several different kinds of systems. Moreover, the use of rigid interfaces cannot be appropriate for SoS as they change over time (Section 2.3.4). Furthermore, the number of components in such interfaces can increase and so the complexity for SoS users. In another work (Curry, 2012) the authors provided some more flexible interfaces with interaction being done through queries in natural language. However, the approach followed a query-answer fashion and did not go further, being the support to users limited and passive (user-dependent), without providing more help or interaction like the one that could be done during a dialogue with an assistant. Besides, there are other works in the literature (Wong et al. (2014); Varga et al. (2017)) that state they provide some interface to SoS users, but no further details are provided, for instance, what such interfaces can do or what are the requirements to implement them.

The works considering robustness in SoS try more to analyze, optimize or evaluate it by using mainly models and networks (Davendralingam and DeLaurentis (2013); Pape and Dagli (2013); Guariniello and DeLaurentis (2014); Ferrario et al. (2016); Enos et al. (2017); Nomaguchi and Fujita (2017)). Some other works are more conceptual, for instance, Ingram et al. (2015) argued that some architectural patterns could increase robustness. Besides, there are also other authors that state or show explicitly their intentions regarding SoS robustness, for example, in (Biran et al., 2017) the authors have demonstrated their desire to improve robustness by developing code with the best practices. Although all of these works are important for supporting SoS robustness, they are more concerned with design stages and focus mainly on providing information for decision-making. We did not find an approach that tries to support SoS robustness from a practical point of view, i.e., directly during SoS operation. For instance, none of

the approaches try to let an SoS coherent after its constituent systems go down. This can be important as once systems are reconnected to the SoS their state may not be anymore consistent with the rest of the SoS, and thus information provided by them can be incoherent and spread over all the SoS.

The main differences between the works of the literature to the approach proposed by this research is that we propose a framework which is based on a domain-independent peer-to-peer (P2P) architecture with loose coupling between its elements, focusing on the development of SoS from a practical point of view. Our approach does not rely on centralized components or keep references between each of its elements. Moreover, thanks to the loose coupling provided by our architecture, the SoS developed with it can be easily adapted or changed between different domains. Furthermore, our approach provides means for knowledge capitalization and management during SoS operation, and uses a single communication protocol providing syntactic and semantic interoperability between constituent systems. Moreover, it takes into account the user dimension by providing proactive interfaces capable of interacting with SoS users through dialogues with Personal Assistants, and offering proactive support when they are using the SoS. In addition, our approach considers robustness during SoS operation, trying to support it keeping coherent when constituent systems go down.

2.5 Summary

In this chapter we have provided fundamental concepts and background related to this research. First, we have presented an overview showing a brief history of Systems of Systems (SoS), starting with their foundations in the 1940s until the notion of SoS used nowadays in the literature. During the overview, we have also shown the definition of *system* (monolithic) adopted by our research. After that, we have shown that currently there are divergences in the literature for providing a single definition for SoS, and then presented their characteristics explaining that they have been used as a complement for improving the definition of SoS. Then, we have presented the definition of SoS adopted by this research. Next, we have shown the main challenges for developing SoS related to the scope of this research such as *interoperability*, *robustness*, *knowledge management*, *user dimension*, and *evolution*. After that, we have gone deep in the literature and we have shown the state of the art related to this research. We have presented works concerning *architectural support for developing for SoS*, *user dimension*, and *robustness*. We have finished this chapter by discussing the state of the art and highlighting the main differences to our approach.

Chapter 3

Multi-agent systems

This chapter presents the main concepts and definitions of Multi-agent systems (MAS) related to the scope of this research. Our goal is to give enough information for understanding why MAS are a key element in the approach proposed in this research (details in Chapter 4).

We begin the chapter by stating the fundamental concepts of MAS concerning the scope of this research. After that, we discuss the main arguments in favor of our choice of MAS for supporting SoS.

3.1 Characteristics

In this section we present concepts and definitions of MAS regarding the scope of this research. First, we present foundations for the concept of software agent, then after that present the basis for multi-agent systems.

3.1.1 Agent

In this research the notion of agent comes from the concept of *software agent* in Artificial Intelligence (AI). For matters of simplification and readability, in this research we refer to software agents simply as *agents*.

According to Jennings et al. (1998) an agent can be defined as a "computer system, situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives." By autonomy the authors mean that the agent should be able to act without direct intervention of humans or other agents, and should have control over its own actions and internal state.

In a more precise definition citing more characteristics, Ferber (1999) specifies that an agent is an entity that:

- is capable of acting in an environment,
- can communicate directly with other agents,
- is driven by a set of tendencies (in the form of individual objectives or of satisfaction/survival function which tries to optimize),
- possesses resources of its own,
- is capable of perceiving its environment (but to a limited extent),
- has only a partial representation of this environment (and perhaps none at all),
- possesses skills and can offer services,
- may be able to reproduce itself,
- whose behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representations and communications.

In this research we see the definitions above as complementary for describing an agent. They share common aspects though different perspectives, for instance, the agent environment, actions performed by the agent in such an environment, and the agent goals. Moreover, although it is not directly explicitly the definition of Ferber (1999) also concerns agent autonomy. According to the author, an agent is "its own engine," i.e., it is endowed with autonomy, meaning that they are not driven by commands from users or other agents, but by a set of tendencies.

When interacting with their environment, agents need to decide which actions must be performed. During the decision phase, they reason for choosing the actions needed. Concerning the degree of reasoning, agents can be classified in two extremes, named *reactive* and *cognitive* (Shen et al., 2003).

Reactive agents use purely reactive mechanisms to take their decisions. They do not keep explicitly representations of their environments, reacting only to external stimuli and do not communicating directly. An example using reactive agents can be find in (Brooks, 1990). In particular brooks developed mobile robots interconnected and using sensors, being capable of reacting to external conditions directly driving actuators.

On the other hand, cognitive agents have an explicitly representation of their environments, for instance, by using ontologies. Such a representation may contain knowledge, for example, related to the domain, goals, plans or other agents. Cognitive agents interact and cooperate with other agents. Note that, in some cases the use of reactive

approach may be not enough for the application being developed, or the cognitive one may result in a too complex approach, then a mix of them can used, creating thus *hybrid agents* (Shen et al., 2003).

A specific example of cognitive agent is the *Personal Assistant* agent (PA). As envisioned by Negroponte (1996), a PA intends to be a "Digital Butler" of a human being. The goal of the PA is to provide assistance to its user, also called master, with her daily tasks (Ramos, 2000). A PA keeps representations and knowledge, for instance, related to its user's tasks and environment, using then it for helping her. When supporting its user, the PA aims to find, engage and coordinate agents capable of doing the tasks (Ramos, 2000). Such agents engaged to perform the tasks compose the PA *staff* (Enembreck, 2003). Note that the interaction between user, PA and staff focus more on a sense of delegation (Negroponte, 1996). The following extract from Wooldridge and Jennings (1995) illustrates what would be the behavior of a PA according to the authors' mind:

Example 3.1. *Upon logging in to your computer, you are presented with a list of email messages, sorted into order of importance by your PA. You are then presented with a similar list of news articles; the assistant draws your attention to one particular article, which describes hitherto unknown work that is very close to your own. After an electronic discussion with a number of other PAs, your PA has already obtained a relevant technical report for you from an FTP (File Transfer Protocol) site, in the anticipation that it will be of interest.*

Different kinds of interfaces have been proposed for supporting the interaction between PAs and users. Recently, there has been a booming interest in using natural language interfaces between PAs and users, coming from both academia and industry. For instance, in academia researches have been proposing interfaces through dialogues (Paraiso and Barthès (2006); Fuckner et al. (2014)), and multimodal interaction (Jones et al., 2012) mainly by vocal means and interactive tables. Coming from industry, we have noticed the popularization of applications like Google Assistant[1] from Google, Siri[2] from Apple, and Cortana[3] from Microsoft.

3.1.2 Multi-agent systems

In practice "single agent systems" are rare (Bordini et al., 2007). The key idea behind Multi-agent systems (MAS) is that a system could get better results if agents worked in cooperation, instead of in full isolation.

According to Uhrmacher and Weyns (2009), a multi-agent system is a system composed of micro level entities, named agents, which have autonomous and proactive behavior and

[1]See `https://assistant.google.com/` for more information.
[2]See `https://www.apple.com/ios/siri/` for more information.
[3]See `https://www.microsoft.com/en-us/windows/cortana` for more information.

interact through an environment, thus producing the overall system behavior observed at the macro level. Note that, an agent belonging to an MAS is not capable of working in complete isolation or alone, i.e., the agent depends on cooperating with the other agents to perform its tasks. For Sycara (1998) an MAS has four main characteristics:

- Each agent is part of the overall solution. It does not have enough information to solve the whole problem.
- The control is not centralized.
- Information used for problem-solving is decentralized, i.e., agents are distributed.
- The problem-solving mechanism is asynchronous.

MAS offer naturally two powerful traits for handling complexity, named modularity and abstraction (Sycara, 1998). When the problem to be addressed is complex and large, the easiest way to solve it is breaking it in small pieces and distributing them to different agents. Thus, usually each agent in an MAS does not hold a high degree of complexity as the problem is decomposed among them to alleviate it. When some kind of abstraction is needed, for instance, a loose-coupling between heterogeneous elements, then agents can provide such an abstraction acting as a middleware.

Multi-agent systems have been developed through a number of different architectures (Shen et al., 2003). Such architectures aim at providing a structure and a means for the agents of an MAS to interact and cooperate. Following, we show some typical architectures that have been largely employed in MAS.

Hierarchical multi-agent architectures are based on a hierarchy of layers or levels. Broadly speaking, such architectures follow a "tree" fashion. The elements belonging to higher levels exert more control under the elements belonging to the lower ones. There is also a centralization in the hierarchy as lower levels depend on the upper ones. To illustrate a multi-agent hierarchical architecture we give an example in the context of manufacturing systems. As shown in Figure 3.1, the highest level corresponds to a manufacturing enterprise composed of different resources, for instance, databases with monitoring, decision-making and control mechanisms. Subsystem managers like shop floor managers are subordinated to the manufacturing enterprise level. Each subsystem manager coordinates a group of resource agents playing different roles such as manufacturing devices. The main drawbacks of hierarchical architectures are that they rely on centralized elements and create a tight coupling between elements. Failures in upper levels can trigger chain reactions affecting the lower ones. Moreover, data is shared globally making the architecture more brittle.

Another kind of MAS architecture is the facilitator proposed by McGuire et al. (1993). In this architecture, agents are combined into a group (see Figure 3.2) and the communication between them and other groups is realized through an interface called facilitator.

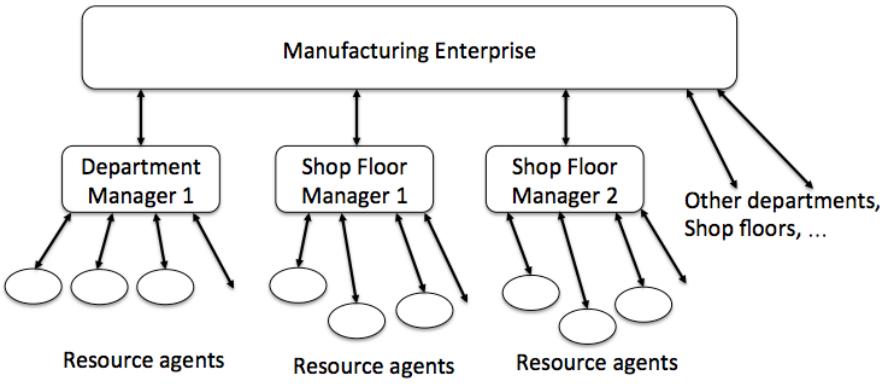


Figure 3.1: A hierarchical multi-agent architecture for manufacturing systems (adapted from (Shen et al., 2003)).

The facilitator is responsible for translating and routing messages from/to the agents of its group to remote groups. One of the advantages of such an approach is that the complexity of handling messages is made by the facilitators, leaving the agents of their groups focused directly on their tasks. On the other hand, some of the disadvantages are the tight coupling between intra-groups (agents/facilitator) and inter-groups (facilitator/facilitator) for the communication, agents within a group rely on a centralized element (the facilitator), and the complexity of a facilitator increases as the number of agents augment in its group.

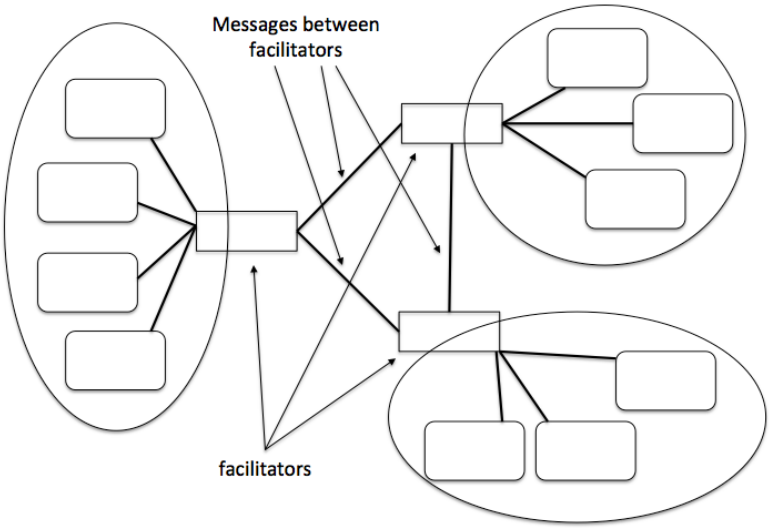


Figure 3.2: A multi-agent architecture using facilitators (adapted from (Shen et al., 2003)).

The broker architecture is based on a brokering mechanism consisting of broker agents. Brokers are "intermediaries" that receive requests from agents, trying to find potential receptors for such requests. Then, once results for such requests are available, receptors send them to brokers that in turn transfer them back to the requester agents. Figure 3.3 shows a MAS architecture based on brokers. Comparing to the facilitator approach which each facilitator is responsible for a selected group of agents, the broker architecture

allows all agents in the system contact all brokers in the same system for finding required services. Although the broker approach introduces new intermediary elements, it is decentralized and provides loose coupling between agents as they do not need to rely on a single broker.

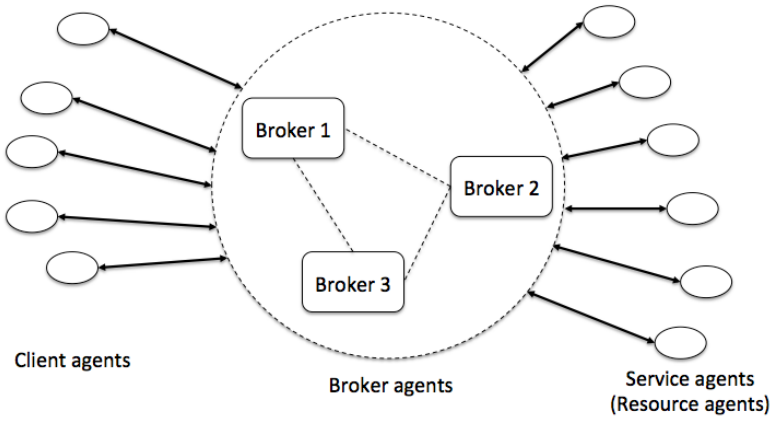


Figure 3.3: A multi-agent architecture using broker agents (adapted from (Shen et al., 2003)).

The matchmaker approach uses the brokering mechanism to match agents, but adding a new step of connection between them. Once service agents or providers are found, the matchmaker agent establishes links between requesters and providers, stepping out of scene and letting them to communicate directly with each other (see Figure 3.4). One of the advantages of the matchmaker approach is that it is good for searching and information retrieval. However, it is a more complex approach as it performs more steps after finding providers. Moreover, it creates a strong coupling by linking agents directly with each other.

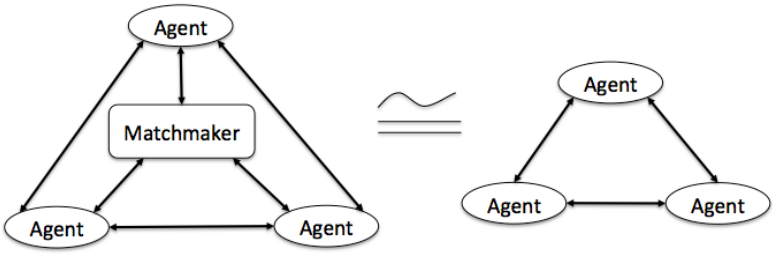


Figure 3.4: A multi-agent architecture based on matchmaking mechanism (adapted from (Shen et al., 2003)).

Blackboards have also been used as MAS architectures. For instance, an agent-based blackboard approach for managing agent interactions was proposed in Lander et al. (1996). In such an approach (see Figure 3.5), each group of agents shares a local data repository (a blackboard) which is used for storing and retrieving shared data between the group of agents more efficiently. Each group of agent has a control shell responsible for notifying agents about relevant events. Although the approach may increase efficiency, all the shared data within an agent group is centralized in a blackboard.

Moreover, there is a dependency between agents and the control shell (which is also centralized) to receive updates of changes made in the blackboard.

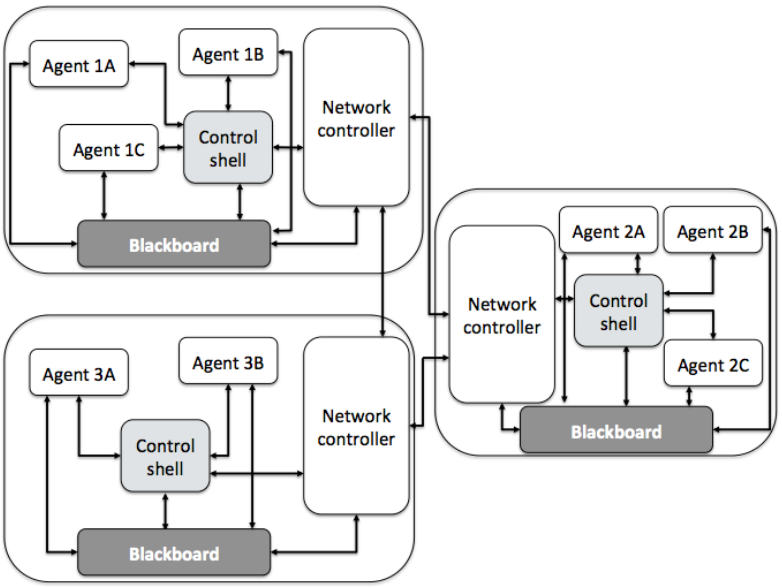


Figure 3.5: An example of multi-blackboard architecture (adapted from (Shen et al., 2003)).

As we stated in the beginning of this chapter, MAS are a key element in the approach proposed by this research for supporting the development of SoS. For being so, it is important that they lie in some structure such as one of those just described above. In our approach, we have chosen for the MAS element a structure based on the **broker approach**. The main reasons were that the broker approach is decentralized and provided loose coupling between its elements, thus we believe that it can fit especially well for SoS because of their distributed and evolutionary nature (details in Chapter 2).

Next we discuss the reasons that have led us to use MAS in the approach proposed in this research.

3.2 Discussion: Why use MAS for supporting SoS?

Because MAS are a key element in our architecture (details in Chapter 4) it is important to understand why we have chosen them to be part of our approach, and how they can be useful for SoS. In this section we make a discussion and provide arguments for supporting our choice. Our arguments are mainly based on the assumption that MAS and SoS share common characteristics.

The first reason that has guided us towards employing MAS in our approach was its *cooperative nature* which is also a trait in SoS. In an MAS, agents have their own goals performing part of the overall solution, and cooperating with each other to achieve the

global goal. In the case of SoS, the constituent systems have their own purposes and cooperate together to accomplish the SoS higher purpose.

Second, MAS and SoS have both a characteristic of *geographic distribution* between their elements. In an SoS, constituent systems may be located in many different places. In the same line, agents in an MAS can also be situated in distinct spots.

Third, constituent systems of an SoS are heterogeneous and provide functionalities to achieve a higher purpose. In the same way, the *heterogeneity* is also a trait in an MAS which agents are responsible for performing different tasks to accomplish a common goal.

Moreover, as we stated in Chapter 2 one of the main characteristics of SoS is their *evolutionary nature*. Because we have chosen using in our approach a MAS based on a broker structure (stated in the previous section), we believe it can be useful for supporting the evolution in SoS. We think that thanks to the decentralization and loose coupling provided by such a structure, elements in an SoS can be easily added or removed.

Furthermore, MAS agents are autonomous in the sense that they act without direct intervention of humans or other agents, and have control over their own actions and internal states. SoS follow the same idea concerning managerial independence (details in Chapter 2) between their constituent systems. SoS systems are governed by their own rules rather than by external ones. However, in an SoS constituent systems can be owned or operated directly by humans, i.e., nothing prevents that.

In addition, although, agents are autonomous and self-governed, they may interact with human beings. This is the case of Personal Assistant agents (PA) that interact and provide assistance to humans, but as agents they still keep autonomy. For instance, PAs can show proactive behavior and depending on the task they can refuse to perform it, for example, in the case they are overloaded. In the case of SoS, constituents may have operators. Thus, human *users* may be part of SoS and also of MAS.

We believe that by sharing such a number of characteristics with SoS, MAS can be a useful element in our approach for supporting the development of SoS. The point is that MAS already provide a structure capable of fitting many SoS characteristics, thus to our mind it can be useful to facilitate and avoid redoing things, specially concerning SoS cooperation, geographic distribution, heterogeneity, evolutionary nature, independence, and users.

Despite sharing many common characteristics, MAS and SoS have several important differences between them. Following we highlight the main differences between them.

Because of the inherent modularity, agents of an MAS follow the tendency of being designed with a low degree of complexity. That is, the problem is naturally decomposed into “small pieces” and distributed among the agents. On the other hand, SoS are

usually assembled with larger existing systems or legacy systems. The systems are from different nature or domains, and they are not designed or built from scratch especially for the SoS. The degree of complexity in SoS systems is much higher than those of agents. Moreover, due to their lower degree of complexity the number of agents in an MAS is typically much larger than the number of systems in an SoS.

Besides, when we speak about operational independence (Chapter 2) the agents of an MAS are not capable of working in complete isolation, i.e., they are part of an overall solution and dependent on the cooperation with the other MAS agents (Section 3.1.2). Conversely, the systems of an SoS are operational independently from one another, meaning that they keep performing their own and natural tasks at the same time they cooperate with the SoS.

3.3 Summary

In this chapter we have presented fundamental concepts of Multi-agents systems (MAS) related to the scope of this research. We have started the chapter by presenting the main definitions and characteristics of an agent. Then, we have highlighted the main differences between reactive and cognitive agents, the latter keeping a representation and knowledge about their environment. After that, we have described and shown characteristics of a special kind of agent, named Personal Assistant (PA), specialized in interacting with humans. Next, we have presented MAS describing their characteristics and different kinds of architectures used as structures for supporting the interactions and cooperation between MAS agents. We have also stated that the MAS used in our approach adopts a structure based on a brokering mechanism, mainly because it fits well the SoS evolutionary nature. We have ended this chapter by making a discussion showing why MAS are a key element in the approach proposed in this research, and by describing the main differences between an MAS and an SoS.

Chapter 4

An architecture for facilitating the development of Systems of Systems

This chapter presents the Memory-Broker-Agent (MBA) architecture which is the core of the approach proposed by this research to facilitate the development of Systems of Systems.

Comparing to the works shown in Chapter 2, the relevance of the approach proposed by this research is that it is a framework based on a domain-independent peer-to-peer (P2P) architecture with loose coupling between its elements, focusing on the development of SoS from a practical point of view. One of the advantages of the proposed architecture is that it does not rely on centralized components or keep references between each of its elements. Moreover, thanks to the loose coupling provided by our architecture, the SoS developed with it can be easily adapted or changed between different domains. Furthermore, our approach provides means for knowledge capitalization and management during SoS operation, and uses a single communication protocol providing syntactic and semantic interoperability between constituent systems. Moreover, it takes into account the user dimension by providing proactive interfaces capable of interacting with SoS users through dialogues with Personal Assistants, and offering proactive support when they are using the SoS. In addition, our approach considers robustness during SoS operation, trying to support it keeping coherent when constituent systems go down.

In the chapter, first we introduce the Memory-Broker-Agent Model on which the proposed approach is based. Then, we present our MBA architecture for facilitating the development of Systems of Systems. After that, we describe the important role played by ontologies in our approach (Section 4.3), and then we present the *work order*, which is the communication protocol used by our architecture.

4.1 The Memory-Broker-Agent model

In this section we introduce the Memory-Broker-Agent (MBA) model which is used by the architecture proposed in this research. The goal is to understand how such a model can be appropriate to facilitate the development of SoS, and also to provide the definitions needed for its comprehension. Next, we first give an overview showing how the MBA model can facilitate the SoS development. After that, we present a formal definition of the model.

4.1.1 Overview

The MBA model consists of agents, brokers and memories. Following we described how each of these elements can be appropriate to facilitate the development of SoS.

The first element of the model are agents. We chose them to be part of the model for two reasons: (i) because they share common characteristics with SoS, as we stated in Chapter 3; and (ii) for the sake of standardizing the elements of and interactions in an SoS.

First, SoS and MAS share common characteristics such as geographical distribution, evolution and cooperation to achieve a common goal. This compatibility has motivated us to use agents as they can support naturally SoS regarding the shared characteristics.

The second reason for using agents was that as SoS lie in a heterogeneous environment, we have intended to standardize the SoS elements (constituent systems) and the exchanges (interactions) between them, in order to try to facilitate the SoS development. To do that, we proposed to agentify the SoS aiming to provide a common medium for SoS elements and the exchanges between them.

Another element of the model are brokers. In this research, the notion of broker comes from the broker pattern as it has been discussed by Buschmann et al. (1996) and Shen et al. (2003). Basically, brokers are intermediaries responsible for receiving requests, trying to find suitable providers and then transferring the results back to the callers. According to Buschmann et al. (1996), the broker pattern is suitable for distributed and heterogeneous environments with independent cooperating components. Moreover, it is appropriate when we need to add, remove, exchange, and locate such components.

The choice for adding the notion of broker to the model of our architecture was because it fits especially well SoS, mainly concerning its heterogeneity, evolutionary nature and interoperability. For instance, by using brokers constituent systems remain loosely coupled to the other ones as they are not connected directly with each other, making it easier to adapt and change the SoS.

The last element of the MBA model are memories. Our intention in adding them to the model was to allow the knowledge capitalization and management in SoS. The main advantage provided by memories to SoS is the possibility for reusing knowledge from them and their domains. Below we give some examples showing how memories can be useful and facilitate things in SoS.

First, when SoS systems request some information, depending on the situation, it is interesting to check first if it was already provided on the past. If memories are being used and keeping knowledge related to the SoS, then such an information could be easily retrieved and provided to requesters. This avoids letting systems redoing tasks or being overloaded with unnecessary work. Second, another example is when we have systems not belonging anymore to the SoS, but for some reason it needs later a given information provided by them to achieve the expected results of its goal. In this case, if the SoS has not used memories, then it can be necessary to reconnect those systems again to provide the information required. Finally, when a constituent system goes down and fresh information is not required, memories could be used to support the SoS, providing information stored from such a system while it is not reconnected to the SoS.

Next, we present a formal definition of the MBA model.

4.1.2 Formal definition

Definition 4.1. Formally, the MBA model can be defined as a tuple $\psi := (M, B, A, Y)$, where

- M is the finite set of memories (m be a memory belonging to M).
- B is the finite set of brokers (b be a broker belonging to B).
- A is the finite set of agents (a be an agent belonging to A).
- Y is a ternary relation among them, i.e., $Y \subseteq M \times B \times A$. This relation specifies the coupling among parts.

The role of the agents (Chapter 3) is to interface constituent systems in an SoS, letting them exchange information and cooperate with each other. In this context, let x be an SoS and S be the set of constituent systems (s is a constituent system belonging to S) of x. Then, we define the concept of *proxy agent* as follows:

Definition 4.2. A *proxy agent* is an agent a that interfaces a constituent system s in an SoS x, allowing such a system to exchange information and cooperate with other constituent systems of x. The proxy agent a is responsible for translating the requests from s into a common communication protocol understandable by the other proxy agents

interfacing the constituent systems of x. When receiving information, a takes the responsibility of doing the reverse process, translating it from the common communication protocol into the raw format used by s.

Formally, we can represent the process of interfacing a constituent system s by a proxy agent a by the sentence: $i(s,a) \rightarrow ((m,b,a) \in Y \wedge (s \in S) \wedge (i \in I))$, where i is an interface belonging to the set I of interfaces between constituent systems and proxy agents.

Another component of the MBA model are brokers. They are "intermediaries" that receive requests for functionalities and try to find providers offering them. Once the tasks are allocated, brokers are responsible for transferring the results back to the requesters. Formally, we define brokers in this research as:

Definition 4.3. *Brokers* are intermediaries that receive requests for functionalities from constituent systems through their proxy agents, try to find potential providers through their proxy agents, and then when results are available they are responsible for transferring them back to the proxy agents of requesters.

The process of receiving requests for functionalities, finding providers, and transferring results back, realized by brokers can be formally represented by the sentence: $D_f(i(s,a),f) \rightarrow ((m,b,a) \in Y \wedge (i \in I) \wedge (f \in F))$, where D_f is a function used by the constituent system s, interfaced by the proxy agent a through the interface i, to request the functionality f (f is a constant and belongs to F, the set of functionalities belonging to the SoS x). Note that the request made by s is handled by the broker b. The function D_f returns a list l of results found by b, i.e., $D_f : I \times F \rightarrow l(\forall s((s,f) \in S \times F) \rightarrow D_e(s,f))$, where D_e is a function used by the constituent system s to perform the functionality f, returning the results R (set of results for functionalities in the SoS x) after doing that. The function D_e is formally represented by $D_e : S \times F \rightarrow R$.

The last component of the MBA model are memories. Their function is to capitalize and manage knowledge from the SoS and its domain, in order to be reused later by constituent systems and SoS users. Memories are defined in this context as:

Definition 4.4. *Memories* capitalize and manage knowledge from an SoS and its domain, when requested by constituent systems through their proxy agents.

Formally, the process of capitalizing and managing knowledge performed by memories can be represented by the sentence: $D_{f_m}(i(s,a),f_m) \rightarrow ((m,b,a) \in Y \wedge (i \in I) \wedge (f_m \in F_m))$, where D_{f_m} is the restriction in the function D_f on a memory m, meaning that D_{f_m} handles requests for functionalities provided by m. Moreover, in a such sentence f_m is the functionality requested provided by m, and F_m ($F_m \subseteq F$) is the set of functionalities provided by m. The function D_{f_m} can be represented as $D_{f_m} : I \times F_m \rightarrow l(\forall s((s,f_m) \in S \times F_m) \rightarrow D_e(s,f_m))$ with $D_e : S \times F_m \rightarrow R$.

After defining the Memory Broker Agent model, the next step is to present the MBA architecture proposed by this research to facilitate the development of Systems of Systems. Following, in the Section 4.2 we present the proposed architecture.

4.2 The MBA architecture

The architecture proposed by this research was designed according to the following assumptions:

- The architecture focuses on developing SoS from a practical point of view. That is, it provides a structure for constituent systems exchange information and cooperate during SoS operations.
- The architecture is based on the MBA model (Section 4.1).
- The exchanging of messages between constituent systems is made through a single communication protocol which is named *work order* (details in Section 4.4).
- The architecture relies on ontologies for modeling the SoS domain, modeling the users of constituent systems, handling the semantics of the communication protocol (work order), and decoding interactions in natural language. Details about the role of ontologies in our approach are shown in Section 4.3.
- Each constituent system of the SoS is able to provide at least an API (Application Programming Interface).
- The goals, requirements, and constituent systems for the development of the SoS are provided by the organization.

A minimal example of the proposed architecture which is based on the MBA model is shown in Figure 4.1. The MBA is a domain-independent core architecture, meaning that it is extended according to the domain, goals and systems of the SoS being developed. In the architecture, there are three main elements: "system," "broker" and "memory." We describe and detail each of these elements of the MBA architecture in the sequence.

The "system" element represents a constituent system of an SoS being developed with the architecture. To become a constituent system of an MBA SoS, a system must provide at least an API (Application Programming Interface), in order to be interfaced by a proxy agent (Section 4.1). Figure 4.2 shows a system being interfaced by a proxy agent. After being interfaced the constituent system will consist of its API, a communication protocol, ontologies, an optional interface, and a functionality for synchronization ("sync").

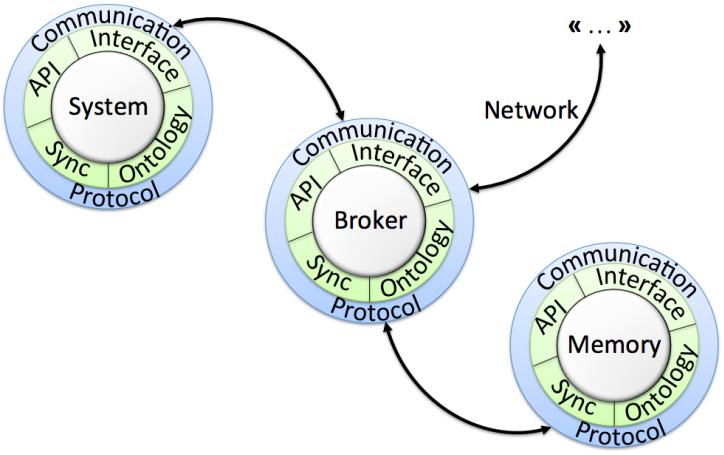


Figure 4.1: The MBA architecture proposed by this research.

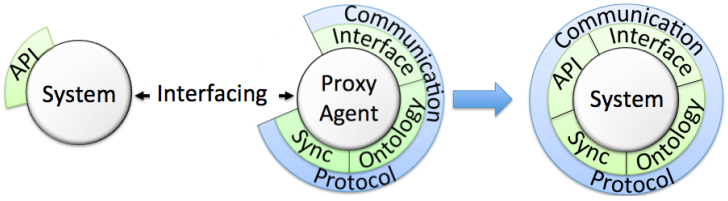


Figure 4.2: A system being interfaced by a proxy agent.

The communication protocol is used to exchange messages and information between the system and the other constituent systems in the MBA SoS. In the MBA architecture, when systems exchange information they make use of a single communication protocol, named *work order*, which is the protocol provided by our approach. We detail the work order communication protocol in Section 4.4.

The ontologies of the constituent system have different roles like modeling the SoS domain and system users, providing a vocabulary and semantics for the communication protocol, and decoding interactions in natural language. Details of the roles of ontologies in our approach are shown and explained in Section 4.3.

Moreover, after being interfaced by a proxy agent, the system will be composed of an optional interface that can be used by the system users to request and receive information from the SoS. The goal of such an interface is to take into account and support the user dimension in SoS. The interface provides an interaction in natural language to the system users. Moreover, such an interface is capable of acting proactively to support system users with their tasks, aiming to achieve the expected results of the SoS goal. The interaction between users and the interface is made through dialogues with Personal Assistants. We detail and show how to integrate such an interface with an MBA system in the Chapter 5.

Another component of the MBA system is a functionality for synchronizing it with other constituent systems. This is an extended functionality provided by the MBA architecture

aiming to support the SoS robustness and facilitate the work of SoS architects. For instance, the synchronization can be used to let systems coherent with the current state of the SoS, when they go down. Details about the process of synchronization between systems are provided in Chapter 5.

The second element of MBA architecture is the "broker." This element intends to receive requests for functionalities from constituent systems, try to find potential providers, and then once the tasks are allocated transfer the results back to the requesters. Besides providing loose coupling between systems (Section 4.1.1), an important characteristic of the broker of our architecture is that it is a multi-agent system (MAS). A broker MAS consists of several agents "brokers," and in our approach it is possible to have as many "brokers" as needed. The use of a broker MAS has some advantages such as: (i) it avoids overloading a single broker, thus enhancing efficiency; and (ii) it makes the MBA architecture more robust, as it does not rely on a centralized element.

The last element of the MBA architecture is the "Memory." The memory goal is to capitalize and manage knowledge concerning the SoS and its domain. In our approach, it is possible to have as many memories as needed or required by a given domain, including redundant elements. The main advantage of memories in our approach is that we are able to reuse knowledge from the SoS and its domain. The MBA architecture provides a generic interface for memories through a proxy agent with an *Abstract Statement* for handling four basic memory operations, named *insert* to insert data in the memory, *update* to update data in the memory, *remove* to remove data from the memory, and *select* to retrieve data from the memory. The Abstract Statement is defined by the following grammar in EBNF format (Wirth, 1977):

```
<Abstract Statement> ::= <operator> <class> <data> [<result>]
<operator> ::= <insert> | <update> | <remove> | <select>
<data> ::= {<property> (<value> | {<abstract-statement>})}
```

The advantage of doing this, is that our approach facilitates the work of SoS architects, requiring them only to customize such a proxy agent according to the SoS being developed.

To summarize: (i) our architecture is domain-independent; (ii) it focuses on the development of an SoS from a practical point of view; (iii) it uses a P2P approach; (iv) it provides loose coupling between its elements; (v) it can capitalize and manage SoS knowledge; (vi) it takes into account the user dimension in SoS; and (vii) it considers SoS robustness.

First, the MBA architecture is not constrained by a specific domain, meaning that our approach is domain-independent for developing SoS. When developing an SoS with our

approach, the SoS architect is provided with a core architecture, and it is up to him to extend it according to the domain, goals and systems of the target SoS.

Moreover, the approach proposed in this research intends to facilitate the development of SoS in practice, i.e., it provides a structure for constituent systems exchange information and cooperate during SoS operations. This is done by agentifying the SoS through interfaces between systems and proxy agents.

Furthermore, the MBA architecture lies in a P2P fashion thanks to its proxy agents and the MAS broker. This means that our approach is not dependent on centralized components or services relied by all the other architecture elements. When centralized components are present, once they face problems they can be spread through all the other elements of the approach, making it more brittle.

Our approach also provides loose coupling between its elements thanks to its brokers. Thus, an MBA SoS can be adapted or changed easier, in order to handle the evolutionary nature of SoS or to be applied to different domains.

In addition, thanks to its memories the MBA architecture is able to capitalize and manage SoS knowledge. This facilitates the reuse of knowledge later, avoiding possible reworking of systems or the SoS architect.

Finally, the users of constituent systems can receive support through natural language interfaces, and the MBA systems can be synchronized to try to support the SoS robustness.

After presenting the MBA architecture and its elements, we next present in Section 4.3 the crucial role played by ontologies in our approach.

4.3 The role of ontologies

In this section we present the role of ontologies in the approach proposed by this research. Understanding it is fundamental, since ontologies constitute the backbone of our architecture. We thus developed a specific ontology named *OntoMBA* which is a core ontology provided by our architecture. The goal of the OntoMBA is to be used as a point of departure by SoS architects when developing the ontologies of an MBA SoS. Thus, the OntoMBA intends to be extended and customized according the requirements and specifications of the SoS being developed.

The OntoMBA consists of concepts, relations and attributes describing the SoS in terms of its domain, its constituent systems and its users. Such an ontology plays four different roles: (i) modeling the SoS domain; (ii) handling the semantics of the communication protocol; (iii) decoding interactions in natural language; and (iv) modeling the users

of constituent systems. For matters of space, we detail OntoMBA in Appendix A. Hereafter, we detail each of the OntoMBA roles.

4.3.1 Modeling the SoS domain

An essential role of the OntoMBA is to model the SoS domain, providing semantics and serving as a common vocabulary for the SoS being developed. The act of building a common vocabulary is especially important in the context of SoS, as they are usually composed of different systems not always belonging to the same domain. When extending the OntoMBA for modeling the SoS domain, we usually describe concepts such as "Organization," "SoS," "Goal," "System," and "User."

We consider the OntoMBA developed for modeling the SoS domain, as the *global ontology* of an MBA SoS. This ontology is kept within the proxy agents interfacing the memories of an MBA SoS.

4.3.2 Handling the semantics of the communication protocol

The OntoMBA also helps to define the communication protocol used by constituent systems in the MBA architecture to exchange information. In the MBA architecture, systems use the work order protocol (Section 4.4) for exchanging information. The OntoMBA supports the work order regarding the semantics of the message content and the action (i.e., the functionality) being requested. In the work order, the properties of the message content and the action being requested correspond to concepts of the OntoMBA ontology. The main goal of the OntoMBA in supporting the work order protocol is to allow a syntactical and semantic interoperability among constituent systems of an MBA SoS when they exchange information.

The ontologies used to support the semantics of the work order are derived from the OntoMBA global ontology (Section 4.3.1), and are kept within proxy agents, hence we call them as *proxy agent ontologies*. Note that such ontologies may differ in each proxy agent, being specialized according to the idiosyncrasies of the systems they are interfacing. Thus, the ontologies of two given proxy agents need not be the same but must be at least compatible, in order to allow them to exchange information coherently. In addition, when the SoS architect is deriving a proxy agent ontology, usually the main concept to describe is the "Functionality" which concerns to the SoS functionalities the system interfaced with proxy agent can request or provide.

4.3.3 Decoding interactions in natural language

Another role of the OntoMBA is to decode the interactions in natural language in our approach. As we stated in Section 4.2, after systems are interfaced by proxy agents they can have an optional interface providing interactions in natural language. Such interfaces can be used for supporting the users of constituents to interact with the SoS, aiming to achieve the expected results of its goal. In this context, the OntoMBA intends to help decoding, interpreting and understanding the users' utterances in natural language.

The ontologies used to decode the interactions in natural language are the own proxy agent ontologies (Section 4.3.2) of constituent systems.

4.3.4 Modeling the users of constituent systems

The OntoMBA is also used to model the users of constituent systems. The goal is to represent the behavior of the user of a constituent system when interacting with it, in order to provide personalized support to her for achieving the expected results of the SoS goal. When the OntoMBA intends to model the user of a constituent system, we call it *User Model* (UM).

The UM of a user is derived from the proxy agent ontology (Section 4.3.2) of her system. Each UM is kept within the staff agent of the Personal Assistant (PA) of a proactive interface (presented and described in Section 5.1). When we derive a UM the main focus is to describe the concepts of "User" (if needed), "Score," and "Shared-Info."

After presenting the role played by ontologies in the proposed approach, we next present in Section 4.4 the communication protocol used in the MBA architecture.

4.4 Work order: The MBA communication protocol

In this section we present the work order which is the communication protocol used by the constituent systems of the MBA architecture. The protocol is used by constituent systems to request functionalities and exchange information between them.

First, we present a formal definition of the protocol to provide a better understanding of the concepts it is based on. Then, in order to highlight its features, we show different examples applying the work order protocol.

4.4.1 Formal definition

In order to understand the communication protocol of our approach, we first need to introduce some definitions. However, to facilitate the comprehension of such definitions we provide a simple example illustrating them as they are being defined.

First, we define the concept of *pattern*:

Definition 4.5. A *pattern* in the MBA architecture specifies how the results from a request should be structured in order to be understandable by the requester. It is a list of linguistic cues expressed as a tree of ontological terms. These terms are concepts and attributes taken from the ontology of the requester's proxy agent.

Patterns with concepts, attributes and relations to other concepts can be described by the following grammar in EBNF format:

```
<pattern> ::= (<concept> ({<attribute>}*)) | {<relation> <pattern>}*
```

Then, we define the concept of *work order*:

Definition 4.6. A *work order* describes a request for functionality made by a given constituent system in the MBA architecture. Formally, it can be defined by the following EBNF:

```
<work order> ::= (<message-content>, [<action>], <timeout>,
                  [<contract-net strategy>])
```

Where,

- `<message-content>` is the content or arguments of the work order, which are detailed below in the EBNF describing the message content.
- `<action>` is an optional parameter, being the name (a keyword) of a functionality requested. To succeed in requesting by using `<action>`, the requester must know the exact name of a functionality being offered.
- `<timeout>` is a number representing the maximum amount of time a requester will wait to receive an answer for its request.
- `<contract-net strategy>` is a parameter used by requesters to specify the strategy used in a Contract Net (Smith, 1980) to find out functionality providers.

The MBA architecture adopts the approach that systems are permitted not to answer requests, thus timeouts can be specified in the work order. There may be different reasons for a request not to be answered: either the proxy agents interfacing the systems

are fully busy working on several requests, or they simply do not want to answer. Doing so, it avoids for example the fact that if answers are always required, and one of the systems is very busy, and a new request arrives, then either the requester would be blocked for an unknown length of time until it gets its answer, or the busy system would have to stop its tasks to provide an answer. In the last case, other problems could arise like delaying the tasks being performed.

The arguments or content of a work order, i.e., `<message-content>`, can be defined by the following EBNF:

```
<message-content> ::= ([<query>], [<data>], [<pattern>], <language>)
```

where,

- `<query>` represents the request made in a query format (the ontologies of the proxy agents of requesters and providers must be at least compatible).
- `<data>` stands for some input data.
- `<pattern>` a pattern determining how results should be structured.
- `<language>`the language used in the message content.

Now, to illustrate a request made by a constituent system of the MBA architecture, let s be a system belonging to S (set of constituent systems of an SoS developed with the MBA architecture). Supposing that the system s needs a specific functionality, say the address of an employee named "Lambert," then it will send a work order to a broker b ($b \in B$) such as:

```
<work order> ::= (<message-content> ::= (<query> ::= ("person"("name" is
                  "Lambert")), <pattern> ::= ("address"("surname")("name")
                  ("city")("country")),
                  <language> ::= "English")
                  <timeout> ::= "10 seconds")
```

where "person" is a concept and "name" is an attribute of the ontology of the requester's proxy agent. Note that in this simple example we used a query format to perform the request, and we did not use the `<data>` , the `<action>` and `<contract-net strategy>` optional parameters of the work order.

When the broker b receives the work order from s, it tries to find providers for the requested functionality. To do that, b broadcasts the work order of s to all the other systems of S (the default behavior when the Contract Net is not used), and waits the timeout defined, in this case 10 seconds.

After accomplishing the allocated tasks, providers are responsible for structuring the results in a format understandable by the requester. To do that, they perform what is called *summarizing process*:

Definition 4.7. *Summarizing* is the process realized by providers to structure the results of their tasks, associating them with the concepts and attributes of a pattern given by a requester when requiring a functionality by means of a work order.

For instance, the results found for the address of "Lambert" will be retrieved in some raw data type used by providers to store the information, like a string. In this example, for the name "Lambert" two results were found and the raw data retrieved is the string:

```
raw-result = "Mathilde Lambert, Paris, France; Andrew Lambert,
              Sydney, Australia"
```

Then, to structure the results, providers perform a summarizing to fill the pattern provided by the requester s. The pattern filled with all the addresses retrieved for named "Lambert" will be:

```
<pattern> ::= (("address"("surname" "Lambert")("name" "Mathilde")
                          ("city" "Paris")("Country" "France"))
               ("address"("surname" "Lambert")("name" "Andrew")
                          ("city" "Sydney")("country" "Australia"))
              )
```

Once providers have completed the summarizing process, they send the results back to the broker b. When b receives the first answer it transfers it automatically to the system s, and sets the timeout to zero. In the case there are more providers performing the requested functionality, after taking the first answer the broker b will send a message to those providers to cancel the tasks, and any answers coming after that will be discarded by b. This is the default behavior adopted by the work order when a Contract Net is not being used.

As we have stated in Section 4.3.2, the work order protocol is supported by ontologies (stored in the proxy agents), in order to provide a **syntactic and semantic interoperability** between constituents systems of an MBA SoS. Thus, we restate that in order to exchange information coherently, the ontologies of two given constituent systems need not be the same but must be at least compatible.

After presenting the main definitions of the work order protocol, we next show the potential and the advantages of our protocol through several different examples.

4.4.2 Examples of the work orders

In this section we show some examples applying the work order protocol. To exemplify the protocol, we use the MBA SoS shown in Figure 4.3. This example of MBA SoS is composed of four constituent systems: a database based on object data model, a database of relational data model, a base on key-value data model, and a database of triplestore data model.

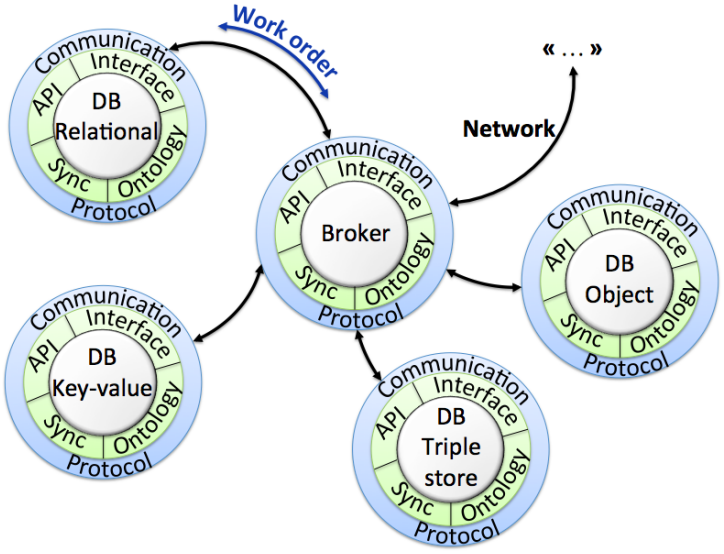


Figure 4.3: The MBA SoS used to exemplify the work order protocol.

In the **first example**, the goal is to send a work order for requesting a functionality to store a given address. For instance, say the system "DB Object" is the only one containing the address of "Andrew Lambert," and it wants to store such an address in the other three systems ("DB Relational," "DB Key-value" and "DB Triplestore") of the SoS. Then, the "DB Object" will send a request to his proxy agent that in turn will translate it as a work order such as:

```
<work order> ::= (<message-content> ::=
                      (<query> ::=
                          ("functionality"("name" is "store address")),
                       <data> ::=
                          ("address"("surname" "Lambert")
                                    ("name" "Andrew")
                                    ("city" "Sydney")
                                    ("country" "Australia")),
                       <language> ::= "English")
                    <timeout> ::= "10 seconds")
```

Where in the `<query>` parameter, the "functionality" is a concept and "name" is an attribute of it. Moreover, in the `<data>` parameter, the "address" is a concept and the

"surname," the "name," "city" and "country" are attributes of it. Note that, these concepts and attributes were extracted from the "DB Object" proxy agent.

In this work order we have not used the `<pattern>` , the `<action>` and the `<contract-net strategy>` parameters. As this is a request to store information without retrieving results, we have not specified the pattern. Furthermore, the request is made using a query format, thus the action has also been ignored. Furthermore, in this example we will not use a Contract Net in our work order. On the other hand, in this example we used the `<data>` parameter to specify the address we want to store, i.e., the input data of the request.

After that, the proxy agent of the "DB Object" system will send the work order to the broker, waiting for a timeout of 10 seconds to receive some answer. When the broker receives the work order from the "DB Object," it will perform a broadcast to all the three other database systems to know which of them can store addresses. Figure 4.4 symbolizes the broadcast performed by the broker to send the work order in this first example.

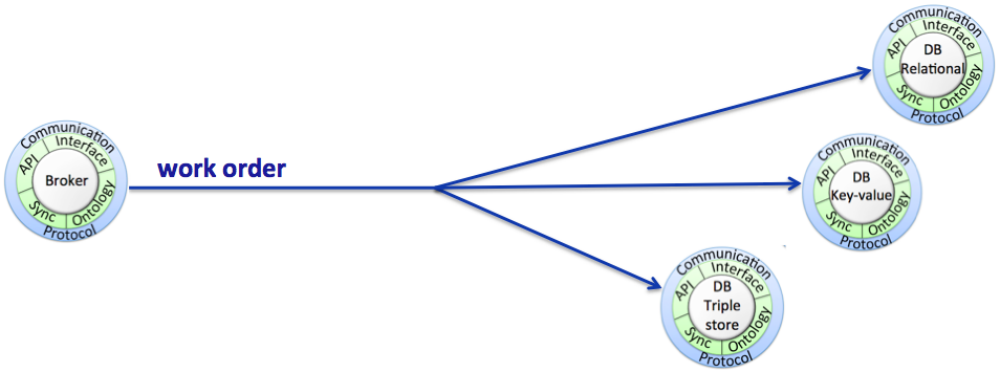


Figure 4.4: The broadcast performed by the broker to send the work order in the first example to demonstrate the protocol of the MBA architecture.

Then, when the databases receive the work order, the first thing they do is to check if they can provide the functionality. In this case, their proxy agents will take the query from the work order received and will perform a reasoning on their ontologies, in order to know if the systems they interface can provide the requested functionality.

Here, we assume that all the three databases can provide the functionality to store addresses, and that they are available to provide it as soon as they receive the work order, i.e., they are not performing other requests or are overloaded. Then, each proxy agent of the providers will take the data from the work order, and will then translate it to the raw format used by its system, for instance, to the format of a relational, key-value or triplestore data model. After that, the systems will receive the information translated by the proxy agents and will start to perform the tasks.

When the tasks are performed, the three providers send an answer back saying that they have accomplished the job. Then, the broker transfers it to the requester ("DB Object").

After describing the first example, some important points can be highlighted about the work order protocol: (i) only a single work order is needed to request and perform a given functionality in all of the systems providing it; and (ii) the interoperability provided by the protocol.

First, as we have shown through our example, only a single work order was needed to request and perform a functionality in different constituent systems of an MBA SoS. This means that, constituent systems of an MBA SoS need to send only a single work order to request and perform functionalities in all of the systems providing them. The advantage of doing it, is that our approach removes the load from requesters regarding the need of sending the same request several times, one for each system in the SoS. Moreover, it also avoids overloading the network of organizations with useless messages.

The second point is related to the interoperability provided the protocol. In the example, when the providers have received the work order they were capable of understanding what was being requested and the information associated with it. The work order protocol is capable of providing a syntactical and semantic interoperability between constituent systems. This is thanks to the ontologies of proxy agents which as we have stated in Section 4.3.2 must be at least compatible.

In the next examples to demonstrate the work order, we will simplify the description of some steps when using the protocol. For instance, we assume that the process of translating a message in a raw from a system into a work order, or vice versa is already being done intrinsically. The reason is that we plan to focus on the main points intended to be shown by the examples, thus avoiding unnecessary information already shown on the previous examples.

In the **second example** to demonstrate the work order protocol, the goal is to send a request to retrieve a given address. Supposing that the "DB relational" system wants the address of "Catterina Brooks," then it will send to the broker a work order such as:

```
<work order> ::= (<message-content> ::=
                     (<action> ::= ("retrieve address"),
                      <data> ::=
                         (("surname" "Brooks")
                          ("name" "Catterina")),
                      <pattern> ::=
                          ("address"("surname")("name")
                                    ("city")("country")),
                      <language> ::= "English")
                  <timeout> ::= "10 seconds")
```

When the broker receives the work order from the "DB relational," it will broadcast it to all the other systems in the SoS to find potential providers. Then, the three database systems will receive the work order and perform a simple syntactical check, as in this case the request is being performed using an action and not a query, to know if they can provide the functionality. Assuming that only the "DB Key-value" and the "DB Object" are able to retrieve addresses, then they will start to perform the task as soon as possible for them. The other system, i.e., the "DB triplestore" will ignore the request and not answer it, as it cannot provide the functionality.

After retrieving the address of "Catterina Brooks," the providers perform a summarizing to structure the results according to the pattern provided. For instance, supposing the raw format of the retrieved result was:

```
raw-result = "Catterina Brooks, Rome, Italy"
```

Then, the summarizing performed by the system to fill the pattern provided will structure it to:

```
<pattern> ::= ("address"("surname" "Brooks")("name" "Catterina")
                        ("city" "Rome")("country" "Italy"))
```

After doing that, the "DB Key-value" and the "DB Object" send the results back to the broker. By default, the broker will take the first answer received that in turn will be transferred to the "DB Relational" system. The other answer will be discarded.

Through this second example, one may see that with the work order protocol systems are also able to perform requests directly in a syntactical level, by providing the name of the functionality required. Moreover, the default strategy adopted by the protocol to let the broker take the first answer returned by providers, can enhance efficiency as it does not need to wait until the all the answers are completely processed. Furthermore, this example has also shown that only the systems providing the functionality will answer, i.e., the other ones do not need to waste their time creating a new message, and it avoids overloading the broker and the network with useless messages.

In the **third example** the goal is to send a request to retrieve an address according to some criteria. For instance, say the "DB triplestore" wants the address of "" coming from the system(s) able to start performing the task as soon as possible. To do that, the "DB triplestore" sends to the broker a work order such as:

```
<work order> ::= (<message-content> ::=
                    (<query> ::= ("person"("name" is "")),
                     <pattern> ::=
                         ("address"("surname")("name")
                                   ("city")("country")),
                     <language> ::= "English")
```

```
<timeout> ::= "10 seconds"
<contract-net strategy> ::= "collect-answers")
```

In this case, when the broker receives the work order it will broadcast a message to form a Contract Net, asking for "bids" to find potential providers. In a such situation, the broker assumes the role of a "manager" and the providers of "contractors." The Contract Net is formed according to the strategy provided in the work order. Currently, the protocol allows some strategies, named *collect-answers*, *take-first-answer*, and *collect-first-n-answers*.

The collect-answers strategy waits and collects answers (before a timeout occurs) in order to assign the work to one or more providers. On timeout, if answers were received, the broker selects the better one(s) with respect to some criteria, in order to allocate the tasks. The criteria may be for instance related to costs, quality or time. It is up the SoS architect to define it according to his requirements.

In the take-first-answer strategy, as soon as the broker gets an answer from a provider, it accepts it and establishes a contract with the provider to perform the tasks.

The collect-first-n-answers strategy aims at taking a specific number of answers and then to start processing the tasks. That is, when the broker broadcasts a message to form the Contract Net sometimes it knows exactly which are the systems providing the functionality.

For the three strategies above, if no answers are received from service providers, then the broker returns a failure message to the requester. If other strategies are required, they need to be implemented by the SoS architect.

Following our example, the strategy provided in the work order was the "collect-answers." Here, we adopt the criteria to form the Contract Net refers as the time providers are able to start performing the task requested. Supposing all the three systems ("DB Object," "DB Key-value" and "DB Relational") can perform the task, then they will answer the broker with their "bids," i.e., the time each one is able to start doing the job.

After receiving the "bids," the broker chooses the best one which in this case is the shortest time to start to do the task. Assuming the "DB Object" can do the job first, then the broker will grant it the task. When, the "DB Object" finishes the work, it sends the results back to the broker that in turn transfers them to the "DB triplestore."

Through this third example, we have intended to emphasize an important aspect of the work order protocol which is the possibility to select providers according to a given strategy. By defining different strategies for a Contract Net through the work order, requesters of an MBA SoS can select providers to perform the tasks based for instance, on the fastest answer received, on the number of providers, on time or cost, among others.

4.5 Discussion

We began the chapter by introducing the Memory-Broker-Agent (MBA) model on which the proposed architecture is based. First, we described how the model is appropriate to facilitate the development of SoS, and then we presented a formal definition for it. The model consists of agents, brokers and memories, providing several advantages to facilitate the development of SoS such as:

- It fits naturally to the distributed, evolutionary and cooperative nature of SoS.
- It standardizes the heterogeneous environment of SoS.
- It provides loose coupling between constituent systems, making it easier to adapt and change the SoS.
- It provides the possibility for reusing knowledge from the SoS and its domain, avoiding for instance, reworking for SoS systems and users.

After introducing the model, we presented our MBA architecture for facilitating the development of SoS. The relevance of the proposed approach is that it is a domain-independent P2P architecture with loose coupling between its elements, focusing on the development of SoS from a practical point of view. In addition to the advantages of the described above, it is worth to highlight that the MBA architecture: (i) uses a single communication protocol between constituent systems; (ii) takes into account the user dimension to support SoS users; and (iii) considers SoS robustness.

In this chapter we also showed the crucial role played by ontologies in our approach. They are the backbone of our architecture, providing the semantic glue between its elements.

Finally, we presented and detailed the communication protocol used by the MBA architecture which is named *work order*. First, we presented a formal definition describing the main concepts and elements of the protocol, and then we demonstrated it through examples to show its potential and features. The protocol used by our approach offers some advantages for constituent systems, for instance, syntactical and semantic interoperability, and the possibility to select providers according to a given strategy.

In the next chapter we present and detail the extended functionalities provided by the MBA architecture, named *proactive interfaces* for supporting the user dimension in SoS, and *synchronization between constituent systems* to support SoS robustness by trying to make an SoS coherent when its systems go down.

Chapter 5

Extended functionalities provided by the MBA framework

This chapter presents extended functionalities provided by the MBA framework. Currently, such functionalities focus on user dimension and robustness.

As we have stated in Chapter 2, the user dimension is a challenge in an SoS mainly because of its dynamics and changing environment, leading SoS users to suffer from switching context and excessive cognitive loads. Thus, our approach in this research tries to help SoS users by providing more flexible proactive interfaces in natural language.

Moreover, SoS have an inherent emergent behavior resulting from interdependencies among heterogeneous systems and from the SoS evolutionary nature (Chapter 2). Such a behavior enriches capabilities but may also introduce vulnerabilities to the SoS. Thus, considering and supporting robustness in an SoS is fundamental. Besides providing loose coupling and thus reducing interdependencies, the MBA framework offers a functionality for synchronizing constituents during SoS operation, in order to recover gracefully when they go down bringing the SoS back to a consistent state.

In the chapter, we first present the proactive interfaces. After that, we highlight the main points of the synchronization among constituent systems for supporting SoS robustness. For the sake of readability and matter of space of this thesis, we detail our approach for synchronization in Appendix B.

5.1 Proactive interfaces

To support SoS users our approach provides an optional interface that can be easily integrated into constituent systems of an MBA SoS. The interfaces offered by the

MBA framework are proactive interfaces interacting with users through Natural Language (NL). The interaction is made through dialogues with Personal Assistant agents (Chapter 3) capable of demonstrating proactive behavior. The interfaces can: (i) act proactively for supporting users cooperation with the SoS and achieve the expected results of its goal; (ii) provide customized support; (iii) reduce the users' cognitive load; and (iv) try to increase cooperation with other SoS users.

First, we describe an introductory example for better comprehension of our proactive interfaces. Then, we present how to integrate the interfaces into constituent systems. Next, we show how to add proactive behaviors to them. After that, we discuss the advantages of such interfaces for SoS users.

5.1.1 Introductory example

In this section, we present an introductory example for contextualizing and explaining our proactive interfaces.

Assume a simple MBA SoS belongs to the domain of collaborative software development, and consists of four systems (see Figure 5.1) say a coding system used by developers for coding, a code quality system to analyze the quality of the code being developed, and a memory to store and retrieve knowledge. The main goal of this SoS is to improve the quality of the software being developed.

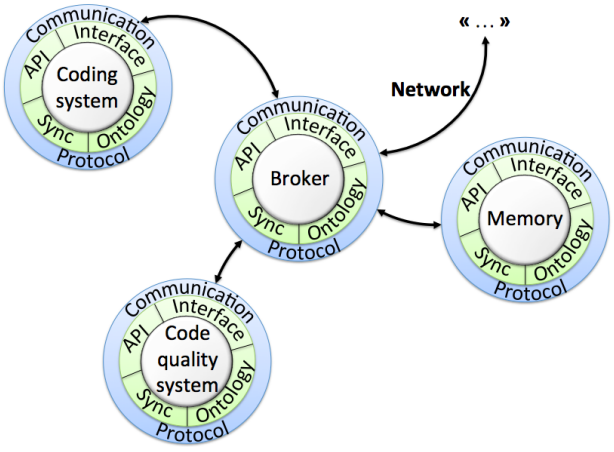


Figure 5.1: An example of MBA SoS used for introducing the proactive interfaces of this research.

In this simple SoS a developer can interact in Natural Language (NL) with the Personal Assistant (PA) of her interface, requesting functionalities (services) from other systems, for instance, to improve the quality of her code.

To give an example of the interaction between users and the PAs integrated in the interfaces of their systems, we focus on a given developer and her coding system. Table 5.1 illustrates a request made in NL by the developer to her Personal Assistant (PA). The

developer asks her PA for a functionality to analyze the quality of her code. The utterance is captured and analyzed by the PA of the interface integrated in the coding system. Next, the PA tries to find potential providers for the requested functionality, using the usual ways in the MBA architecture, i.e., work orders and brokers (see Chapter 4). Once the PA has results for the request, it warns the developer and presents the results.

Table 5.1: An example showing the interaction in natural language between a developer of software development and her personal assistant.

Interlocutor	Utterance
Personal Assistant	Hello. What Can I do for you?
Developer	Analyze the quality of my code.
Personal Assistant	OK, I'll try to do that. When I have results I'll let you know.
Developer	Thanks.
Personal Assistant	You're welcome.
Personal Assistant	I have received results for your request of quality analysis.
Personal Assistant	Please see the report below.
...	...

Now, concerning proactivity, supposing the example of the dialogue shown in Table 5.2. In this case, the developer also asks her PA to analyze the quality of her code. The utterance is then captured and analyzed the personal assistant. Then, the PA performs the request and waits for possible results. When results are available, the PA verifies that the quality of the code of its user is not going well. Then, it acts proactively to find some information that can be useful for supporting its user to improve the code quality. To find such an information, the PA attempts to fetch quality recommendations from the SoS memory, and it also tries to cooperate with the other developers' PAs in the SoS. During the cooperation, the developer's PA focuses on finding PAs belonging to developers with good code quality. Assuming the cooperation was successful, and the developer's PA has retrieved the contact of a developer with good code quality, then it synthesizes the results and sends them to its user. The message is translated in NL to warn its developer about the results.

Next, we show and detail how to integrate the interfaces in a constituent system of an MBA SoS.

5.1.2 Integrating natural language interfaces into an MBA system

In our approach, proactive interfaces are designed in an "SoS fashion," i.e., they are integrated and kept connected within constituent systems, preserving the distributed and independence features of the SoS. Currently, two approaches can be used when integrating the interfaces, named *MBA Browser* and *augmented interface*.

Table 5.2: An example showing the proactive behavior of a PA on behalf of its developer of collaborative software development.

Interlocutor	Utterance
...	...
Developer	Analyze code quality.
Personal Assistant	I got it. When I have results I warn you.
Personal Assistant	I have received results for your request of quality analysis.
Personal Assistant	There are some quality issues in your code.
Personal Assistant	I found some information that may be useful for handling such issues.
Personal Assistant	Please see the report below.
...	...

The MBA Browser is a system providing a generic interface already given by the MBA framework. Figure 5.2 shows a proactive interface integrated in an MBA system through the MBA Browser. The system consists of a PA, staff agents used for more specialized services, for instance, keeping the User Model (see *the role of ontologies* Chapter 4) of the system user, a browser interface, and an agent acting as a local server for connecting the browser with the other elements. The browser interface provides input/output channels for allowing a user to interact with her PA through NL. It has dialogue panels displaying the text of the interaction and a dynamic avatar that embodies the PA and synthesizes text into voice. Besides, as an option, the interface also provides a speech-to-text engine, for translating speech into text and allowing some vocal interaction.

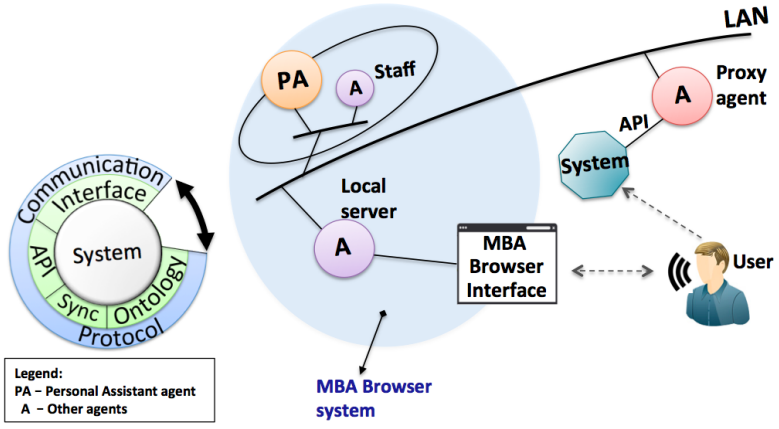


Figure 5.2: A proactive interface integrated in a constituent system of the MBA architecture by using the MBA Browser system.

To integrate the MBA Browser system with an MBA constituent, the SoS architect needs only to connect the agents (PA, staffs and local server) of the browser system to the same LAN ("Local Area Network") containing the proxy agent of the constituent (see Figure 5.2). The messages between the browser interface and the PA are handled by the local server, and the messages like requests from the PA to other SoS systems

pass through the proxy agent. Because it is a domain-dependent task, it is up to the SoS architect to develop the content of the dialogues used in the interaction between the PA and the constituent system user. However, it could be dialogues focused on the SoS functionalities the user's system can request or provide. We give examples of implementing dialogues between PAs and users in practice in the Chapter 6.

Another approach that can be used for integrating the proactive interfaces is the augmented approach (see Figure 5.3). In this case instead of the browser interface, the SoS architect is required to augment the user's system. For instance, the architect could create a specific window by using the system API, and then add some modules for the natural language interaction such as text input/output, or if also desired, couple speech-to-text and text-to-speech engines to the window for vocal interaction. After that, the architect needs to connect the interface agents, now only the PA and its staff, to the same network loop of the proxy agent of the MBA system. The messages between the PA, the augmented system and other SoS systems, are handled by the proxy agent. As with the MBA Browser, it is also up to the architect to implement the content of the dialogues between the PA and the system user.

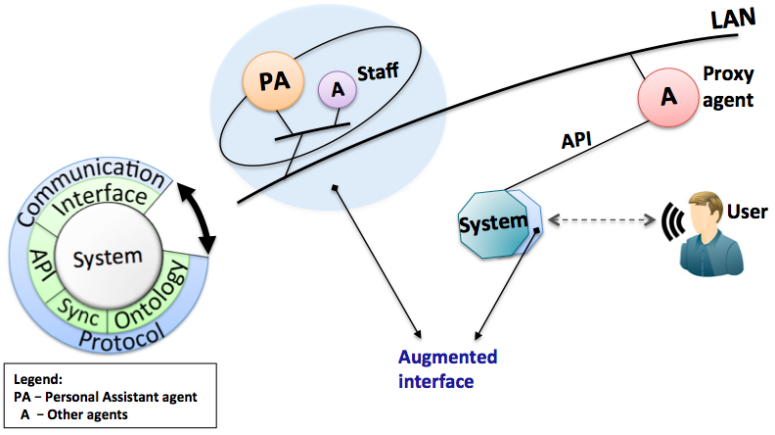


Figure 5.3: A proactive interface integrated in a constituent system of the MBA architecture by using an augmented interface.

One should note that when decoding the interactions in natural language (dialogues) in both approaches, the ontology used is the one belonging to the proxy agents, as we stated in Chapter 4.

Comparing the two approaches, one of the main advantages of using the MBA Browser as interface is that it is generic and can be used with different systems. That is, to change between domains, the architect only needs to develop the content of dialogues which is a task related to the domain of the SoS. Moreover, the MBA Browser already provides the end-user interface with required modules for allowing the user to interact through NL with her PA. On the other hand, a disadvantage can be the switching among different windows, i.e., system and browser interface, though in case of vocal interaction it could be avoided in some way like letting the window minimized. Conversely, the

augmented interface is tailored and specialized to its system. Nevertheless, such a kind of interface may require more effort and time to implement, it allows users to stay in their (constituent-) system environment, avoiding switching among different windows. Note that the MBA architecture requires that systems have an API to be part of the SoS (see Chapter 4). Thus, building augmented interfaces does not constitute an additional constraint when using our approach.

In the next section present how to add proactive behavior to the interfaces.

5.1.3 Adding proactive behavior to the interfaces

In this section we indicate how to add proactive behavior to the PAs offering the interfaces integrated in constituent systems.

In our approach, the PAs display proactive behavior by assisting their users to achieve the expected results of the SoS goal, regarding the SoS functionalities requested through their systems. To add proactive behavior to a PA, an SoS architect needs to follow the activities of a method we have developed, named *ProPA*:"Proactive Personal Assistant," shown in Figure 5.4. The ProPA has two activities, named "Create User Model" and "Act Proactively." To facilitate and provide a better comprehension of the method, we use the developer and her coding system from the introductory example (Section 5.1.1) during the descriptions of the ProPA activities.

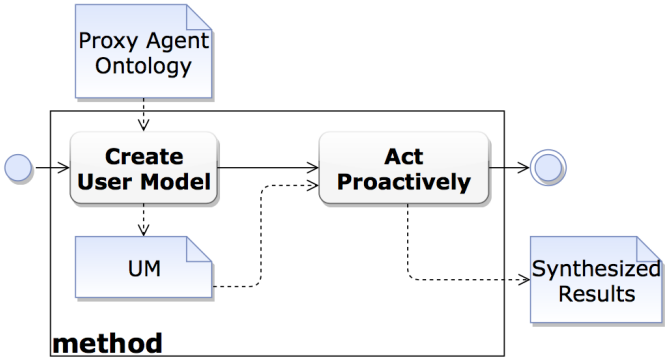


Figure 5.4: The flow of the ProPA method to add proactive behavior to the personal assistants of the proactive interfaces.

In the "Create User Model" activity, the SoS architect will take the proxy agent ontology (see *the role of ontologies* in Chapter 4) of the system in which the PA is integrated, and then will extend and customize it for deriving a User Model (UM) describing the user of the system. The purpose of doing that is to create a structure rendering it fit for use in the "Act Proactively" activity of the method. As shown in Figure 5.5, the steps that compose the "Create User Model" activity are: "Describe User," "Define User Scores," and "Define Shared Information."

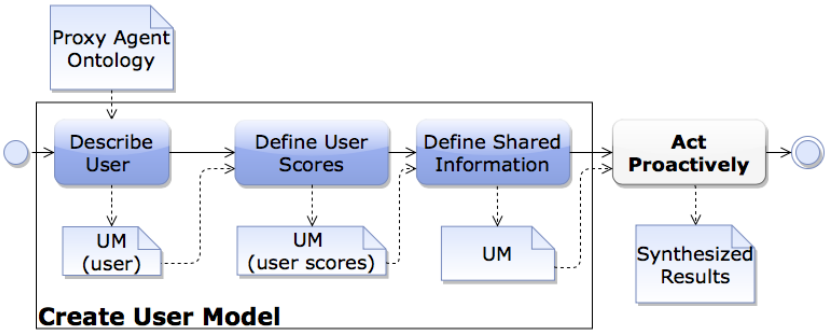


Figure 5.5: The steps of the "Create User Model" activity of the ProPA method.

"Describe User" receives the proxy agent ontology and focuses on describing the user of the system. Note that if the architect has already made it when defining the SoS global ontology (see *the role of ontologies* in Chapter 4), then she can skip this step. Otherwise, in this step, the architect is required to extend and describe the concept "User" and its attributes in the proxy agent ontology. The output of this step is the UM updated with user information. For the introductory example of the developer, the concept "User" of the UM would be extended as "Developer," being something as the one show in Figure 5.6.

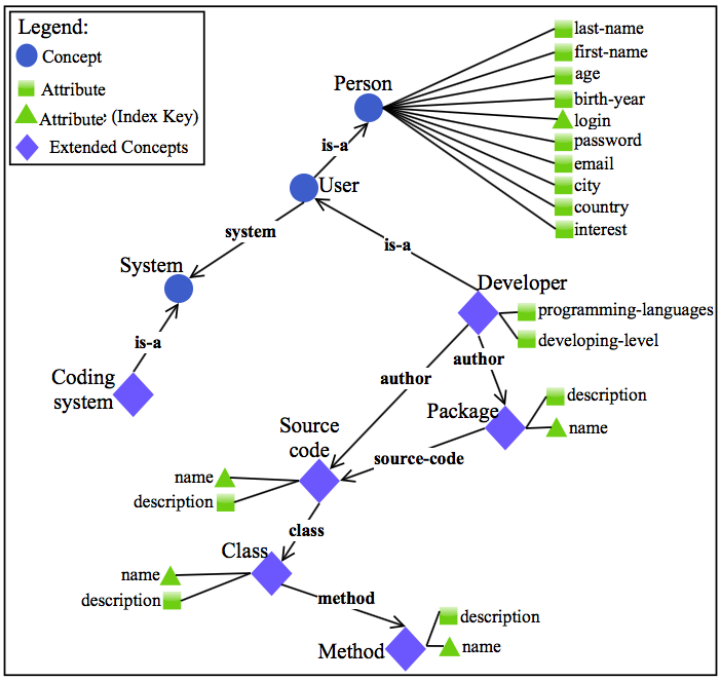


Figure 5.6: An extract of the example of the concept "Developer" extending the "User" of the proxy agent ontology.

The step "Define User Scores" receives as input the UM updated in the previous step. In this step, the SoS architect defines scores for the user regarding SoS functionalities that can be requested by her system. The goal is to define scores showing how well the user is performing tasks related to the SoS functionalities. Note that there is no limit for

the number of SoS functionalities chosen to be associated with user's scores. However, currently we recommend for the ProPA method only a single score per functionality. Moreover, there are no restrictions for the type of the scores, they can be values describing, for instance, the performance or something showing how much well the user is doing tasks related to the SoS functionalities. The importance of the scores is that later in the "Act Proactively" activity of the ProPA, they will be useful for supporting the user when requesting or receiving results for such SoS functionalities. In addition, defining the scores is done by extending and describing the concept "Score" of the UM. Note that, the concept "Score" has the attributes "low-threshold" and "high-threshold" (inherited from the OntoMBA Appendix A) which are ranges for it. The output of this step is the UM updated with the user's scores. For our introductory example, one of the SoS functionalities that the developer can request (in the case) through her coding system is *analyze code quality* (see Section 5.1.1). Therefore, a developer's score for such a functionality could be related to *quality metrics* measuring the quality of the code she is developing. Figure 5.7 shows an extract of the UM updated with such a developer's score.

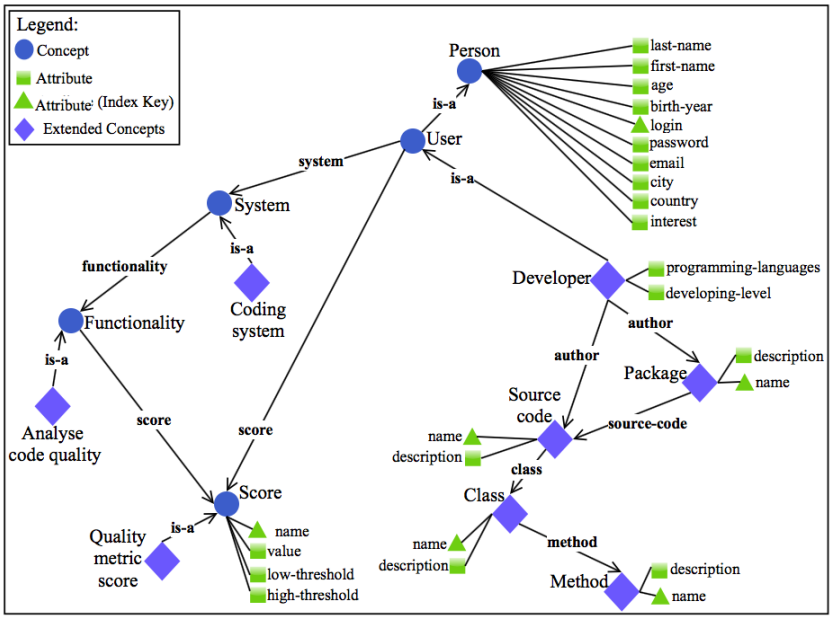


Figure 5.7: An extract of the example of the concept "Quality Metric Score" extending the "Score" of the proxy agent ontology.

The last step of the "Create User Model" activity" is "Define Shared Information," which aims to define public and shared information about the user in her UM. By using the UM from the previous step, in this step the architect describes the concept "Shared information" defining public and shared information about the user. The purpose of that is to allow the UM to provide user information that later may be useful in the "Act Proactively" activity of the ProPA. For instance, when acting proactively, the PAs can exchange information creating an intrinsic cooperation between their users for

supporting the expected results of the SoS goal. For our example, the shared information could be the developer's contact.

The output of the "Define Shared Information" step and the "Create User Model" activity is the final UM. We recommend the SoS architect to keep the UM within a staff agent of the PA (see *the role of ontologies* in Chapter 4). Figure 5.8 shows an extract of the UM created for the example of the developer and her coding system.

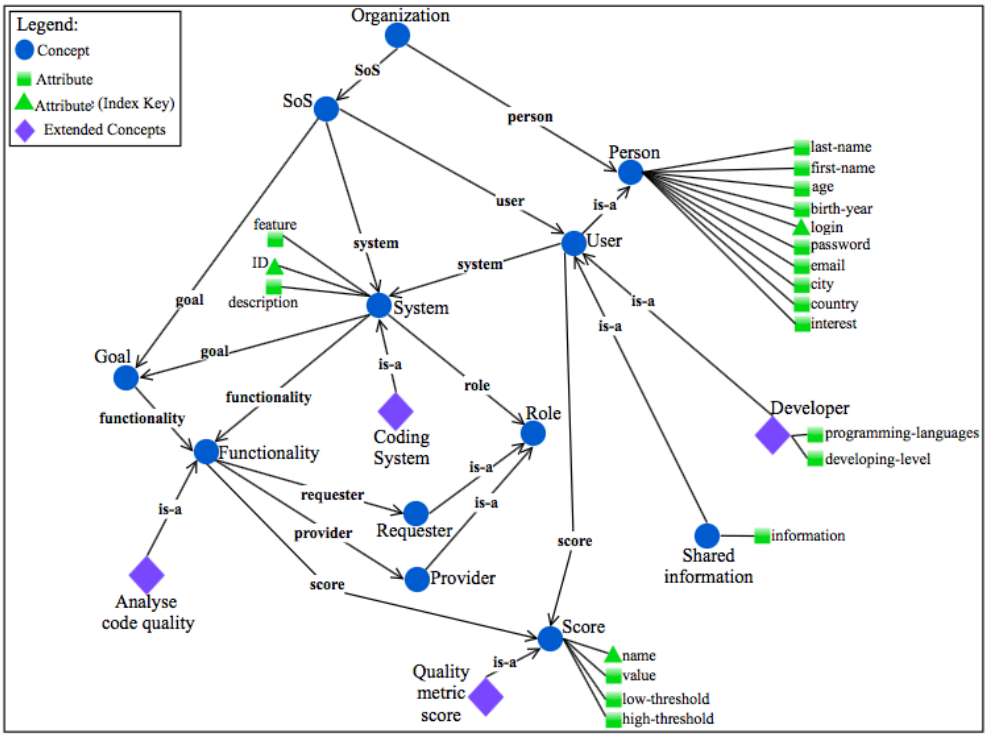


Figure 5.8: An extract of User Model created in the "Create User Model" activity of the ProPA method.

The "Act Proactively" activity of the ProPA method aims to let the PA acting proactively based on the UM created in the "Create User Model" activity of the method. As shown in Figure 5.9, the activity consists of the following steps: "Results User Request," "Update User Score," "Evaluate User Score," "Retrieve Recommendation," "Try User Cooperation," and "Synthesize Results." In each step of the activity, we describe the behavior performed by the user's PA, being up to the SoS architect to implement it according to his preferences.

In the "Receive Results Request" step, the PA receives the results for an SoS functionality requested by its user. Once the PA receives results for the request performed, it uses the UM to verify if there are user's scores associated with the functionality requested. In case negative, the PA steps out of the method and sends the results to her user. The output of this step are the results for the request performed. For instance, for the developer's example the PA would bring to its user results for the request *analyze code quality* indicating, for instance, the parts of the code with issues.

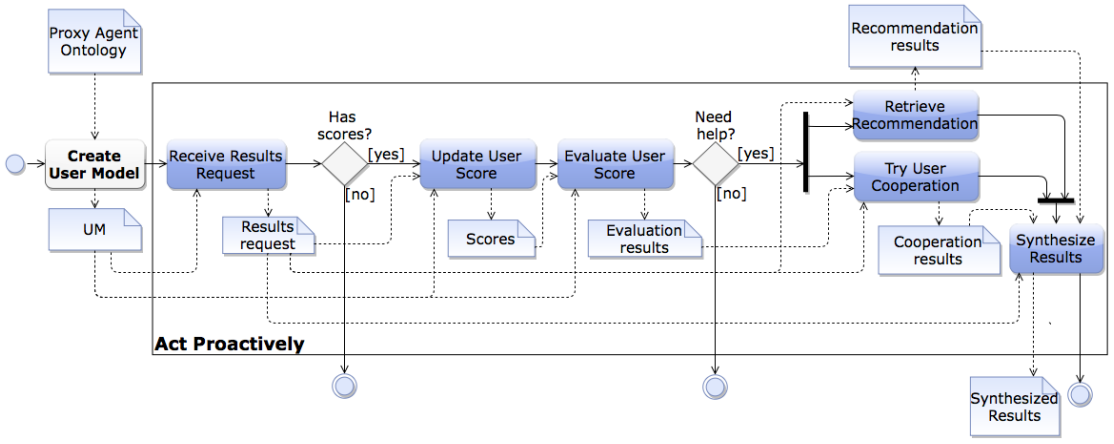


Figure 5.9: The steps of the "Act Proactively" activity of the ProPA method.

On the other hand, if the requested functionality is associated with some score, then the PA performs the step "Update User Scores." In this step the PA receives the UM, and the results for the request. Then, based on such information it updates the user's scores associated with the request made. The purpose is to let the scores up to date for being used in the next steps. The output of this step are the new scores. For the developer's example, the outputted scores would be related to quality metric values indicating how well she codes, regarding the quality of her code.

The next step is "Evaluate User Scores," which aims to evaluate the user's scores outputted in the previous step. The PA takes the new scores and applies the thresholds associated with them which were defined in the step "Define User Scores" of the "Create User Model" activity. Based on the thresholds the PA evaluates the scores and outputs the results. For instance, when evaluating the developer's scores of code quality the PA would check if they belong to certain ranges of the thresholds associated with them. After that, based on the results of the evaluation the PA verifies if its user needs some help. If the results of the evaluation indicate that the user is coding with good quality, then the PA steps out of the method and sends the results of the request to its user.

On the other hand, if the results of the evaluation indicate that further assistance would be good for the user, then the PA executes the steps "Retrieve Recommendation" and "Try User Cooperation" in parallel. Note that the PA is not obliged to execute both steps in the same interaction, it can executes either one or the other. It is up to the architect to define that. However, both of them receive as input the results for the request performed and the results for the score evaluation.

In the "Retrieve Recommendation," the PA tries to retrieve some recommendation from the SoS memories that could be useful for helping its user to improve her tasks for achieving the SoS goal. The recommendations could be found, for instance, through keywords describing the tasks. The output of this step are the recommendations. For the developer's example, supposing she had an issue related to "complexity in source

code methods," then some keywords describing the issue could be "complexity" and "method." Then, the PA could fetch from the SoS memories recommendations such as texts like "Try to create other methods and call then inside the original one," or links to external pages providing useful information related to such an issue.

In the step "Try User Cooperation" the PA aims to find useful information for supporting its user by cooperating with other SoS PAs. To do that, the PA exchanges messages with the other personal assistants in the SoS, focusing mainly on the ones belonging to users showing good values for a certain criteria, for instance, good scores when compared to the ones of its user. The PA attempts to do that by establishing Contract Nets with other PAs through work orders. The message content of such work orders could contain the information about the criteria required like the scores, and also some keywords related to the information required.

Then, the information provided by the personal assistants are retrieved from the UMs of their users, mainly regarding the concept "Shared information" (see "Define Shared Information" step in the "Create User Model" activity" of the ProPA). If no information related to the keywords is found in the UMs, then the PAs could provide standard information like the contact of their users. Note that, the information coming from other SoS users could create an intrinsic cooperation among them, besides providing support to achieve the expected results of the SoS goal. Once the PA receives information resulting from the cooperation with other SoS PAs, it outputs and sends it to the next step of the ProPA, named "Synthesize Results." For the example of the developer, the information received by her PA could be the contact of other developers in the SoS with better scores in code quality. Such information could be useful to let her in touch with other developers, for instance, to exchange ideas for improving the quality of her code.

The last step of the "Act Proactively" activity is "Synthesize Results." In this step, the PA synthesizes the results from the "Receive Results Request," the "Retrieve Recommendation" and "Try User Cooperation" steps. Finally, the PA sends the results synthesized to its user.

In the next section we discuss the advantages provided by our proactive interfaces when used by the users of constituent systems.

5.1.4 Discussion

Although they are optional, the proactive interfaces provided by the MBA framework can support users of constituent systems in several ways. The main advantages provided by the interfaces are: (i) proactive support for helping users cooperation with the SoS and achieve the expected results of its goal; (ii) providing customized support; (iii) reducing the users' cognitive load; and (iv) trying to increase cooperation with other SoS users.

First, the PA of an interface can act proactively for supporting its user regarding tasks related to the SoS functionalities requested through her system. Based on the results received from a request made by its user, the PA verifies if the user needs some assistance in the tasks related to the request. When acting proactively, the PA exchanges information with other constituent systems for supporting its user regarding tasks related to the SoS functionalities, and thus improving cooperation in the SoS for achieving its goal.

Moreover, each PA of the proactive interfaces is a butler for its user, focusing on her tasks and behaviors. Thus, each user in an MBA SoS receives customized support from her PA, independently from the system being used. The support is based on the UMs specially tailored for each user. By using the interfaces, each SoS user can focus specifically on her issues to improve them for achieving the expected results of the SoS goal.

Furthermore, the proactive interfaces may increase cooperation among SoS users. For instance, when acting proactively the PAs try to cooperate with each other for supporting their users to improve tasks related to the SoS functionalities. This creates a kind of intrinsic cooperation among SoS users, as the PAs exchange shared information coming from the UM of their users. Increasing the cooperation among SoS users may facilitate the SoS to achieve its goal.

As we stated in the Chapter 2, SoS users suffer from switching context and excessive cognitive loads because of the evolutionary nature of SoS. Usually, the user interfaces in SoS are rigid following the "WIMP" (Windows, icons, menus, pointer) paradigm (see Chapter 2). The proactive interfaces of the MBA framework provide interaction in Natural Language (NL) to SoS users. The interaction between users of constituent systems and the SoS is standardized through NL, thus keeping always the same medium of interaction, independent of the constituent system. Moreover, by using NL interfaces users express their requests as they do when interacting with other people, meaning that they do not need to memorize, for instance, new sequences of menus, buttons or commands, each time the SoS changes. Thus, the MBA proactive interfaces can reduce the SoS users' cognitive load when compared to the usual approaches.

However, to make the proactive interfaces available to users, first the SoS architect needs to integrate them into the constituent systems. It may require some effort, depending on the way it is done. Currently, we provide two manners of integrating the interfaces (see Section 5.1.2), named *MBA Browser* and *augmented interface*. The MBA Browser is generic and already provides an end-user interface through browser, however it creates a new window for SoS users and their systems. On the other hand, the augmented interface requires the architect to augment the user's system, by creating, for instance, a window within it. Although, this approach may require more effort to be implemented,

it allows the users to stay in their (constituent-) system environments. Besides, it is up to the architect to implement in the interfaces the dialogues between a PA and its user.

Finally, our approach provides the ProPA ("Proactive Personal Assistant") method for adding proactive behavior to the interfaces. The method is generic and to change between domains the architect needs to derive a User Model from the proxy agent ontology, customizing it according to the user of the system. Note that, implementing the proactive behavior is not obliged by our approach, meaning that SoS users can simple have NL interfaces with PAs for interacting with the SoS.

With all that in mind, perhaps one may still think what are the differences between our Personal Assistants and the embodied agents used in other fields like virtual reality. First, the Personal Assistant agents of our approach are digital butlers (see Chapter 3) focusing on assisting human beings, i.e., their masters (in the case SoS users). Moreover, the PAs are cognitive and use their knowledge (UM) for providing personalized support to their masters. Furthermore, in our approach the natural language interaction between SoS users and PAs is much more simple, as in SoS usually the interactions involve requests for SoS functionalities for achieving the expected results of the SoS goal (we go further in this in the Chapter 6). Conversely, usually embodied agents of virtual reality take a more social and psychological perspective, approaching aspects such as emotions and trust (Lhommet et al. (2012); Callebert et al. (2016)). Furthermore, the conversations also tend to be longer (Stein and Ohler, 2017) in order to exploit such an aspects.

5.2 Synchronization among constituent systems

For supporting SoS robustness, the MBA framework provides: (i) loose coupling among the constituent systems of an SoS, thus reducing interdependencies; and (ii) it offers a functionality for synchronizing constituents during SoS operation, aiming to support them when they go down for letting the SoS in a coherent state. In this section, we highlight the main points of the synchronization provided by our approach. For the sake of readability we detail it in the Appendix B.

Providing the synchronization is important as once constituent systems are reconnected to the SoS their state may not be anymore consistent with the rest of the SoS, and thus information provided by them can be incoherent and spread over all the SoS. With this in mind, we need to state that our synchronization is intended to be used by systems that change their internal states when cooperating in an SoS. For instance, a Database (DB) could be an example of such a kind of system as it inserts or removes data, changing its internal state.

Our approach for synchronization is based on *Extended Abstract Statements* (ExAS) which are Abstract Statements (Chapter 4) extended by a label for providing a unique identifier for each one of them. To illustrate, after reconnecting a DB relying on the synchronization to its SoS, the proxy agent of the DB exchanges automatically work orders with other constituents to retrieve the ExAS related to the period the DB was down. The ExAS received by the proxy agent are used to put the internal state of the DB in coherence with the SoS, regarding the information exchanged during the period it was disconnected. This process is made through a method we call *SyncMBA*.

One of the main advantages of our synchronization is that it does not rely on checkpointing or rollback-recovery mechanisms for letting systems in a coherent state. For instance, when approaches rely on those kind of mechanisms, systems are restored back to a certain stable point of in time, and thus information is lost. Besides loosing information, systems go to a past state with not to date information, and thus one more step needs to be done, perhaps manually, by the SoS architect for letting them coherent to the current state of the SoS. In the MBA synchronization, constituent systems are synchronized automatically from the point they stopped receiving information capable of changing their internal state, until the newest information exchanged in the SoS capable of doing that. Thus, our approach goes towards the present, meaning that, information is not lost, and only a single step is performed automatically (by the proxy agents interfacing the systems).

Conversely, the main drawback of our synchronization may be summarized by the fact that it makes the assumption that *for each system intended to be synchronized in an SoS, there is at least one other system in such an SoS offering the same SoS functionalities it does*. However, if in an SoS there is only a single system offering its functionalities and then it goes down, depending on the role and importance of such a system, the entire SoS may break down. In fact, it all boils down to the SoS application, but we need to highlight that in the case of our synchronization, it is currently in its early stages and further improvements may be needed as we continue testing it.

In addition, one may find in Appendix B a method called *MeTRo*:"Method for Testing Robustness," aiming to test the robustness of MBA SoS. The MeTRo intends to test the MBA robustness based on the *synchronization* functionality and the *loose coupling* feature provided by our approach. Note that, currently the MeTRo is a proposition, and we have not had yet the opportunity to test it. We plan to do that in the near future.

5.3 Summary

In this chapter we presented extended functionalities provided by the MBA architecture. Currently, our approach provides extended functionalities focused on supporting SoS users and robustness. For supporting SoS users, the MBA architecture provides proactive

interfaces, based on personal assistants that conduct exchanges using natural language in a textual or vocal mode. Personal assistants can be proactive and make use of user profiles to better customize interactions. Conversely, besides providing loose coupling among constituent systems, our approach supports SoS robustness with a functionality for synchronization. When constituents go down, they can be synchronized during operation for letting the SoS in a coherent state.

Chapter 6

The ACE4SD System of Systems

This chapter describes a functional prototype, named ACE4SD (Advanced Collaborative Environment for Software Development), which was the first SoS we built in this research. The role of the ACE4SD SoS was paramount in our research. As we stated in the Chapter 1, we used it as a basis for performing experiments and tests which then culminated in designing and improving our MBA architecture, as well as supporting us to move it towards a framework. The SoS was built in the domain of collaborative software development for improving code quality, and it also aimed to validate and show the feasibility of our approach.

Following, we describe the context and the problem statement motivating the development of the ACE4SD SoS. Then, we show the SoS detailing its constituent systems. After that, we show the dynamic of the SoS through examples of work orders exchanged among the ACE4SD systems. Next, we describe the implementation in practice, the challenges faced and how we have overcome them. We finish the chapter with a general discussion about the SoS.

6.1 Context and problem statement

In this section we describe the context of collaborative software development and the motivation for developing the ACE4SD SoS.

Developing software is often done by teams of developers under the supervision of a manager. To produce their codes team members use several different and independent systems such as coding systems, databases, and code quality analysis systems. The final product created through such systems results from the integration of the code written by each developer. Such pieces of code can be of different quality, which has an impact on the whole project.

According to a study conducted at the University of Cambridge (Cambridge University, 2013), the annual cost of correcting source code defects is around US$312 billions. A code of low quality impacts the delivery date, needs repairs and rework that become the major cost drivers of the project. Moreover, low quality reduces customer satisfaction, can affect market share, and in some cases can even lead to criminal charges. On the other hand, a software of high quality reduces the need for repairs and rework, sometimes by more than 50%. Furthermore, the costs linked to the maintenance and support of the application are also reduced. Having a high quality software also improves testing and delivery schedules as mentioned by Bettenburg and Hassan (2013) and Bonsignour and Jones (2011).

However, because the systems used by team members are not integrated, it is difficult to use all of them at the same time and exchange information between systems and participants. Thus, to support the role of developers and managers to improve the quality of their code, we have developed the ACE4SD SoS connecting systems and tools used during collaborative software development.

Next, we show and detail the ACE4SD and its constituent systems.

6.2 The SoS and its constituent systems

Figure 6.1 shows the ACE4SD SoS which aims to support developers and managers to improve the quality of the code developed. The SoS consists of coding systems, a code analysis system, a versioning system, different kinds of DataBases (DB) such as relational, object, key-value, and triplestore, Manager information systems, and a Web search system.

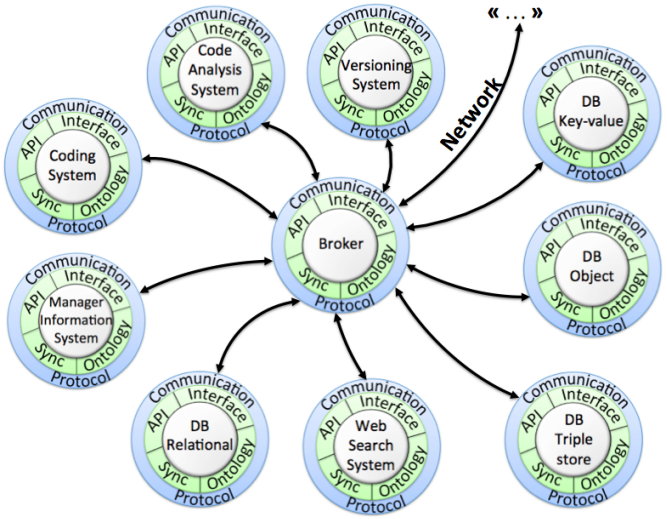


Figure 6.1: The ACE4SD SoS developed with the MBA architecture for the domain of collaborative software development.

The coding systems are used by developers to produce their codes. The goal of a coding system is to provide the environment for developers writing and developing their code with quality. The system is augmented with an MBA proactive vocal interface of the type augmented (see Chapter 5). Thus, developers can interact with Personal Assistants for receiving customized support and improve the quality of their code. Moreover, they can also request their PAs for exchanging messages with other developers or managers. The number of coding systems varies according to the number of developers on a collaborative software development team.

The versioning system intends to store and manage globally the different versions of code produced by teams. The goal is to allow team members to organize, control and access the versions of code developed collaboratively.

The code quality analysis system analyzes the quality of the code produced by teams. The system measures and evaluates the code quality, and then outputs values indicating it.

Another systems of the ACE4SD are the databases which are the memories of the SoS. The goal of the DBs is to capitalize and provide means for knowledge management in the SoS. For instance, the memories may capitalize knowledge concerning the quality analysis made by the analysis system, or quality recommendations. The ACE4SD has four different kinds of DBs, named relational, object, key-value and triplestore. Thus, it offers the opportunity for team members to store and retrieve knowledge in a larger number of formats, which may be useful, for instance, to handle legacy systems and increase efficiency.

The Manager information is a system to support the role of the managers, letting them receiving information about the quality of the code of their teams. As developers, managers have also Personal Assistants interacting and supporting them with the quality information. Through their PAs, managers can also send messages and communicate with developers and other managers.

The Web search is a system capable of realizing personalized searches in the Web. The system aims to support the SoS by finding useful information for team members to improve the quality of their codes. Examples of such information may be quality recommendations, or links to external pages proving useful information about code quality.

Table 6.1 shows the SoS functionalities provided by the constituent systems of the ACE4SD focusing on the SoS cooperation to improve code quality. Note that, we focus on the functionalities used for supporting the expected results of the SoS goal.

To show the dynamic of the ACE4SD to achieve the goal of improving code quality, next we provide examples of work orders exchanged among the constituent systems.

Table 6.1: The SoS functionalities provided by constituent systems of the ACE4SD focusing on the SoS goal of improving code quality.

		Systems								
		Coding system	Versioning system	Code analysis	DB relational	DB object	DB key-value	DB triplestore	Manager information	Web Search
Functionalities	To produce code	X								
	To allow team members request/receive SoS information	X							X	
	To allow communication among team members	X							X	
	To store the code versions produced by teams		X							
	To retrieve the code versions produced by teams		X							
	To measure code quality			X						
	To analyze code quality			X						
	To store SoS knowledge such as quality analysis or quality recommendations				X	X	X	X		
	To retrieve SoS knowledge such as quality analysis or quality recommendations				X	X	X	X		
	To perform personalized searches in the Web									X

6.3 Examples of work orders

In this section, we give examples of the interaction among the constituent systems of the ACE4SD SoS. Our examples show work orders exchanged between users and systems to achieve the goal of improving code quality. Note that, in the figures below showing the examples we have not shown the interactions exchanged with the broker element of the SoS. The reason was to facilitate the visualization and readability of the figures. Despite that, all the work orders sent by systems pass through brokers in the SoS to find providers and transfer results back to requesters.

In the example shown in the sequence diagram of Figure 6.2 the developer request his PA for quality recommendations. Then, the broker receives the work order and broadcasts it (see Chapter 4) into the SoS. The broker found the databases offering the recommendations and ask them to allocate the tasks. When a database finishes its tasks, it send the recommendations to the broker. Then, the broker takes the first answer coming, in the case from the DB Object, and transfers it back to the developer's PA to present the recommendations to her. At the same time, the Web search system performs searches in the Web for information to improve code quality. When the search system has found some information, it sends work orders to store such a knowledge that in this case will be realized by the DBs.

Figure 6.3 shows a sequence diagram representing the proactive behavior of a PA on behalf of its developer. The "Developer-1" sends a request in Natural Language (NL)

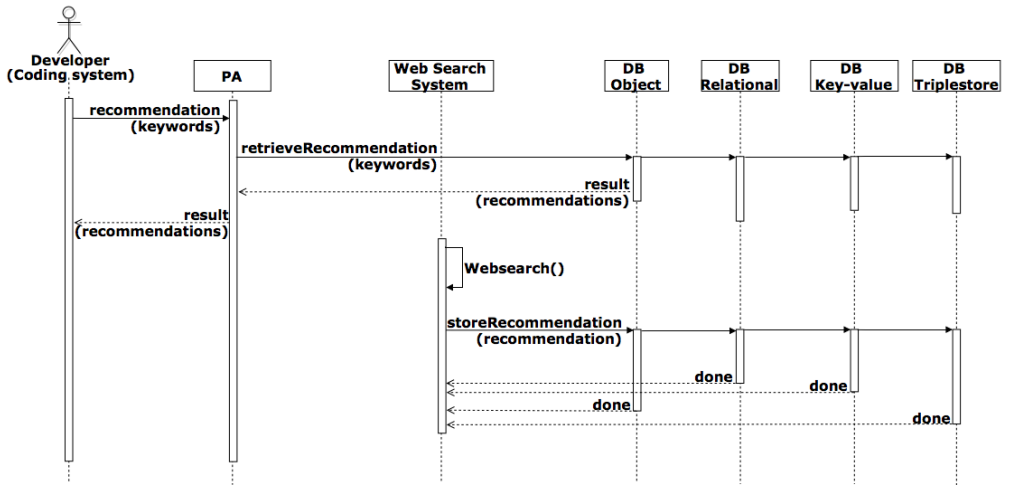


Figure 6.2: A sequence diagram showing a request made by a developer for receiving quality recommendations in the ACE4SD SoS.

through her coding system for analyzing the quality of her code. When the PA integrated in the system receives the request, it analyzes the utterance and then sends a work order to a broker for retrieving the code of its developer. The broker locates the versioning system and allocates the tasks. Once it has the code, it transfers it back to the PA. Then, the PA sends a work order for analyzing the quality of the code. Now, the broker finds the quality analyzer system, and then allocates the required tasks. When results for the analysis are available the broker transfers then back to the PA. In the meantime, the quality analyzer performs requests to store the results of the code analysis. The broker finds the database systems which then store the information.

After receiving the results, the PA acts proactively verifying if its user needs some help. First, the PA checks that the "Developer-1" has scores for the functionality of analyzing code quality, and then it requests its staff agent to update such scores. Next, the PA asks its staff to evaluate the scores according to code quality thresholds. Assuming that the results received were not good, then the personal assistant sends a work order requesting recommendations based on keywords related to the quality scores. After that the PA receives results for the recommendations which in this case were fetched first from the DB key-value. In addition, the personal assistant also sends a work order looking for the developers holding the best scores compared to the ones of its master. The PA do that by sending a work order containing a contract net parameter (see Chapter 4). In this example, the "Developer-2" is the one with the best scores, and then her PA receives the request and asks its staff for retrieving its User Model (UM). The "PA-Dev-2" checks if there is some shared or public information in the UM that could be useful for the requester, based on the information (keywords) received. Supposing the UM does not has any specific information, then the "PA-Dev-2" sends a general information like a contact of its master. Finally, the PA of the "Developer-1" receives the results and synthesizes all the information to present to its master, i.e., the results for the code

analysis, the recommendations from the DBs, and the contact from the cooperation with other developers/PAs.

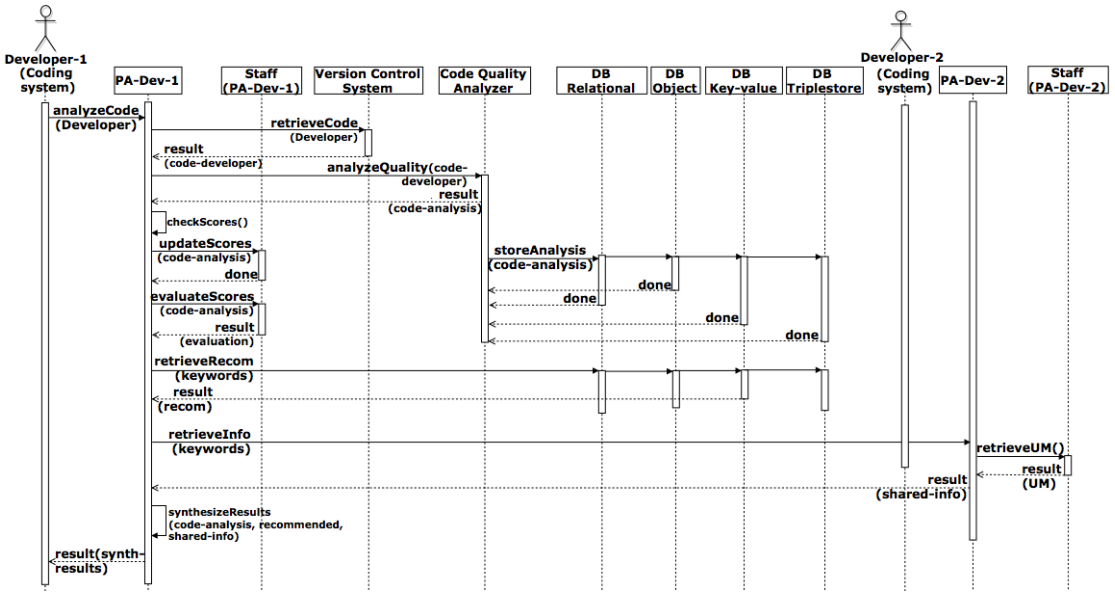


Figure 6.3: A sequence diagram showing the proactive behavior of a PA when a developer requests a code quality analysis in the ACE4SD SoS.

Meanwhile the developer is interacting with the ACE4SD, a manager of software development sends a work order for a Web search (see Figure 6.4) regarding information about a specific quality issue, for instance, related to the coupling of code packages. The manager's PA retrieves her User Model (UM) and sends it with the request to the broker. The broker receives the work order and finds the Web search system offering the service. The system uses the manager's UM for providing a personalized Web search focused on her interests. When it has results, the Web system sends them to the broker which in turn transfers them to the manager's PA to shown them to her. After that, the manager decides to send a message to the developer, for instance, regarding the information for the package quality issue she has received. The manager's PA sends the work order to the broker that in turn transfers it directly to the developer's PA. The developer receives the message which is presented by her PA.

Next, we describe how we have implemented the ACE4SD SoS in practice, the main challenges faced and how we have overcome them.

6.4 ACE4SD implementation

In this section we detail the implementation of the ACE4SD SoS. We show the main challenges faced when implementing the SoS and the solutions we have used to address them.

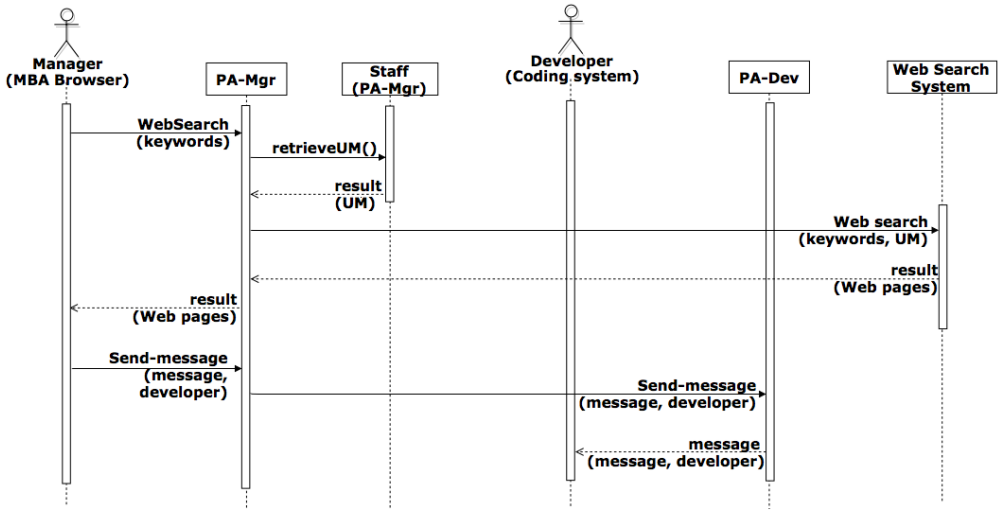


Figure 6.4: A sequence diagram showing a request made by a manager for code quality information, and interaction with other team members in the ACE4SD SoS.

Following, we detail first how we have interfaced constituent systems with proxy agents. Then, we discuss about two approaches we have implemented for the SoS broker, named Real Broker and Virtual broker. After that, we present the implementation of the proactive interfaces for SoS users, and then the dialogues used in such interfaces. Finally, we discuss how we handled the network in the SoS through coteries.

6.4.1 Interfacing systems with proxy agents

In the implementation of the ACE4SD we focused on improving the quality of Java code. We have used several different systems to implement the SoS.

- The coding system used by each developer was EclipA[2] (details in Appendix E) which is the Eclipse IDE[1] augmented with some modules like plugins, proactive interfaces and vocal applications.
- The versioning system was GitHub[2].
- The analysis system was the CoQAS (details in Appendix C) which is a multi-agent system applying rules to compute the quality of Java code.
- The relational database was MySQL[3].
- The object database was AllegroCache[4].
- The key-value database was Redis[5].

[1]See `https://www.eclipse.org` for more information.
[2]See `https://github.com/` for more information.
[3]See `https://www.mysql.com/` for more information.
[4]See `https://franz.com/products/allegrocache/` for more information.
[5]See `https://redis.io/` for more information.

- The Web search system was the PeWeS[2] (details in Appendix D) which realizes personalized Web searches on the top of well-known engines like Bing[6].
- The Manager information system used by managers was the MBA Browser (details in Appendix F).
- The triplestore was actually MEMORAe[7] (Abel, 2015), a knowledge management system developed in our laboratory, handling documents viewed as resources. MEMORAe is a knowledge platform capable of capitalizing knowledge and skills in the context of organizations. In this platform, resources and knowledge are indexed using a semantic map (an ontology) visualized as concepts and instances.
- The MAS environment was provided by OMAS[8] (Barthès, 2011) which is a platform developed in our laboratory. The platform is developed in Common Lisp and offers a rich environment for developing multi-agent applications. OMAS agents are cognitive, persistent, multi-threaded; they have skills (what they can do) and goals (what they plan to do). Each agent can have its own ontology to understand the expressions (vocabulary) of the message content language, to build a knowledge base, or to interpret (decode) utterances from users.

We have used different types of agents provided by the OMAS platform, named Service Agent (SA), Transfer Agent or postman (XA), and Personal Assistant (PA). SAs provide a particular type of service corresponding to specific skills. XA are gateways that communicate with platforms having different structures, implementing their own protocol, and translating communication and content languages. PAs are in charge of interfacing humans to the system, they follow the metaphor of "digital butler" described in the Chapter 3.

The proxy agents were instances of the OMAS Transfer Agent (XA). We took an XA and created a stub within it for connecting systems in the MBA architecture. The goal was to standardize the proxy agent and facilitate the work of SoS architects, allowing the stub to be reused and extended by each constituent system in the SoS. The proxy agent already provides means for connecting systems through different types of network standards such as TCP (Transmission Control Protocol), UDP (User Datagram Protocol) and HTTP (Hyper Text Transfer Protocol). Because the OMAS platform is written in Common Lisp, the proxy agent also provides functions for translating JSON data into Common Lisp lists (and vice versa) to facilitate the data manipulation within the SoS. Moreover, the proxy agent is composed by predefined skills (what it can do) for receiving and handling the work orders requesting functionalities of the system it interfaces, and to translate the requests of the system into work orders to send them to the SoS brokers.

[6] See `https://www.bing.com` for more information.
[7] See `http://memorae.hds.utc.fr/demo/labo/` for more information.
[8] See `http://www.utc.fr/~barthes/OMAS/` for full documentation.

Concerning the format of the message content for exchanging information among proxy agents and systems, and also among the other elements in the SoS, we have adopted a very easy and simple one which was the JSON[9] standard. Note that, for some systems we could connect them with proxy agents directly using Common Lisp, meaning that in such a case the format of the message content was Lisp Lists. However, in the lists we kept a JSON fashion through property-value relations.

As we have stated in Chapter 4, systems must provide at least an API to be interfaced with proxy agents. When implementing the ACE4SD we have used some different approaches for connecting systems with proxy agents. Figure 6.5 shows a UML deployment diagram representing the different approaches (*a) Network Standards*, *b) Lisp API* and *c) Agent Protocol*) we used to connect systems and proxy agents.

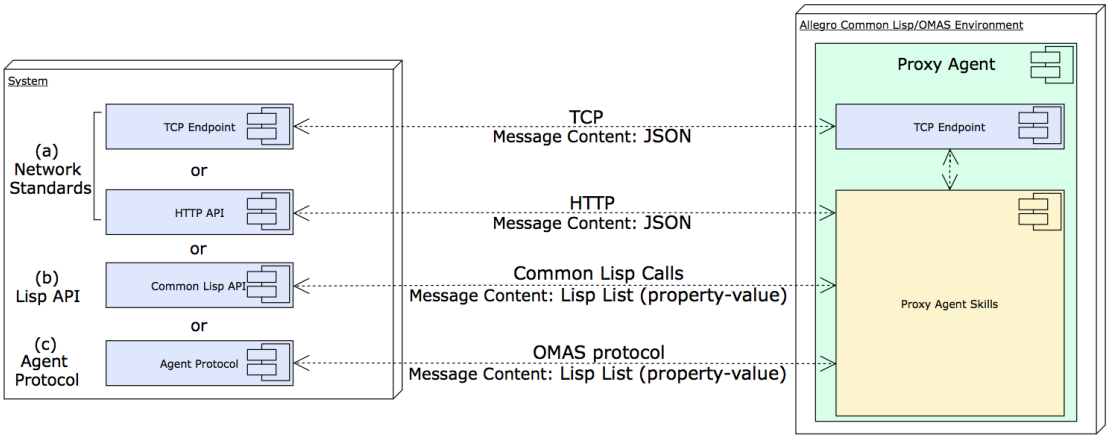


Figure 6.5: The different approaches, namely Network Standards, Lisp API and Agent Protocol, used for connecting systems and proxy agents.

The first approach (*a) Network Standards* in Figure 6.5) was to connect systems and proxy agents through network standards such as TCP or HTTP. For instance, for the EclipA^2 system we have used TCP creating endpoints in both sides system and proxy agent. In the EclipA^2, we added a plugin in the Eclipse IDE, making it proper for sending and receiving information through TCP connections. The MEMORAe platform is another example we have used such network standards. The platform provided an API in HTTP with message content in JSON, thus we have called its functionalities through such an API directly from the skills of the proxy agent.

The second approach (*b) Lisp API* in Figure 6.5) we used for connecting systems and proxy agents was realized directly through APIs available in Common Lisp. Once a system was reachable through a Lisp API the proxy agent was able to access and call its functionalities directly from its skills. In the ACE4SD, we have connected the MySQL, the Redis and the AllegroCache by using this approach.

[9]More information in: `http://www.json.org`

The last approach (*c) Agent Protocol* in Figure 6.5) we used for connecting systems and proxy agents was by using agent protocols. For instance, we interfaced the agent-based systems CoQAS, PeWeS^2 and MBA Browser with their proxy agents by using the agent protocol of the OMAS platform.

It is important to highlight that to facilitate the connection and also keep the SoS distributed and independent nature, in all the approaches above we have put a given proxy agent with its system in the same LAN (Local Area Network).

In our prototype, we have implemented ontologies through the MOSS[10] framework, which stores data as objects and is part of the OMAS platform. The first ontology we implemented was the ACE4SD global ontology. We took the OntoMBA core ontology (Appendix A) and derived the SoS global one, modeling its domain by specializing concepts such as "Organization," "SoS," "Goal," "System," or "User." Figure 6.6 shows an extract of the ACE4SD global ontology. Such ontology was one of the ontologies we kept within the proxy agents of the memories, i.e., AllegroCache, MySQL, Redis, and MEMORAe.

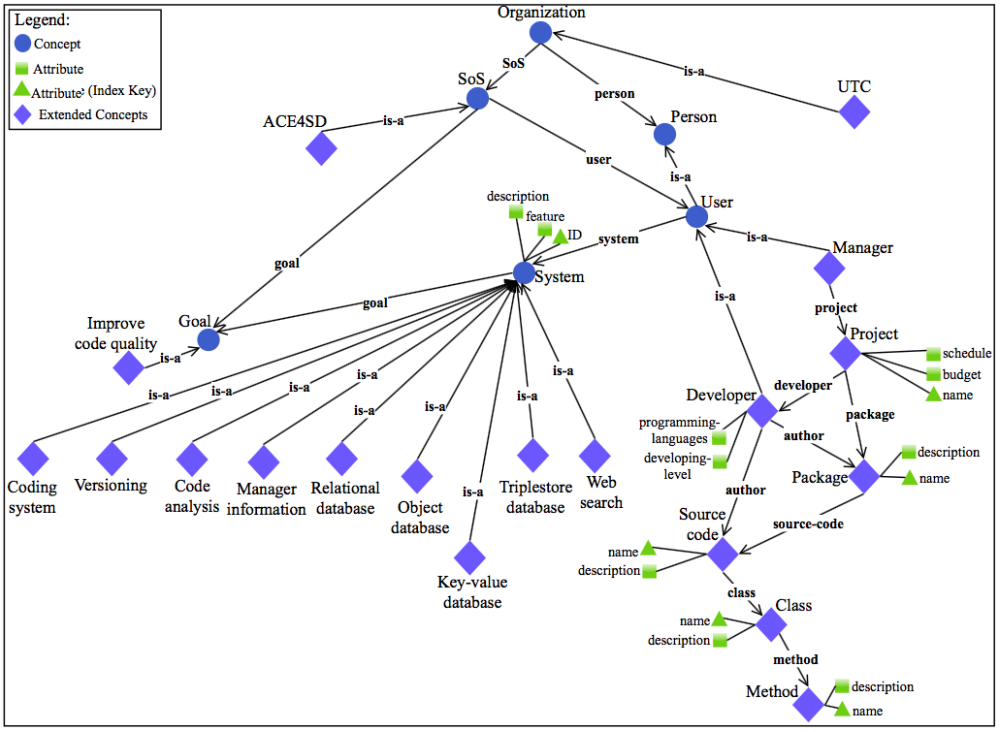


Figure 6.6: An extract of the ACE4SD global ontology.

After that, for each proxy agent we added an ontology derived from the SoS global one, by adding concepts related to the *SoS functionalities* of the system it interfaced could provide or request. The goal was to match the inter-agent message structure to the idiosyncrasies of each element system. For example, for the MySQL database we defined

[10] See `http://www.utc.fr/~barthes/MOSS/` for more information.

it provides functionalities based on SQL statements for storing and retrieving information about the ACE4SD domain such as code quality analysis or recommendations. The proxy agent ontology supported the matching of the message structures with the ones of MySQL relational tables. Figure 6.7 shows an extract of the proxy agent ontology of the MySQL DB.

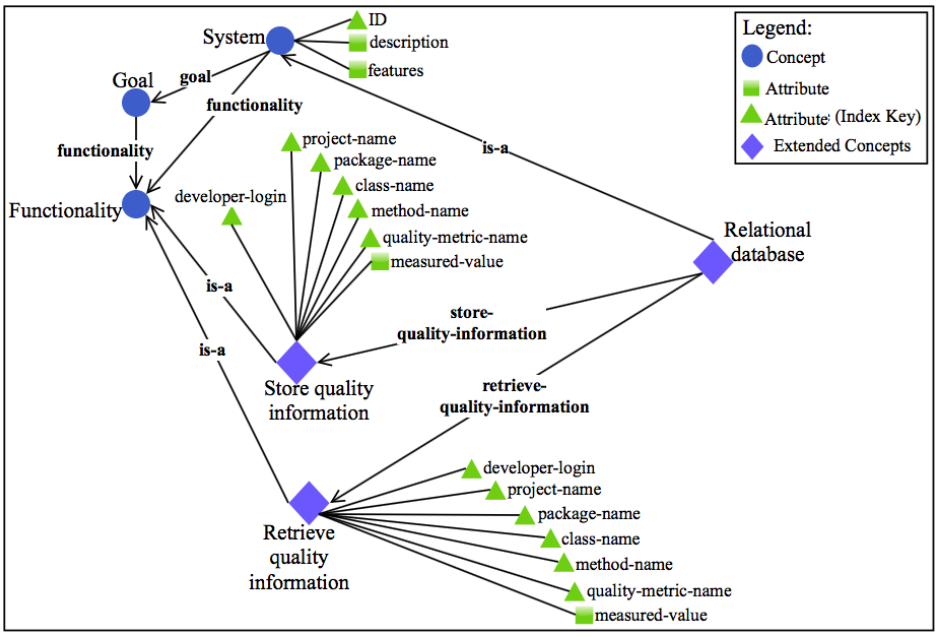


Figure 6.7: An extract of the proxy agent ontology of the relational database (MySQL) of the ACE4SD.

For each functionality a constituent system could request or provide we developed a skill in its proxy agent to perform and execute the tasks needed. We could say that the proxy agent skills acted as a "bridge" among the system functionalities (provided or requested) and the SoS. For example, in the proxy agent of the code quality analysis system we created a skill to handle requests for storing quality analysis results. Such a skill contained, for instance, some calls for formatting results or calls to send work orders. The very simple skill of this example is shown in Common Lisp as follows:

```
(defskill :STORE-ANALYSIS :CODE-ANALYSIS-SYSTEM
  :static-fcn static-store-analysis
  )

(defun static-store-analysis (agent message &key data)
  (let (code-analysis)
     ;; formatting code analysis
     (setq code-analysis (format-code-data data))

     ;; sending a work order for storing the analysis
     (send-subtask agent :to :BROKER
                         :from :CODE-ANALYSIS-SYSTEM
                         :action :WORK-ORDER-BOX
```

```
                        :args `(:data (("query" "store-analysis")
                                       ,code-analysis))
                        :timeout 5)
  (static-exit agent :DONE)))
```

6.4.2 Real broker vs. Virtual broker

When we began to develop the ACE4SD, we started by using only a single broker agent, i.e., a *real broker*, in the SoS. Such an agent was a Service Agent (SA) of the OMAS platform. The broker agent had skills for receiving work orders and send them through broadcasts or Contract Nets, depending on the parameters provided by requesters. After that, it transferred the results back to the callers. We implemented the work orders in the ACE4SD by putting them on the top of the MAS protocol provided by the OMAS platform. The OMAS implements several communication protocols such as simple, broadcast and Contract Net. The OMAS agent communication language is simple and not very original, resembling KQML with the exception of a cancel-grant performative that allows granting contracts to a set of agents while at the same time cancelling the task for the other ones. This reduces racing conditions. Moreover, the choice of the protocol is done at the message level. An example of work order implemented on the top of the OMAS protocol is shown in Common Lisp as follows:

```
(send-subtask agent :to :ALL :action :store-analysis
                             :args `(("code-quality-analysis" (...)))
                             :timeout 4 :protocol :contract-net
                             :strategy :collect-answers)
```

In the work order above, the broker broadcasts a request for a functionality to store code quality analysis through a contract-net with a strategy of collect-answers, i.e., the broker waits the timeout for some answer (details in Chapter 4).

However, the approach of the single real broker may have some issues. For instance, It is well-known that centralized elements are brittle and make the approach more vulnerable. Among the issues that may occur one can mention complete failures or bottlenecks, as we have discussed in Chapter 2.

Thus, we decided to replace the single agent by a broker MAS to handle the requests between constituent systems. The goal was to remove the centralized single element and make the approach more robust. With a broker MAS, the real broker could be distributed and thus we could remove the single point of failure. Despite that, by adding more broker agents the complexity was also augmented and more resources such as processes, memory and network were required.

Then, after experimenting and testing our prototype we discovered that we could replace the broker MAS by using conditional addressing allowing to deliver messages only to agents that satisfy certain conditions formulated using the sender ontology and the ontology query structure. The OMAS platform is quite efficient with broadcast messages or Contract Net messages since the call for bids can be done with a single message. This approach, defining a *virtual broker*, preserves the P2P nature of the exchanges among agents. For instance, when a system, say the code quality analyzer, sends a work order using conditional addressing it takes a feature from the its ontology describing the provider(s) of the functionality being requested. Once the work order is sent, there is a reasoning and the systems with such a feature and capable of providing the requested functionality have the belief of that to start to perform the tasks. The mechanism for the conditional addressing and the reasoning are handled by the OMAS and MOSS. An example of work order using conditional addressing is shown in Common Lisp as follows:

```
(send-subtask agent :to '(:_cond ("provider" ("feature" :is "database")))
                        :action :store-analysis
                        :args `(("code-quality-analysis" (...)))
                        :timeout 4 :protocol :contract-net
                        :strategy :collect-answers)
```

Now, in the work order using conditional addressing shown above the own code analyzer sends the request and waits for the results. When the work order is thrown the systems reason to known which on is capable of performing the tasks. The systems that have the feature of "database" are the ones able to provide the functionality and then they may start to perform the tasks. When results are available, providers send them back to the requesters. Note that, the ontologies of the requesters must be at least compatible with the ontologies of the providers, regarding the features and functionalities used in the work order.

By comparing the two approaches of real and virtual brokers we can highlight the main advantages and disadvantages of each one. The main advantage of using a real broker (MAS) is that the systems do not need to align or update their ontologies regarding the features among them. On the other hand, one needs to implement the broker MAS physically, the complexity increases with the number of broker agents, and it demands more computational resources. For the virtual broker, although it requires some alignments in the ontologies of constituent systems, the main advantages are that one does not need to implement it physically, the messages are treated only by the systems able to provide the requested functionalities, and it keeps a pure P2P among constituent systems.

Here, we need to highlight that after realizing we could use a virtual broker in the SoS, we updated the stub of proxy agent we developed by adding predefined skills to handle requests (sending/receiving) using conditional addressing. Our goal was the same we

did when developing the stub, i.e., to facilitate the development by letting it available to be reused, but now through a virtual broker approach.

6.4.3 Integrating natural language interfaces

In the ACE4SD SoS we developed natural language interfaces for developers and managers of collaborative software development. We used the two approaches described in the Chapter 5 to integrate the interfaces, named augmented and MBA Browser.

The first kind of interface we integrated in the constituent systems of the SoS was based on an augmented approach. We applied such an approach for developing the interface of the EclipA^2 system used by developers for coding. Figure 6.8 shows a UML deployment diagram for the integration of the interface in the EclipA^2. To develop the interface we augmented the Eclipse IDE component of the EclipA^2 by extending the plugin we have created (see Section 6.4.1) with an Eclipse "view." The plugin was developed in Java and provides input/output in text for allowing the interaction in natural language between developers and their PAs. There was also a button for allowing users to interact vocally with their PAs through speech-to-text and text-to-speech engines. We implemented the vocal interaction by creating a simple application in C# that uses the Microsoft System.Speech API[11] for handling both engines. Then, we connected such an application to the Eclipse plugin by using a simple TCP connection. The result was that users could interact vocally with their PAs by clicking on such a button.

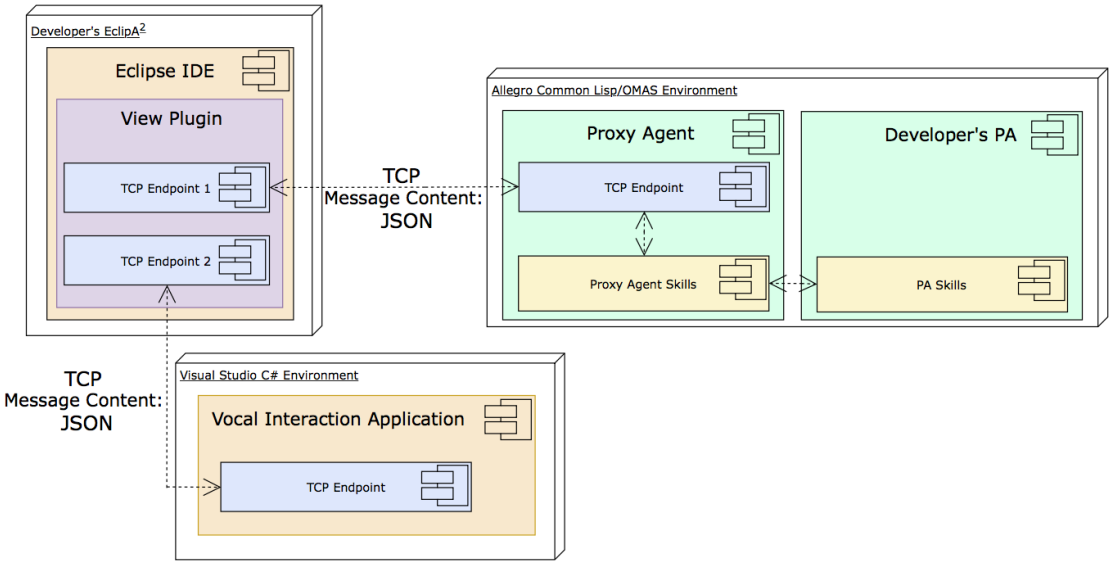


Figure 6.8: Integrating a natural language interface in the EclipA^2 through an augmented approach.

After building the plugin, we added the PA and its staff agent in the same LAN of the proxy agent interfacing the system. The connection between the augmented interface

[11] See `https://msdn.microsoft.com/en-us/library/office/hh361625` for more information.

and the PA was intermediated by the proxy agent in both senses. Then, we created a skill in the PA for handling the messages arrived from and sent to the proxy agent which resulted from the interaction with the user. Concerning the staff agent, we used an SA provided by the OMAS platform, and it had an ontology implemented with MOSS that represented the User Model (UM) of the developer. The UM was derived and extended from the proxy agent ontology (Section 6.4.1) of the system. To developed the model, we followed the steps described in the "Create User Model" activity of the ProPA method Chapter 5.

Figure 6.9 shows the augmented interface we created in the Eclipse IDE for the developers of the ACE4SD. One may see in the figure the plugin (a window *view*) showing the communication in Natural Language (NL) between a developer and her PA. In the example of the figure a developer asks her PA in NL for a code quality analysis. Note that, while the developer is working on her code, she can interact with the SoS through her PA, staying in the same (constituent-) environment of her coding system.

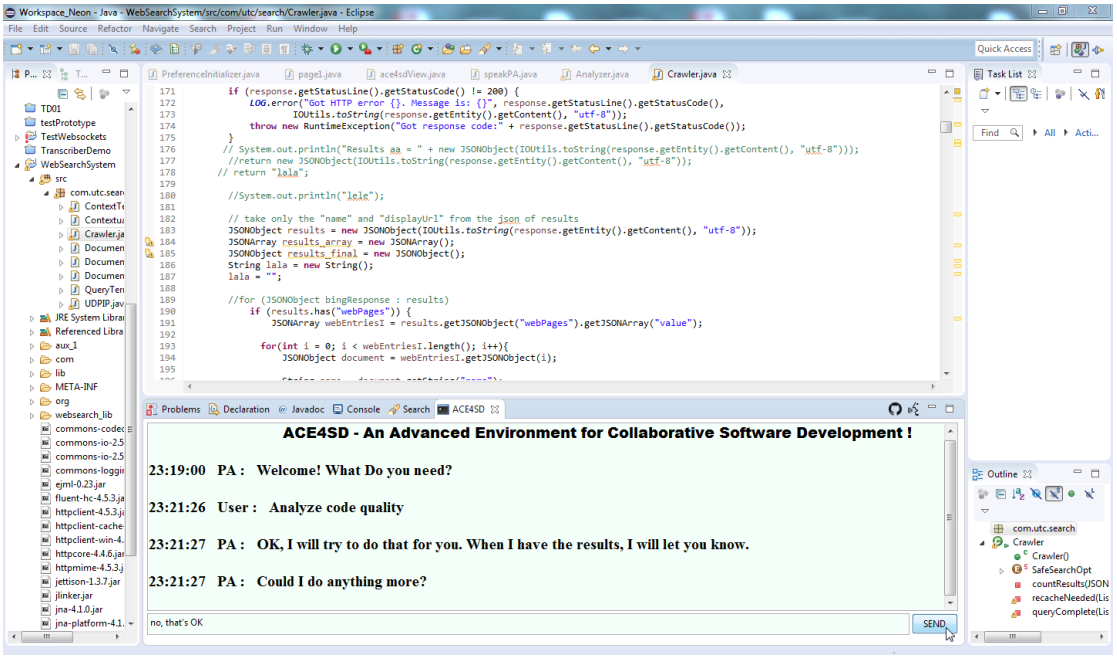


Figure 6.9: The augmented interface integrated in the Eclipse IDE component of the EclipA[2], and which was used by developers of the ACE4SD.

The second kind of interface we implemented in the ACE4SD was through the MBA Browser system. We used the approach for interfacing the role of managers in the SoS. Figure 6.10 shows a UML deployment diagram for the integration of the MBA Browser interface. To implement the browser interface of the approach we used HTML5 and JavaScript to create and control the browser components. In the interface, we developed dialogue panels for allowing managers to interact in natural language with their PAs. Moreover, we coupled to it speech-to-text and text-to-speech engines using

the Google Web Speech API[12] for allowing vocal interaction. In addition, we added a dynamic avatar to the interface for embodying the PA and making the interaction more friendly. The avatar was implemented through the Botlibre[13] platform and was capable of synthesizing text.

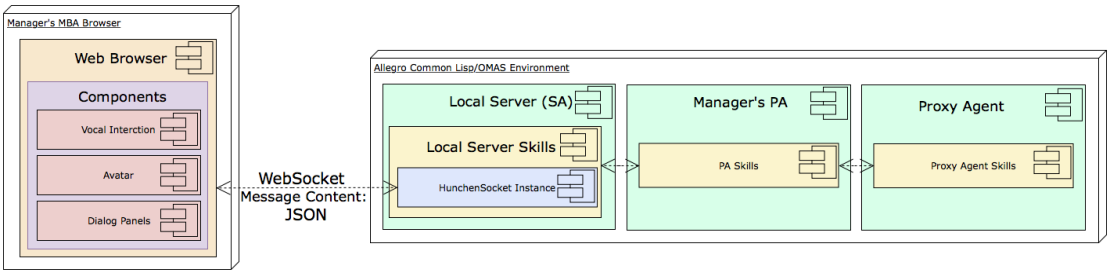


Figure 6.10: Integrating a natural language interface through the MBA Browser.

The browser interface of the system was connected to a local server handled by an SA that in turn was connected to the same LAN of the PA and its staff agent. The SA of the local server made use of the HunchenSocket[14] which relies on the Hunchentoot[15] Web server for creating a WebSockets server in Common Lisp, and then providing a full-duplex communication over a single TCP connection between the browser and the PA. After that, we created a skill in the PA for handling the messages arrived from and sent to the SA of the local server, resulted from the interaction with the user. Moreover, we also defined a skill for treating the messages regarding requests from the PA to other SoS systems. Note that this kind of message pass through the proxy agent interfacing the MBA Browser system.

Figure 6.11 shows the interface of the MBA Browser used by managers of the ACE4SD. In the example of the figure a manager asks her PA for a chart showing an overview of code quality in a project. The PA brought a chart categorizing the project quality into code classes, methods and packages.

Through the implementation of the two approaches described above we could learn some lessons regarding the difficulties and the advantages of each one. We discuss them below.

Developing the augmented interface has required much more time and effort. Creating the plugin inside the Eclipse IDE has posed some issues. For instance, choosing a component for handling the input/output channels in the plugin "view" was not so trivial. After some experiments, we finished by adopting a "StyledText" component as a text I/O channels for supporting the natural language interaction between developers and PAs. Moreover, there is some complexity behind the plugin taking care of the connections with the vocal application, and the proxy agent of the system. However, the main benefit brought by such an approach was that developers remained comfortable

[12] See `https://www.google.com/intl/en/chrome/demos/speech.html` for more information.
[13] See `https://www.botlibre.com` for more information.
[14] See `https://github.com/joaotavora/hunchensocket` for more information.
[15] See `https://edicl.github.io/hunchentoot/` for more information.

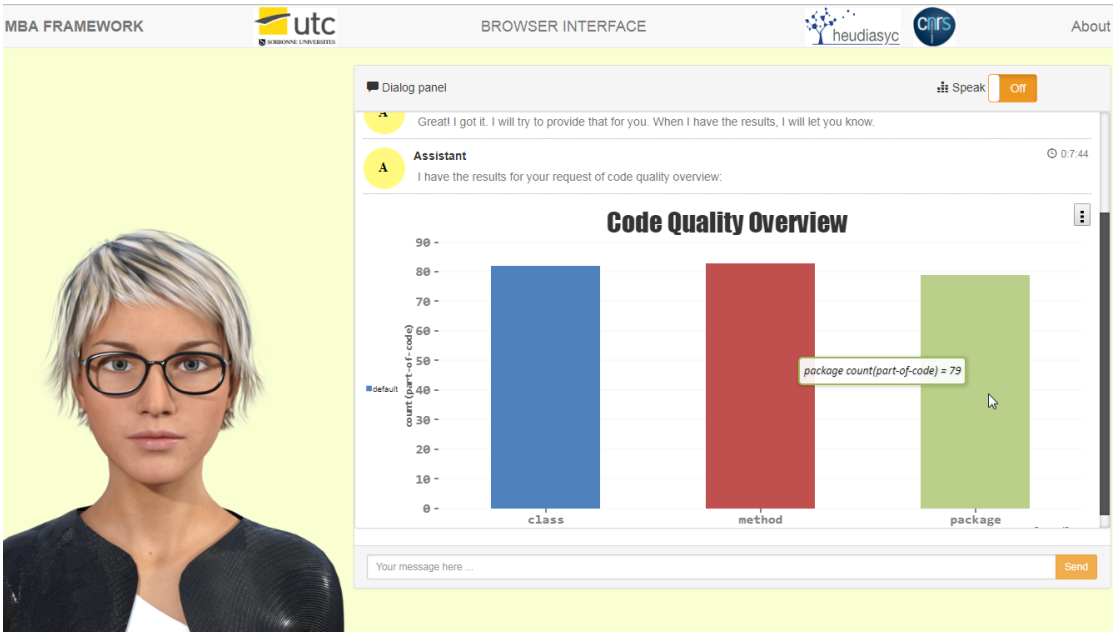


Figure 6.11: The interface of MBA Browser used by managers of the ACE4SD.

in their (constituent-) system environment, coding as usual and now also interacting with the SoS.

The MBA Browser was easier to implement for being a natural language interface of the SoS. The browser interface was developed very easily mostly by using a high level code in HTML5 and calling APIs through JavaScript. The client for the WebSockets was written in few lines of code as it is already provided by HTML5. The main required by the MBA Browser approach was to develop the local server of the SA. We made tests with different libraries to find one compatible with the MAS environment.

Although it was not our main focus, during the development of the prototype, we have also tried to couple vocal interaction in both natural language interfaces. The goal was to allow users to interact vocally with the SoS at the same time they keep doing their tasks in their systems.

For the augmented approach and more specifically in the case of the Eclipse plugin, we have tried to use the Microsoft System.Speech API as it allows desktop speech recognition in offline mode. We intended not to require the use of the Internet when developers were coding, as they can do that offline with the Eclipse IDE. However, such a recognition approach suffers of some issues. For example, the accuracy of the recognition is not naturally very good, and for being improved it is dependent on a training phase within the Windows operational system realized by each different prospective user. Another option, we tried for improving the recognition was by using constrained grammars. Although in this case, the recognition was acceptably, it required time and effort to prepare and develop the grammars and their rules.

On the other hand, for the MBA Browser we used an online API that works well without requiring predefined training from system users. Although the good recognition, the weakness of the approach was that it required Internet connection to work, as the information is sent to the Web for recognition.

Next, we detail the implementation of the dialogues used by the Personal Assistants to interact with their users.

6.4.4 Dialogues

The natural language interaction between developers or managers, and their personal assistants was done very easily. The reason is because usually in an SoS the interactions between systems and users involve mainly performing requests for functionalities, and then transferring back results. When the user made a request, her personal assistant tried to select a possible task from a library of tasks. The library consisted of tasks related to the SoS functionalities the system could request or provide. The vocabulary used for decoding the user's utterances and finding the task functionalities was described in the proxy agent ontology of the system.

Once the PA found a task, a dialogue associated with it was triggered in order to acquire missing information necessary to execute such a task. When enough information was gathered, the PA sent the request associated with the task to find providers in the SoS. Once providers were localized, they performed the task and sent the results to the PA that in turn presented them to the user.

To implement the dialogues of the natural language interfaces we used the task-oriented dialogue manager provided by the OMAS platform. The OMAS dialogue manager fits the situation described above, and thus we considered it appropriate for handling the dialogues of our PAs. First, the approach uses a very general top-level dialogue that listens to user's requests and tries to find suitable tasks based on a library of tasks. For instance, if a developer has typed or spoke the request "Analyze the quality of my code" to her PA, then it will takes such an utterance and will segment it into a list of words. After that, for each task in the library, the PA checks the sentence for phrases specified in the index pattern describing the task, and computes a score by using a MYCIN-like formula (i.e. if 2 cues a and b are present, the combined score is computed by the formula $a + b - a \times b$). For example, a library task for analyzing the code quality, with its linguistics cues (*:indexes* parameter) which are terms from the proxy agent ontology, and the dialogue associated with it, can be represented as follows:

```
(deftask "analyze code quality"
  :doc "Analyze the quality of the developer's code."
  :performative :request
  :dialogue _analyze-code-quality
```

```
  :indexes ("quality" .5 "code" .3 "source" .2)
  )
```

For the request of code quality above, the words "code" and "quality" will give a score of $(0.5 + 0.3 - 0.5 \times 0.3) = 0.65$ for the task. After that, tasks are then ordered by decreasing scores, and the task with the highest score is selected, and all the tasks with a score under a specific threshold are discarded.

Once the task is selected, the dialogue associated with it is triggered and executed. To handle task dialogues the OMAS manager uses conversation graphs (Barthès, 2013) in which a node represents a state of the conversation. According to the information acquired from the user, a transition occurs to another state until some action state is reached or the request is abandoned.

Figure 6.12 shows the very simple "Analyze code quality" conversation graph. First, the developer asks her PA for a code quality analysis. Then, the personal assistant sends a work order to retrieve the code, for instance, from a versioning system. After that, the PA sends a work order for analyzing the quality of the code. If it receives results for the analysis, then the dialogue is finished successfully. Otherwise, it enters in a new state, named "Sorry," for example, to warns the developer about the failure and then exits the dialogue.

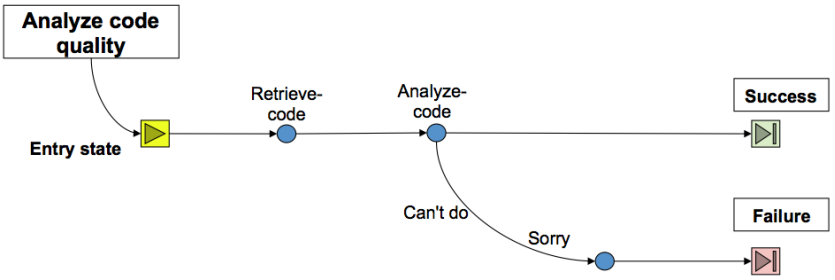


Figure 6.12: Conversation graph implementing a dialogue for code quality analysis.

The following expression in Common Lisp shows the "Analyze-code" state of the conversation graph for the example of code quality analysis above.

```
(defstate
    _analyzecq
    (:entry-state _analyze-code-quality)
  (:label "Request code quality analysis.")
  (:explanation "The PA requests an analysis for the code
                 quality of its master.")
  (:send-message :to :PA_JOHN
                 :from :PA_JOHN
                 :action :analyze-code
                 :args `(((:data ,(get-fact :developer-code))
                          (:pattern
```

```
                          '("Project" ("package") ("source-code")
                                      ("class") ("method") ("score")))
                        (:language :en)))
                :timeout 5)
  (:transitions
   (:on-failure :target _analysis-sorry)
   (:otherwise :exec (send moss::conversation '=display-text
                       (read-fact moss::conversation :answer))
               :success)
  )
 )
```

Here, the PA of the developer "JOHN" sends a message to itself to the skill "analyze-code," containing the developer's code and a pattern specifying the format desirable for the results. The skill is responsible for handling the tasks of sending work orders for finding providers to analyze the code, and to send the results back to the PA dialogue. Then, in the case results for the code analysis were successfully received by the PA in his dialogue, then it sends them to the developer, otherwise it goes to the "Sorry" state. We need to highlight that as we did in this example, for each dialogue of the PAs of our interfaces, we created a specific skill for handling the tasks needed, for instance, to request or receive results from SoS functionalities.

Developing dialogues for the personal assistants is not an easy task. Although, in an SoS the context is usually well-defined through the specification of goals, systems or functionalities, the development of the task-oriented dialogues requires a certain effort from the SoS architect. Task libraries and dialogues need to be defined and designed. Moreover, the complexity of the approach depends on some factors, for example, the number of functionalities requested by the user, and the level of interaction with the PA in each dialogue. However, the use of a dialogue manager as we did in the implementation of our dialogues may reduce the efforts and make the task less arduous.

Next, we detail how we have implemented key aspects when adding proactive behavior for the PAs of the natural language interfaces.

6.4.5 Proactive behavior

In the Section 6.4.3, we have stated that the UM kept in the staff agents of PAs was implemented by following the "Create User Model" activity of the ProPA method Chapter 5. Now, in this section we give details of how we implemented in our prototype some key aspects of the "Act Proactively" activity of the ProPA such as related to the users' scores and the attempt for user cooperation.

In the ACE4SD, when a developer requests a code quality analysis her PA tries to find providers for such a request, and then once tasks are allocated it waits for the results.

Once the analysis results are available the PA checks that the analysis functionality is associated with developer's scores, and then it performs an analysis for supporting its master. This is part of the "Act Proactively" activity of the ProPA method (see Chapter 5). In our prototype, the developers' scores for the code quality analysis functionality are based on software quality metrics.

When the CoQAS receives a code, it performs its analyzes based on several different software quality metrics (see Appendix C). Each quality metric measures a specific quality attribute, for example, coupling, cohesion or cyclomatic complexity of the code. After measuring the code, the CoQAS calculates the average value of the values measured for each software quality metric as shown in Equation 6.1.

$$metric_{avg} = \frac{1}{n} \sum_{i=1}^{n} metricValue_i \tag{6.1}$$

Then, when the CoQAS sends the analysis results which contain the average values above, the PA updates the code quality scores of its user in her UM. For the ACE4SD SoS we adopted as an example, that the developers' scores are averages for the quality metrics, based on the current scores and the new metric averages computed with the Equation 6.1. That is, each metric score of a given developer is calculated by Equation 6.2 as shown as follows.

$$metric_{score} = \frac{CurrentMetric_{score} + metric_{avg}}{2} \tag{6.2}$$

After that, following the steps of the "Act Proactively" activity of the ProPA method, the PA performs an evaluation for each metric score to verify if its developer needs some help. Depending on the results of its evaluation, the PA may try to support its user proactively. For the ACE4SD, we adopted that developers with metric scores below well-studied quality metric thresholds (Martin (1994); Filó et al. (2015)) receive proactive support from their PAs. Then, in such cases the PAs took some actions such as retrieving quality recommendations from the SoS databases and by trying to cooperate with other PAs, as we showed in examples of the ACE4SD work orders (see Figure 6.3 in the Section 6.3).

To retrieve recommendations from the databases for a quality issue related to a metric, we simple matched keywords describing such an issue with keywords associated to recommendations stored in the DBs. For instance, for an issue related to "deep inheritance tree of classes" we described it with the keywords "inheritance," "class," "deep." Then, in the DBs we retrieved the following recommendation "A deep inheritance tree of classes

means that it may inherit a lot of methods and attributes that can make it complex to maintain. Try to reduce the inheritance deep of this class." In the Figure 6.13, after analyzing the code quality of its developer, a PA brought to her a quality report with recommendations for the issues.

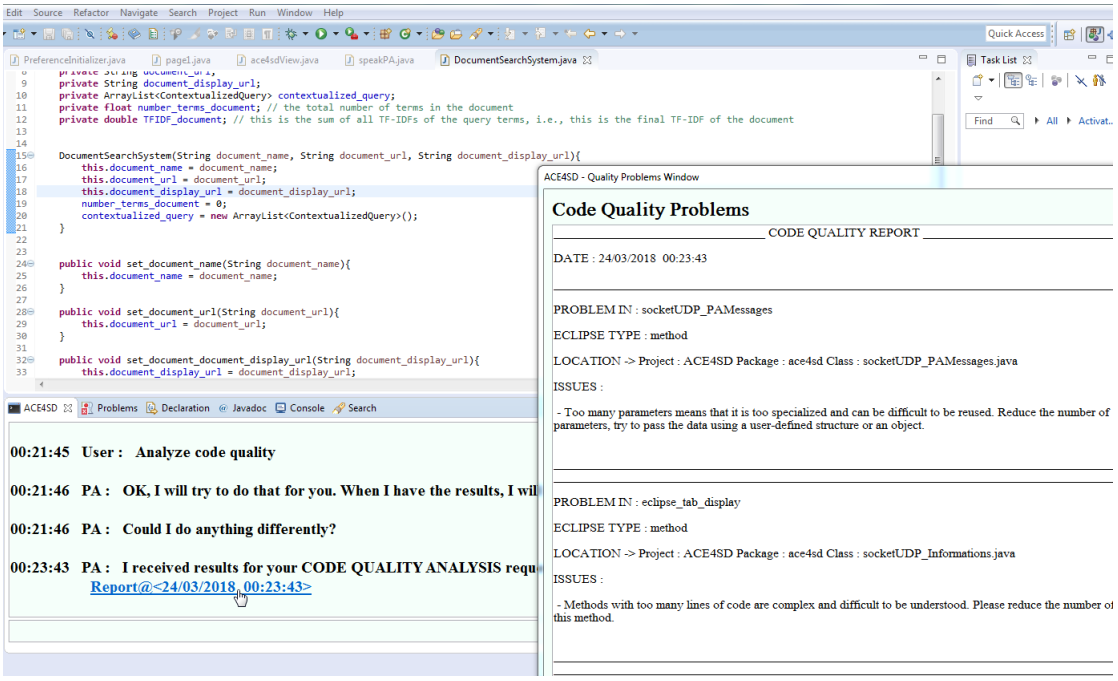


Figure 6.13: A PA acting proactively in the ACE4SD by bringing quality recommendations to her developer.

For the cooperation among PAs, we implemented it in our prototype by trying to form Contract Nets through work orders. When the developer's PA knows that its user needs support with the quality of her code, it sends a work order for requesting the code quality scores of the other developers, by setting the contract-net strategy parameter, for instance, as "collect-answers" (see Chapter 4). Then, the PA waits for possible answers. In the ACE4SD, we assumed that the quality score is free information to be shared, thus it is up to the other developers' PAs to answer or not the call-for-bids of the Contract Net. In timeout, the PA takes all answers received and selects the one (s) with the best scores in quality issue(s) of its master.

Then, the personal assistant asks the PA that has answered with the best scores for some helpful information related to the quality issues. The developer's PA with the best scores receives the request with some keywords describing the information needed. For instance, if the issue was related to cyclomatic complexity, then examples of keywords used were "path," "graph" or "McCabe." We defined such keywords arbitrarily, but taking into account their relation with the issue. After the PA of the developer with the best scores receives the request, it checks the UM of its master, looking for shared information about that. Such an information is added by the developers and can be, for instance, recommendations, libraries or links to Web pages. If there is no special

information related to the issues, a general public information from the developer is fetched, in the case of ACE4SD a contact. Figure 6.14 shows the proactivity of a PA to its developer in the ACE4SD by bringing her a contact after cooperating with other PAs in the SoS.

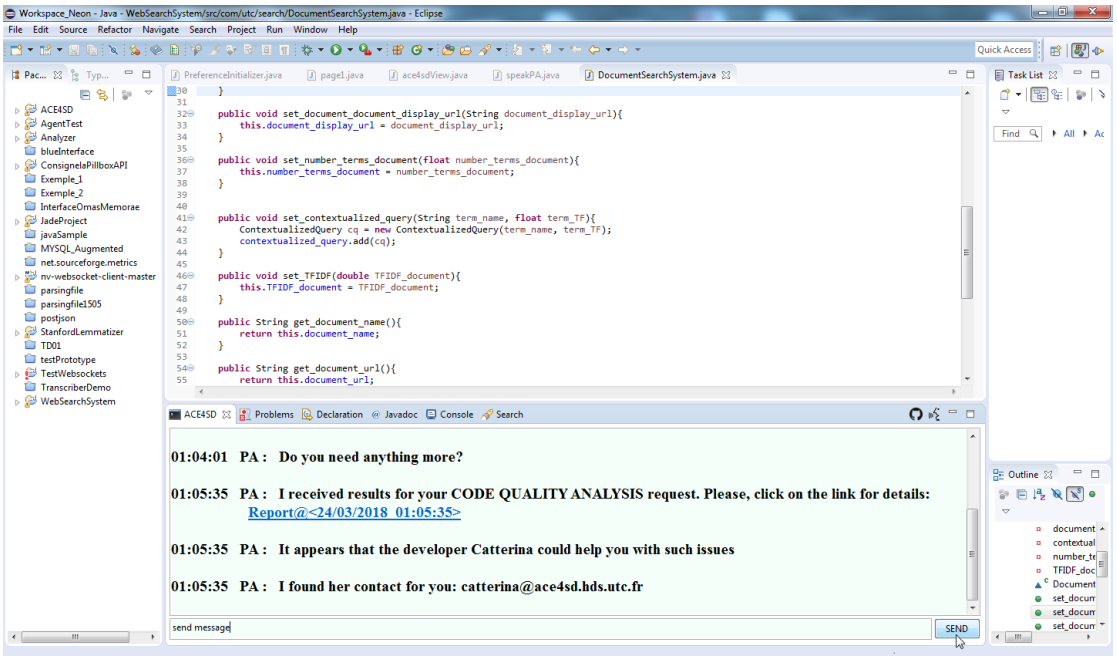


Figure 6.14: A PA acting proactively in the ACE4SD by bringing a contact to its developer after a cooperation with other PAs in the SoS.

With all that in mind, note that the proactivity of our PAs relies on the ProPA method which requires the definition and implementation of functionality scores. Such scores vary according to the functionalities requested or provided, domain or application of the SoS. Thus, if the SoS architect wishes to add proactive behavior to the PAs of the interfaces integrated into the systems, he needs to define such scores according to his requirements. The key point is to analyze if implementing the proactivity to the natural language interfaces will be worth to the SoS needs.

6.4.6 Memory interface

In the ACE4SD SoS we have used four distinct kinds of memories, named relational database (MySQL), key-value database (Redis), object database (AllegroCache), and triplestore database (MEMORAe).

We started by implementing the proxy agent of the AllegroCache database, because it shares a Common Lisp environment with the OMAS platform. First, we took the stub of proxy agent described in Section 6.4.3 and defined “memory skills” (i.e., skills intended to store and retrieve data) related to the SoS domain, for example, storing and

retrieving code quality analysis and recommendations. Then, we implemented them by using the AllegroCache syntax.

Next, we chose to implement the proxy agent of the MySQL database. We followed the same steps we did with the AllegroCache by defining, for instance, the same agent skills, however now using SQL statements as the syntax. When, we finished the implementation of the MySQL memory, we have realized that the changes between the proxy agents of the two databases were mainly concerned to the particular syntax of each one. For instance, within the skill for storing code quality analysis, the structure and the database operation, i.e., *insertion of data*, were the same for both interfaces.

After that, when implementing the proxy agents of the Redis and the MEMORAe databases we made it with much less time and effort, as we have reused partially the first two interfaces. The conclusion was that after implementing the four different memories, it would be better if we could provide some generic interface for the proxy agents of memories, in order to facilitate their development.

The result was that we developed and then incorporated to the MBA architecture, a generic memory interface providing a structure with an **Abstract Statement** for handling four basic DB operations, i.e., insert, remove, update and select (as stated in Chapter 4). This means that, one using such a generic interface needs only to add once the syntax of her database in the four basic functions, and then change it when reusing the interface with other DBs. However, the rest of the interface intends to remain the same, for instance, the skills used for handling SoS functionalities.

6.4.7 Coteries

In this section we describe how we have handled the network structure for allowing the ACE4SD systems communicate through work orders which are based on general broadcasts or Contract Nets.

To allow such a message exchanging we borrowed and adapted to our context the concept of coteries from the OMAS platform. A *coterie* is a set of constituent systems of an SoS residing on the same physical LAN loop. Thus, in this case constituent systems of an MBA SoS are able to broadcast work orders through their proxy agents to all other systems with a single message.

However, when MBA constituent systems reside on distinct or distant physical LAN loops we must provide a communication channel between the two coteries. The easiest approach is to connect the loops through a proxy agent which is implemented as an OMAS Transfer Agent (XA). The XA connects coteries transferring the messages received from one coterie to the other active coteries automatically. Among the protocols

provided by the XA to connect coteries are TCP, UDP and HTTP. Such an approach allows an MBA SoS to keep communicating normally by broadcasting work orders.

The concept of coteries used by the MBA architecture has several advantages for an SoS. First, a constituent system needs to send only a single work order for requesting functionalities from all other systems at the same time. Moreover, the fact of using broadcast messages avoid the needed of direct connections and white or yellow pages between systems. Furthermore, it fits especially well for organizations developing their SoS with a concern on safety, as the information exchanged among constituent systems is visible and received only by them. Finally, still related to safety the approach "protects" the SoS borders, as it is usually open because of its evolutionary nature, thus making it more robust.

6.5 Discussion

In this chapter we have presented the ACE4SD SoS which is a functional prototype we used as a basis for designing, improving and moving the MBA architecture towards a framework. Moreover, the ACE4SD also aimed to validate and show the feasibility of the approach proposed in this research. The SoS was built in the domain of collaborative software development and it focused on supporting the role of developers and managers to improve the quality of code developed.

By developing the prototype we have learned several different lessons that can be useful when building an SoS with the MBA framework. Below we highlight some of them discussed throughout this chapter.

First, through the OMAS platform used in our prototype we discovered that we could replace real brokers by virtual brokers. This brings a lot of advantages for the SoS such as it does not need to be implemented physically, the load of the messages exchanged among systems is reduced, and it keeps a pure P2P among constituent systems.

Moreover, we noted that we could reuse the interfaces of proxy agents, and thus reducing the efforts and the time for developing the SoS. This has led us to create a generic stub of proxy agent containing elements ready to be used by the interfaced systems, for instance, standard network protocols, JSON translators, or skills to handle work orders. Moreover, by interfacing the databases (memories) in the ACE4SD, we experienced that we could take the stub of the proxy agent and generalize it with a focus on the SoS memories. The result was that we created a generic proxy agent interface for memories containing an *Abstract Statement* that was then incorporated to our MBA architecture.

Besides, applying the concept of coteries was very useful and valuable to our approach. First, by using coteries we realized that we do not need to keep white or yellow pages between constituent systems of an MBA SoS, because messages can be sent through

broadcasts. Moreover, it makes the cooperation and exchange of information among SoS systems safer as only the own systems receive it.

Chapter 7

Evaluation

Building the ACE4SD SoS (Chapter 6) in the domain of collaborative software development has led us to develop the MBA architecture. This allowed us to evaluate and perform experiments in the proposed approach. The results are presented in this chapter. The goal of our evaluation is to verify if the MBA architecture facilitates the development of SoS, measuring the effort and difficulty for building and allowing evolution.

Following, first we describe the design of the experiment we have performed in the evaluation. Then, we present the quantitative metrics we have applied for measuring the experiment. After that, we show our experimental set up, and then the results we obtained for performing our experiment. We close the chapter with a discussion of the results.

7.1 Experimental design

The evaluation we conducted intends to verify if the MBA architecture facilitates the development of SoS, regarding the difficulty and effort required when building the interfaces to connect the constituent systems of an SoS. Moreover, it also estimates and compares the maintainability of such interfaces, i.e., the ease with which they can be modified or adapted to changes (Radatz et al., 1990), because SoS change over time due to their evolutionary nature.

To perform our evaluation we have used an extract of the ACE4SD SoS (Chapter 6) we named *ACE4SD-EX*. Figure 7.1 shows the ACE4SD-EX which was developed with the MBA approach and uses a virtual broker, consisting of coding systems, a code analysis system, an object database system, Manager information systems, and a Web search system.

Then, based on the ACE4SD-EX we have built an SoS in a traditional way using an ad hoc approach, i.e., directly and without being constrained by a predefined structure.

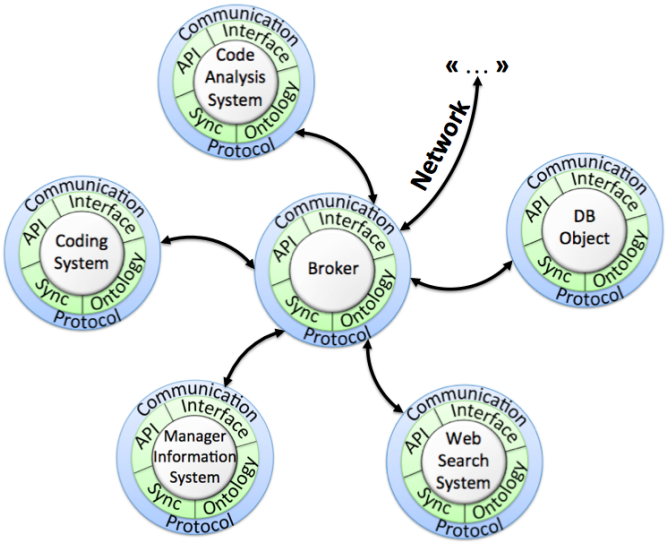


Figure 7.1: The ACE4SD-EX used for performing the evaluation of the approach proposed in this research.

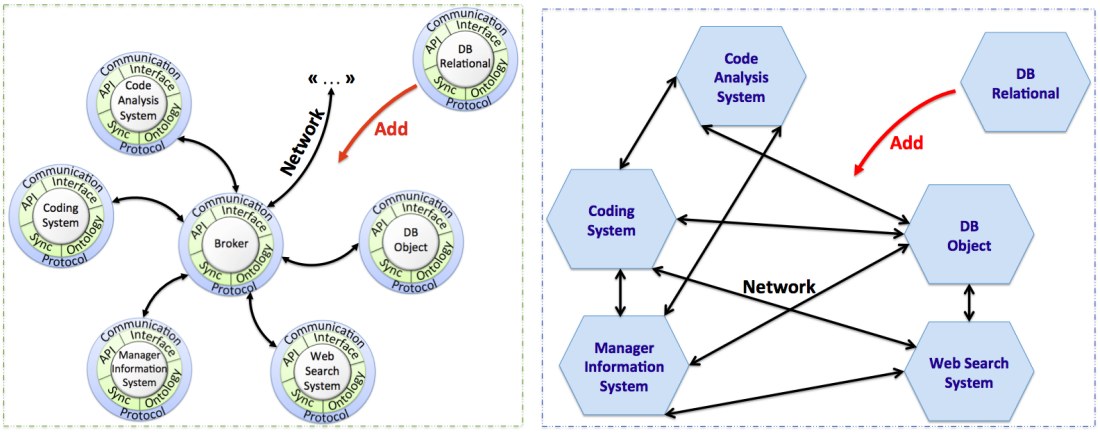
In the ad hoc approach, the constituent systems of the ACE4SD-EX were connected directly, according to the requirements shown in Table 7.1. Such requirements were defined based on the roles of the ACE4SD systems described in Chapter 6.

Table 7.1: The connections among constituents systems used for building the ad hoc approach of the ACE4SD-EX.

	Coding system	Code analysis	Object database	Manager information	Web search
Coding System	-	X	X	X	X
Code analysis	X	-	X	X	-
Object database	X	X	-	X	X
Manager information	X	X	X	-	X
Web search	X	-	X	X	-

After that, to determine the difficulty and effort for building an SoS, we added a new system to the ACE4SD-EX SoS for both the MBA and the ad hoc approach. The system we added was a relational database (see Figure 7.2), and its choice was mainly motivated by our experience of using it with an SoS (see Chapter 6).

Compared with the ACE4SD systems described in Chapter 6, the relational database played the same role as the object database. Meaning that, the SoS functionalities it provided were to store or to retrieve code quality analysis and recommendations. Thus, we connected it in the ad hoc ACE4SD-EX using the same requirements we used for the object database (see Table 7.1).



(a) The ACE4SD-EX built with the MBA approach. (b) The ACE4SD-EX built with an ad hoc approach.

Figure 7.2: Adding a new system to the ACE4SD-EX to determine the difficulty and effort for building SoS through the MBA and ad hoc approaches.

When adding the system, we computed the Halstead complexity metrics (Halstead, 1977) for the programs implementing the interfaces of constituent systems impacted by the change, in both MBA and ad hoc approaches. We chose such an approach because besides it can compute the difficulty and effort to implement the interfaces, it is a traditional and one of the more widely studied software metric suite (Arar and Ayan, 2016).

Moreover, to verify the maintainability of the code of the interfaces impacted by the addition of the new system, we also computed the Maintainability Index (Oman and Hagemeister, 1994) for the MBA and ad hoc approaches. Such an index has been considered a good predictor of maintainability (Bray et al., 1997), and is used in several analysis and code development tools such as Microsoft Visual Studio[1].

Finally, we compared the values obtained for the Halstead's metrics and for the Maintainability Index regarding the MBA and ad hoc ACE4SD-EX SoS.

Next, we present the Halstead complexity metrics, and then the Maintainability Index. After that, we show the experimental set-up, and then the results obtained for our experiment. We close by discussing the results of our evaluation.

7.2 The Halstead complexity metrics

The Halstead's metrics (Halstead, 1977) are software metrics used to compute quantitatively the complexity of source code. The calculation of such metrics is based on the number of distinct operators, the number of distinct operands, the total number of

[1]See `https://docs.microsoft.com/fr-fr/visualstudio/code-quality/code-metrics-values` for more information.

operators and the total number of operands of the program, denoted as η_1, η_2, N_1 and N_2, respectively. The metrics calculated are:

- Vocabulary (η): the total number of distinct operators and operands. Given by $\eta = \eta_1 + \eta_2$
- Program length (N): the sum of all necessary tokens for the computation of the program. Given by $N = N_1 + N_2$
- Volume (V): the number of bits necessary to represent the program. Given by $V = N \times \log_2 \eta$
- Difficulty (D): the difficulty to write or understand the program. The longer the implementation, the higher the difficulty. Given by $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort (E): the effort required to create or understand the program. Given by $E = D \times V$
- Time required to program (T): an approximation of the time to implement a program (in seconds). Given by $T = \frac{E}{18}$

7.3 The Maintainability Index

The Maintainability Index (MI) comes from the context of software quality. It is a single-value that estimates how maintainable (easy to support and change) a source code is (Oman and Hagemeister, 1994). The MI is computed by combining four traditional metrics. It is a weighted composition of the average Halstead's Effort ($aveE$) per module, the average extended Cyclomatic Complexity ($aveV(G')$) per module (Myers, 1977), the average number of lines of code ($aveLOC$) per module, and the average percent of lines of comments ($aveCMT$) per module.

The polynomial used to calculate the MI is given by Equation 7.1:

$$MI = 171 - 3.42 \times \ln(aveE) - 0.23 \times aveV(G') - 16.2 \times \ln(aveLOC) + 50 \times \sin(\sqrt{\frac{2.4 \times aveCMT}{aveLOC}}) \quad (7.1)$$

According to Coleman (1992) and Oman and Hagemeister (1994), a code with an $MI < 65$ is considered having a "low" maintainability, one with MI between 65 and 85 is considered having "medium" maintainability, and code with an $MI > 85$ is considered having a "high" maintainability.

7.4 Experimental set-up

In the experiment the systems we used were:

- EclipA[2] (Appendix E) for the coding systems.
- CoQAS (Appendix C) as the analysis system.
- AllegroCache[2] as the object database.
- MBA Browser (Appendix F) for the manager information systems.
- PeWeS[2] (Appendix D) as the Web search system.
- MySQL[3] as the relational database added to the SoS.

The experiments of the evaluation were performed by an SoS architect with intermediary experience in Common Lisp programming language. Moreover, it was the first time he developed SoS using the MBA approach.

When performing the experiments, the architect has used Common Lisp through the Allegro CL[4] environment for implementing the interfaces of the constituent systems of the ACE4SD-EX. The choice for such a language and environment was because the ACE4SD-EX was an extract of ACE4SD in which agents were implemented using the OMAS platform[5] (Barthès, 2011) that in turn is based on Common Lisp using Allegro CL. Thus, to avoid bias by using different languages, and to keep consistence when comparing results between the MBA and the ad hoc approaches, we resolved to use the Common Lisp language for both approaches.

For the format of data exchange, for instance, the content of the messages exchanged among constituent systems we adopted the JSON[6] standard, which is a very simple and widely used format. Moreover, for the inter-system connection in the ad hoc approach we used the TCP (Transmission Control Protocol) protocol. Our choice for such a protocol was because it is one of the more basic network standards offering reliability in the connections.

In both MBA and ad hoc approaches we handled the MySQL database system by calling its functions through a library[7] provided by the Allegro CL environment. Moreover, it is important to highlight that all the MySQL schemata and tables were already set up for the experiment, meaning that their creation was not taken into account in the evaluation. Our focus was truly in analyzing and evaluating the SoS interfaces.

[2]See `https://franz.com/products/allegrocache/` for more information.
[3]See `https://www.mysql.com/` for more information.
[4]See `https://franz.com/products/allegrocl/` for more information.
[5]See `http://www.utc.fr/~barthes/OMAS/` for full documentation.
[6]More information in: `http://www.json.org`
[7]More information in: `https://franz.com/support/documentation/6.0/doc/mysql.htm`

Moreover, to avoid bias when comparing the approaches, we have implemented minimal working interfaces in the constituent systems of the ACE4SD-EX SoS, for both MBA and ad hoc. Our goal was to take into account the essential elements needed for the interfaces to work.

For each interface impacted by adding the MySQL database in the ACE4SD-EX, we computed the Halstead's metrics including in the calculation the following elements (if used):

- the functions, skills (when MBA approach) and variables added to the interface.
- the inter-system messages added, for instance, for exchanging information or performing requests for SoS functionalities.
- the concepts and attributes added to ontologies (when MBA approach).

7.5 Results

We began our experiment for verifying if the MBA architecture facilitates the development of SoS by letting the architect add the MySQL DataBase (DB) to the ACE4SD-EX built with the MBA approach. To connect the database to the SoS he used the interface of proxy agent focused on a generic memory which is provided by the MBA architecture (Chapter 4). Note that, such an interface was a result of developing the ACE4SD (Chapter 6), and it provides facilities like translation of message content in JSON standard to Common Lisp lists (and vice versa), skills for handling work orders, and an Abstract Statement for handling four basic DB operations, named insert, remove, update, and select.

In the proxy agent, first the architect imported the MySQL library, and then he created two functions for connecting and disconnecting from the DB. After that, he added the SQL syntax of inserting and selecting data, enough for the SoS functionalities of the DB, to the function describing the Abstract Statement. Then, he defined and implemented four skills each one for handling an SoS functionality provided by MySQL. The skills implemented were for:

- storing code analysis.
- retrieving code analysis.
- storing recommendation.
- retrieving recommendation.

Next, he took the ACE4SD-EX global ontology (resulted from the ACE4SD SoS) and he derived one for the proxy agent, extending the concepts of "System" for "MySQL", and "Functionality" for "store-code-analysis," "retrieve-code-analysis," "store-recommendation," and "retrieve-recommendation." The attributes related to such concepts were also added and, specially the "feature" belonging to the "System" concept which describes a feature of MySQL, in the case "database." All the attributes were reused from concepts inherited when the ontology was built. Moreover, note that he did not need to update the ontologies of the other constituent systems using functionalities of the MySQL database. The reason was because such ontologies were already compatible with the one of the MySQL as the SoS functionalities it offered were the same provided by the AllegroCache DB.

Regarding requests for SoS functionalities provided by the MySQL DB, he did not add new work orders for the other constituent systems of the ACE4SD-EX. The reason was because the same work orders used for requesting functionalities of the AllegroCache were also valid for accessing the functionalities of the MySQL DB. For instance, when the CoQAS sent work orders for storing code quality analysis, the conditional addressing (see Chapter 6) of the work orders used the same feature for both DBs, i.e., "database." Thus, a single work order sent by CoQAS could reach SoS functionalities from both MySQL and AllegroCache.

A synthesis of the elements needed to add or change in the interfaces of the MBA ACE4SD-EX is shown in Figure 7.2.

Table 7.2: A synthesis of the elements added to the MBA ACE4SD-EX SoS when adding the MySQL database.

Approach	SoS Interface(s) changed	# of skills added	# of functions added or changed	# of ontology concepts added	# of ontology attributes added	# of work orders added
MBA	MySQL	4	3	4	37	-

On the other hand, when adding the MySQL database to the ad hoc ACE4SD-EX, first he created an interface in Common Lisp and imported the database library. Then he defined two functions for connecting and disconnecting from the DB. After that, he added four more functions for handling the SoS functionalities of storing and retrieving code analysis, and storing and retrieving quality recommendations. To avoid bias in the functionalities, we used the same structure (algorithm logic) we did for the MBA approach. However, for the ad hoc approach he created functions instead of agent skills, and he added SQL statements directly when needed because in this case we did not have a generic Abstract Statement.

Next, he added functions for data translation between JSON and Common Lisp. The reason was to support the message content among constituents of the ad hoc SoS, because

usually in practice its interfaces are not standardized and cannot be reused. He added four simple functions for data translation.

After that, he created functions for handling TCP connections among the database and all the other SoS constituents using its functionalities, based on the requirements described in Section 7.1. He implemented a function for receiving TCP connections and handling them in a multithread fashion, and a function for sending results to requesters. To distinguish among requesters he defined a very simple protocol using JSON syntax which an extract is shown as follows:

```
{
"msg-from":"CoQAS",
"msg-type":"coquas-request",
"functionality-requested": "store-code-analysis",
"input-data":"{"code-analysis":{...}}",
"requester-IP":"172.1.1.1",
"requester-port":"52008",
"receiver-IP":"172.1.1.2",
"receiver-port":"52008",
...}
```

Note that the constituents requesting functionalities from the MySQL DB kept references of the database such as IPs and ports. The information was also used by the DB for sending back the results to the requesters.

Finally, he updated the interfaces of constituent systems using functionalities of the MySQL, for instance, by adding the references above, and inter-system messages using TCP for allowing them perform requests. Table 7.3 summarizes the elements needed to add or change in the interfaces of the ad hoc ACE4SD-EX when adding the MySQL database system.

Table 7.3: A synthesis of the elements added to the ad hoc ACE4SD-EX SoS when adding the MySQL database.

Approach	SoS Interface(s) changed	# of functions added or changed	# of inter-system messages added
Ad hoc	MySQL	8	6
	CoQAS	3	1
	EclipA[2]	3	1
	MBA Browser	3	1
	PeWeS[2]	3	1

When the architect added the MySQL DB to the ACE4SD-EX SoS developed with both MBA and ad hoc approaches, we computed the Halstead's metrics and the Maintainability Index (MI) for the codes implementing the interfaces of constituent systems impacted by the change. Note that, our focus when computing the measures was on the elements added or changed in the interfaces impacted by adding the DB, i.e., not on the

ones which remained unchanged. Table 7.4 shows the results of the Halstead's metrics and the MI for the MBA and the ad hoc approaches. For better readability and comprehension of the results, we scaled the values of the Halstead's metrics between 0-100%, keeping the ratio and the original meaning between both approaches. For scaling, we chose (randomly) the second column of the table, i.e., the MBA approach, as a baseline considering its values as 100%. Then, for each row we divided the values of the other columns by the one corresponding to the baseline column.

Table 7.4: The results of the Halstead's metrics (taken the MBA as a baseline) and the Maintainability Index applied to both MBA and ad hoc approaches.

	MBA	ad hoc				
SoS Interfaces changed	MySQL	MySQL	CoQAS	EclipA^2	MBA Browser	PeWeS^2
Distinct operators (η_1)	100%	153%	13%	13%	13%	13%
Distinct operands (η_2)	100%	89%	14%	14%	14%	14%
Total operators (N_1)	100%	104%	3%	3%	3%	3%
Total operands (N_2)	100%	96%	5%	5%	5%	5%
Vocabulary (η)	100%	106%	14%	14%	14%	14%
Length (N)	100%	101%	4%	4%	4%	4%
Volume (V)	100%	102%	2%	2%	2%	2%
Difficulty (D)	100%	166%	4%	4%	4%	4%
Effort (E)	100%	170%	0.1%	0.1%	0.1%	0.1%
Time to develop (T)	100%	170%	0.1%	0.1%	0.1%	0.1%
Maintainability Index (MI)	27%	17%	97%	97%	97%	97%

One may see that the MBA approach needed to change only the MySQL interface for letting it provide its functionalities to other constituent systems. Conversely, besides changing the MySQL interface, in the ad hoc approach all the SoS systems requiring to use its functionalities had also to change or adapt their interfaces. The CoQAS, the EclipA^2, the MBA Browser, and the PeWeS^2, along with the MySQL database had their interfaces changed in the ad hoc approach. The main reason for that was because in the ad hoc approach the interfaces of constituent systems needed to keep references such as IPS and ports for allowing the SoS cooperation. On the other hand, the MBA approach did not require to do that thanks to work orders and to a virtual broker allowing constituent systems to perform requests through conditional addressing (Chapter 6). Note that, changing interfaces in an SoS may be an issue impacting the entire SoS and the expected results. For instance, constituent systems may have to stop offering their services while their interfaces are being adapted.

Moreover, the reason for the measured values of the CoQAS, the EclipA^2, the MBA Browser, and the PeWeS^2 ad hoc interfaces were exactly the same was because the code

added to their interfaces differed mainly in IPs and port numbers. That is, the TCP structure for the communication between such systems and the MySQL DB was very similar, and changes were mainly related to IP addresses.

Regarding the Maintainability Index (MI), both the MySQL interfaces of the MBA and ad hoc approaches did not show a good maintainability in the context of code quality (Section 7.3). The biggest impact that cause to such values may be assigned to the part of the code developed for handling the SoS functionalities provided by the MySQL database. Such a part corresponded to 88% and 72% of the code developed in the MySQL interface of the MBA and ad hoc SoS, respectively. The quality of code concerning these parts weighted the term of average number of lines of code ($aveLOC$) of the MI equation (Section 7.3). However, the focus of the SoS architect in our experiment was not truly on the code quality, but in developing the interfaces for connecting the DB to the SoS to allow it to cooperate with the other constituent systems. Thus, one may see that the MI for the MBA interface was higher (27%) than the one of the ad hoc approach (17%). Because, the SoS functionalities of the MySQL have almost the same structure for the MBA and ad hoc, as we stated above to avoid bias, such a better MI may be assigned mostly to the proxy agent interfaces provided by the MBA approach. For instance, regarding the generic Abstract Statement for handling DB operations.

The reason for the good MI of the CoQAS, the EclipA[2], the MBA Browser, and the PeWeS[2] ad hoc interfaces was because they were very simple, consisting of few lines of code (about 19), and not complex (mainly IPs and ports) as the SoS functionalities implemented in the MySQL interfaces. Thus, the terms $aveLOC$ and $aveV(G')$ of the MI equation did not impact negatively.

To better visualize and compare the results of the MBA and ad hoc approaches in another perspective, in the chart of Figure 7.3 we show a synthesis focusing on the effort, difficulty, time to develop, MI, and volume obtained by each approach for adding the new system (MySQL) to the SoS. For the ad hoc approach, the values of effort, difficulty, time, and volume correspond to the sum of all the changes made in the SoS interfaces impacted by the addition of the MySQL database. Our goal is to take into account an show all the changes required by both approaches to develop the SoS. Moreover, in the synthesis the MI of both approaches corresponds to the one of the MySQL interface which is the most significant and relevant, based on the effort and volume required to develop it. Furthermore, because the values of the variables used in the synthesis belong to different scales, we fitted some of them to scale between 0-100%, keeping the ratio and the original meaning between both approaches.

Through the chart one may see that developing the SoS through the MBA architecture was easier than with the ad hoc approach. The difficulty for building the SoS with the MBA was 46% less than by using the ad hoc approach. Moreover, the MBA took 42% less time and 40% less effort than the ad hoc for developing the SoS. In addition,

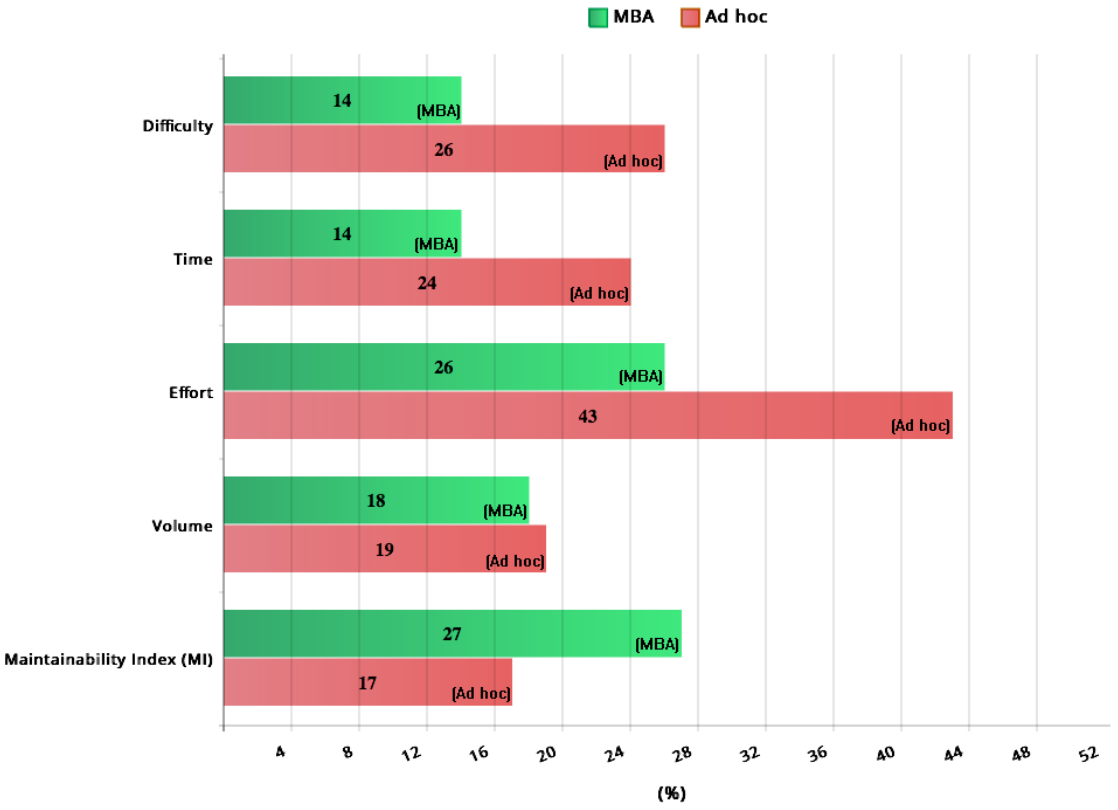


Figure 7.3: A synthesis of the results comparing the MBA and ad hoc approaches.

the size of the development (volume) required to build the interfaces was 5% less for the MBA. Finally, the interfaces developed through the MBA approach provided more maintainability, and may be 37% easier to modify or adapt in future SoS evolutions.

Next, we provide a discussion about the results obtained through the experiment.

7.6 Discussion

Performing the experiments described in this chapter has shown that the MBA architecture facilitates the development of SoS when compared to traditional handcrafted approaches. The evaluation proved that the approach proposed by this research facilitates building SoS, regarding the difficulty and effort required when implementing the interfaces to connect the constituent systems of an SoS. Moreover, our results demonstrate that the MBA architecture supports the evolutionary nature of SoS, building interfaces easier to reuse and to modify.

Some of the MBA aspects that helped achieving such results were:

- the possibility of reusing proxy agent interfaces, which standardizes the heterogeneous SoS environment.

- the work order protocol allowing constituent systems to perform requests using conditional addressing. As a result, a single work order could reach distinct systems based on their features.
- the loose coupling, which avoids keeping references of constituent systems.

On the other side, building the SoS through the ad hoc approach was a laborious task. First, there was no standard interface supporting it in the heterogeneous environment of the SoS, and the development was made practically from scratch. Furthermore, each constituent requiring the functionalities of the new SoS system needed to create and keep corresponding references in its interface. The consequence was that the number of interfaces changes was proportional to the number of constituent systems needed to use the functionalities of the new system. Although creating such references individually was not a difficult task, the true complexity appeared globally, increasing the effort for developing the SoS. Besides, the SoS built through the ad hoc approach is harder to modify. Despite the maintainability of the changes (the references created and kept) in the interfaces of existing constituents was considered good mainly because of their very light code (see Section 7.5), it also increases the entire complexity for allowing evolution of the SoS. The need for creating, removing or updating such references increases the coupling among constituent systems, rendering the evolution of the SoS much more difficult.

In addition, we need also to highlight that besides facilitating the development of the SoS, the MBA also provided syntactical and semantical interoperability (details in Chapter 4) among constituents of the SoS built. Conversely, besides making the development of the SoS harder, in the ad hoc approach, exchanging messages among constituent systems was only done at syntactical level.

However, of course there are also limitations to our approach. For instance, when a brand new SoS is built a domain ontology needs to be developed, perhaps from scratch. Furthermore, currently the interfaces of our proxy agents already provide support for some network standards (TCP, HTTP, and UDP) and a single data exchange format (JSON), meaning that if other or more specific standards or formats are needed they need to be implemented by the architect. Nonetheless, once such components are implemented in the proxy agents, they can be reused. Concerning a more technical aspect, currently the initial prototype of our approach was implemented in Common Lisp, thus architects must have a certain knowledge in this language for using it to build an SoS.

Finally, the design of our experiment focused on developing an SoS by adding a new system was only a first example for evaluating our approach. Of course, other scenarios could be used, for instance, one aiming to remove or to replace constituent systems. It is important to highlight that our approach in this research is an initial proposition for facilitating the development of SoS, and improvements may be required as new tests and experiments are being performed.

Chapter 8

General guidelines for developing MBA SoS

The experience of building the ACE4SD SoS along with the development of the MBA architecture led us to define the key points for developing an MBA SoS. Such key points have been gathered as general guidelines for supporting SoS architects with a logical sequence of steps to follow when developing MBA SoS.

In this chapter, first we present a method showing the proposed guidelines, then we show the results of using it for developing a brand new MBA SoS in the domain of Health Care. The target system is meant to improve the collaboration between patients and caregivers for ensuring the correctly understanding of medication prescriptions. By doing so, our goal was to validate the generic feature of the MBA architecture, testing its potential to be applied to different domains, and moving it towards a framework for developing SoS.

8.1 General guidelines

In this section we provide general guidelines for developing MBA SoS. We present the key points of our guidelines through the activities of a method we call *GAMBAD* (Guide for Actual MBA Development). Figure 8.1 shows the GAMBAD method consisting of the activities:

1. Define SoS Goal
2. Define SoS Functionalities
3. Select SoS Systems
4. Select SoS Memories

5. Define SoS Users
6. Extend the OntoMBA Ontology
7. Interface Systems - Proxy Agents
8. Integrate Proactive Interfaces

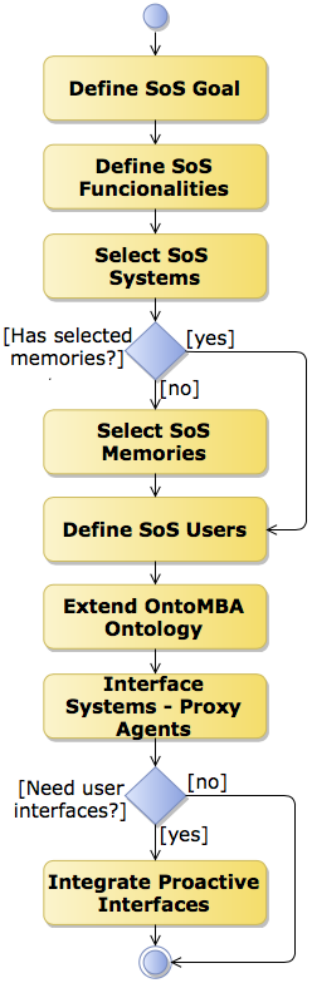


Figure 8.1: The GAMBAD method which provides general guidelines for developing MBA SoS.

The first key point we identified for developing an MBA SoS is "**Define SoS Goal**." This point is straightforward, meaning that the SoS architect needs to define the goal the SoS intends to achieve. For instance, for the ACE4SD SoS we defined the goal of supporting developers and managers to improve the quality of the code developed (details in Chapter 6).

After defining the SoS goal, the second point is "**Define SoS Functionalities**." In this key point, the architect defines the SoS functionalities, i.e., the inter-system functionalities constituents of the SoS can request and provide to each other, aiming to achieve the SoS goal. For example, to achieve the ACE4SD goal of improving code quality

we defined SoS functionalities like "to analyze code quality" provided by code analysis systems, and which could be requested by coding systems.

Next, the "**Select SoS Systems**" intends to select in practice the systems offering the SoS functionalities. For instance, for the ACE4SD we selected the CoQAS as the code analyzer offering analysis for code quality, and the EclipA[2] as the coding system capable of requesting the analysis. Note that, the selected systems must provide at least an API to be constituents of an MBA SoS (see MBA architecture Chapter 4).

The next activity is "**Select SoS Memories**." If memories are not among the selected systems, the architect is now invited to do that for providing means for knowledge capitalization and management during SoS operation. The number or kind of the memories is not an issue in the MBA architecture, thus it varies according to the requirements of the SoS. For instance, in the ACE4SD we selected four distinct kind of memories such relational (MySQL), object (AllegroCache), key-value (Redis) and triplestore (MEMORAe).

The "**Define SoS Users**" aims to define the SoS users and their roles when interacting and using the SoS. For the ACE4SD SoS we defined two users, named developers which used coding systems for producing code and receiving information about the its quality, and managers using browser systems for receiving information and gave orientations to developers to improve the code quality.

After that, in the "**Extend OntoMBA Ontology**" the architect takes the OntoMBA core ontology (Appendix A), and extends it according to the SoS being developed. Defining such ontology has several different roles such as modeling the SoS domain, modeling the users of constituent systems, handling the semantics of the communication protocol, and decoding interactions in natural language (see Chapter 4). When defining the ontology, the architect describes concepts related to the SoS goal, constituent systems or users. For instance, in the ACE4SD we extended the ontology describing, for example, the concept of "System" to include coding systems and the code quality analyzers. The extended ontology is the *global ontology* of the SoS.

The next key point is "**Interface Systems - Proxy Agents**." In this point, the architect interfaces the systems selected in"Select SoS Systems" and "Select SoS Memories," with proxy agents. The goal is to allow constituent systems to exchange information and cooperate with each other to achieve the SoS goal. For interfacing the systems, we recommend the architect to follow the steps shown in Figure 8.2. Details about each of these steps may be found in Chapter 4 and Chapter 6. The steps are:

- "Prepare System" which aims to prepare the system (if needed) like setting up tables in a relational database.
- "Connect System - Proxy Agent" which aims to connect systems that are not memories with proxy agents.

- "Connect Memory - Generic Interface" which intends to connect memories with proxy agents focused on a generic memory interface.
- "Derive Proxy Agent Ontology" which intends to derive ontologies for proxy agents from the global SoS ontology.
- "Store Global Ontology" which aims to store the global ontology in the proxy agents of memories.
- "Define Skills" which aims to define skills in the proxy agents for handling the SoS functionalities requested and provided by the systems connected to them.

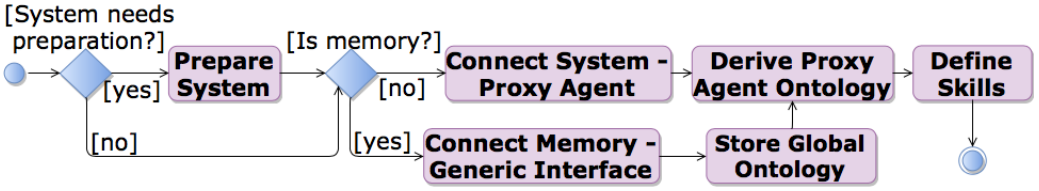


Figure 8.2: The steps of the "Interface Systems - Proxy Agents" activity of the GAMBAD method.

If the SoS users defined in the "Define SoS Users" do not require interfaces for interacting with the SoS, then the architect can skip the next activity and the GAMBAD is finished. Otherwise, the "**Integrate Proactive Interfaces**" aims to integrate proactive interfaces (Chapter 5) into the systems of SoS users for supporting them, for instance, to request or receive information from the SoS to help it to achieve its goal. When integrating the interfaces, we recommend the architect to follow the steps shown in Figure 8.3. Details about each of these steps may be found in Chapter 4, Chapter 5 and Chapter 6. The steps are:

- "Augment System" to augment the user's system for creating a Personal Assistant (PA) interface, if chosen.
- "Use MBA Browser" to use MBA Browsers as the PA interface, if chosen.
- "Connect Local Server" to connect a local server handled by an agent for MBA Browsers.
- "Connect PA and Staff" to connect a Personal Assistant and staff agents to the proactive interface.
- "Define Skills" to define skills in the PA for handling the message exchanging with its user, and SoS functionalities related to the system which it is integrated, and may be helpful for supporting the user.
- "Develop Dialogues" to develop the PA dialogues for interacting with the user of the system.

- "Follow ProPA" to follow the ProPA method (Chapter 5) for adding proactive behavior to the interfaces, if desired.

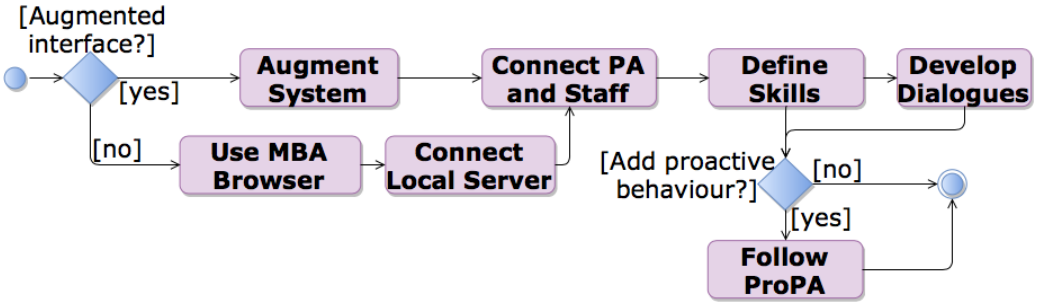


Figure 8.3: The steps of the Integrate Proactive Interfaces of the GAMBAD method.

To our mind, the guidelines described above may be a support and provide a path for SoS architects when developing an MBA SoS. To show in practice the use of our guidelines and also to validate the generic feature of the MBA architecture, next we detail a brand new MBA SoS we developed in the domain of Health Care.

8.2 Using the guidelines for validating the MBA architecture through a brand new SoS

In this section we present and detail an MBA SoS developed in the domain of Health Care by employing our guidelines. By doing that, our goal is to show the usefulness of our guidelines for supporting the development of MBA SoS, while validating the generic feature of the approach proposed in this research.

8.2.1 SoS Context and problem statement

The SoS developed belongs to the context of the CONSIGNELA project which is the French acronym for CONSIGNe ELectroniques Adaptées, i.e., adapted electronic instructions (Wanderley et al., 2017b). The CONSIGNELA is a multidisciplinary research project started in 2015 and it is still active by the time of writing this thesis. The project is coordinated by Prof. Laurent Heurley[1] at the Université de Picardie Jules Verne (France), and is co-funded by the Regional Council Hauts-de-France and the European Regional Development Fund (FEDER) 2014/2020 (see Figure 8.4).

The CONSIGNELA includes teams with competences coming from different fields such as medicine, cognitive psychology, computer science and engineering. Among the research partners developing the project are the research center CRP-CPO[2] and the

[1]Professional Web page available in: `https://www.u-picardie.fr/m-laurent-heurley--147099.kjsp`

[2]See `https://crpcpo.u-picardie.fr` for more information.

Figure 8.4: The co-funders of the CONSIGNELA project.

LNFP[3] laboratory at the Université de Picardie Jules Verne, the Heudiasyc[4] laboratory at the Université de Technologie de Compiègne, and the Neurology department[5] at the Centre Hospitalier Universitaire (CHU) d'Amiens. Figure 8.5 shows the research partners of the CONSIGNELA project.

Figure 8.5: The research partners of the CONSIGNELA project.

The starting point of the CONSIGNELA project was that medication prescriptions are generally communicated via documents designed for standard adults. These documents are not adapted to older patients and patients with cognitive impairment. The CONSIGNELA project aims at identifying the best way to communicate medication prescriptions with tablets and touch-screen devices for two categories of patients: older and Parkinsonian patients. Moreover, the project intends to improve medication adherence by creating a collaborative system designed both for older and Parkinsonian patients and for caregivers in charge of them. Two solutions are combined: A cognitive solution and a technological solution. The cognitive solution consists in studying in real time older and Parkinsonian patients while consulting and executing prescriptions presented in different formats to select the most efficient one. The technological solution aims at improving collaboration between caregivers and patients by developing an SoS.

The CONSIGNELA project has different phases. In the earliest phases the project members developed and tested a research tool, named CONSIGNELA-Appli-R which "R" stands for Research (Vandenbergh et al. (2017a); Vandenbergh et al. (2017b)). The CONSIGNELA-Appli-R is based on virtual pillboxes specially designed to study in real time patient behavior and cognitive processes while consulting and executing, i.e., filling the virtual pillboxes, a prescription presented in different formats on a tablet or a touch-screen. The main goal of the CONSIGNELA-Appli-R was to provide clear answers

[3]See https://www.u-picardie.fr/unites-de-recherche/lnfp/laboratoire-de-neurosciences-fonctionnelles- et-pathologies-382859.kjsp for more information.
[4]See `https://www.hds.utc.fr` for more information.
[5]See `http://www.chu-amiens.fr/patients-visiteurs/services-et-contacts/medecine/neurologie/` for more information.

to the question "What presentation format is best adapted to older and Parkinsonian patients?" (Wanderley et al., 2017b).

On the other hand, in more recent phases of the CONSIGNELA, one of the main concerns was to develop an SoS for improving collaboration between caregivers and patients. The SoS, named CONSIGNELA-Appli-P ("P" for Patients and Prescribers), was designed to be a pedagogical tool for ensuring that patients using virtual pillboxes have correctly understood the instructions contained in their medication prescriptions. Through the CONSIGNELA-Appli-P SoS, caregivers and patients could cooperate at distance for ensuring the correctly understanding of the prescriptions. For instance:

- caregivers could use the pillboxes for creating and parameterizing new medication experiences, to type the prescriptions or also to give advice related to such experiences to their patients.
- patients could use the pillboxes to perform the experiences. They could access the prescriptions, read the advice (if the case) and follow the medication regimens according to the prescriptions. Each time an experience was finished, the results were analyzed by an experience analysis system to know if patients have been followed correctly the prescriptions. After that, the results and analyses were available to be accessed and viewed by caregivers through browsers and a knowledge platform.
- caregivers could use browsers endowed with proactive interfaces, asking them to request analysis in the results of past experiences to better understand them. For instance, after receiving a request of this kind, the experience analysis system could inspect the results and send back the accuracy average related to some parameter such as patient type or age.
- caregivers could cooperate with each other through a knowledge platform, for instance, to exchange ideas or experiences based on their patients' results, in order to improve the patient understanding of the instructions.

In the CONSIGNELA-Appli-P, the raw information resulted from the patient experiences and the analyses performed were stored in relational databases. They were a means for supporting the Knowledge Management (KM) made by the knowledge platform in the SoS.

Figure 8.6 shows the CONSIGNELA-Appli-P SoS we built using our MBA framework. Note that here the broker is represented as a "medium," because in the CONSIGNELA-Appli-P we implemented it as a *virtual broker* following a discovery we made when building the ACE4SD (details in Chapter 6). The use of a virtual broker has its advantages as we stated in Chapter 6, and which we recall some of them in the next sections when we detail how we built the CONSIGNELA-Appli-P SoS by using each key point of our guidelines for developing MBA SoS.

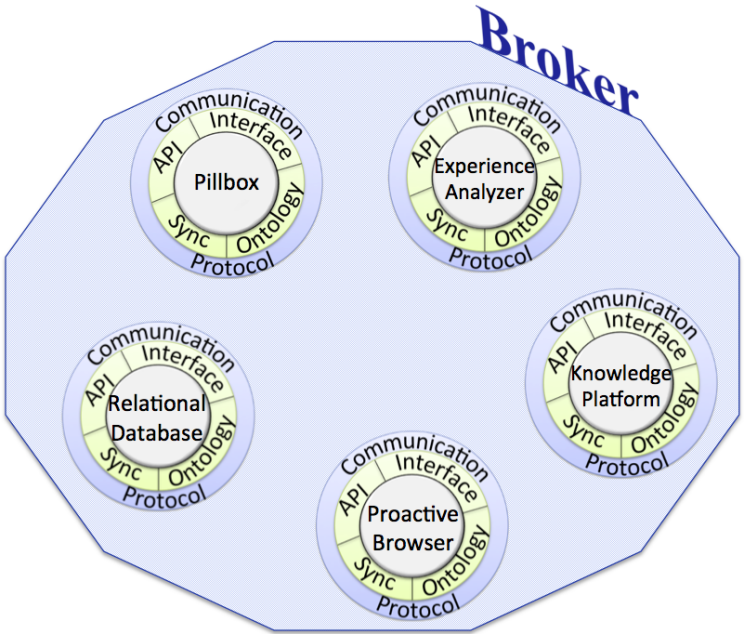


Figure 8.6: The CONSIGNELA-Appli-P SoS built with the MBA framework, and which uses a virtual broker.

8.2.2 Guideline point: Define SoS Goal

Based on the SoS requirements described above for improving collaboration and ensuring correctly understanding of prescriptions, we defined the main goal of the CONSIGNELA-Appli-P SoS as:

- Improving the collaboration between patients and caregivers for ensuring the correct understanding of medication prescriptions.

8.2.3 Guideline point: Define SoS Functionalities

Following the definition of the goal we describe the scenario required by the CONSIGNELA-Appli-P. After that, based on such a scenario we show the functionalities we defined for the SoS.

Assuming a caregiver wants to create a new experience of medication prescription for a patient. Then, first she takes a tablet with a pillbox and touches on “Prepare experience,” starting to fill different parameters regarding the experience and the patient such as “caregiver initials,” “date,” “patient initials,” “patient code,” “patient age,” “patient type,” “prescription type,” and “pillbox type.” Next, she types the prescription for such an experience detailing the medicines, day of week, period of the day, and frequency. Finally, as an option she can enter advice for helping her patient to follow the prescription in the experience.

To begin the experience, the patient touches on "Start experience" in the pillbox, and then she begins to read the caregiver's advice and the medication prescription. When the patient is ready, she starts to perform the experience by filling a pillbox. While she is following her prescription, different actions and indicators concerning her behavior and interaction with the pillbox are recorded in a non-intrusive way. Table 8.1 shows the list of actions and indicators with their possible values recorded during an experience.

Table 8.1: List of actions and indicators recorded during an experience performed by a patient in a virtual pillbox.

Variable	**Indicator**	**Assumed Values**
Ci	Caregiver initials	text
Pc	Patient code	text
Pt	Patient type	Parkinsonian/Elderly/Control/Other
Vt	Pillbox type	Tabular/Circular
J	Date	YYYY-MM-DD
Zt	Prescription type	Written/Tabular
Z	Prescriptions	A medicine prescription
Fvp	Final state of the virtual pillbox	Gives the final state of the pillbox
Tom	Elapsed time to open a medicine box	milliseconds
Toa	Elapsed time to open a compartment of the pillbox	milliseconds
Tma	Elapsed time to put a medicine inside a compartment	milliseconds
Tcm	Elapsed time to close a medicine box	milliseconds
Tca	Elapsed time to close a compartment of the pillbox	milliseconds

Once the patient has completed the experience, the actions and indicators are extracted along with information related to the parameterizing of the experience, the prescription, and the final state of the pillbox. Then, the patient's pillbox sends the information in JSON format to an experience analysis system. The goal of such a system is to perform analyses for supporting the caregiver with a better understanding of the results. An extract of the JSON received by the analysis system is shown as follows:

```
{
"caregiver-initials":"BRA",
"date":"19-1-2018",
"patient-code":"S1_VA70-1AG_19-1-2018",
"patient-type":"AG",
"pillbox-type":"tab",
"prescription-type":"verb",
"steps":[
{"elapsed-time":0,"action":"clicked_start_button",
 "event":
 "triggering-chronometer-&-display-prescription"},
{"elapsed-time":20375,"action":"clicked_pillbox_button",
 "event":"display-pillbox"},
{"elapsed-time":24908,"action":"clicked_compartment-M1J1",
 "event":"not_visible"},
 ...],
"prescription":{
    "id": 1,
```

```
        "number":1,
    "name": "hypertension-prescription",
    "day-period": [
      {
        "period": "morning",
        "day": "Tuesday",
        "abs": 1,
        "coord": 2,
        "medicine-list": [
          {
            "name": "ANIVRAX",
            "amount": 2,
       ...}
     ...]
      },
 ...]
},
"Result":{
    "patient-code":"S1_VA70-1AG_19-1-2018",
    "day-period": [
      {
        "period": "afternoon",
        "day": "Monday",
        "abs": 2,
        "coord": 1,
        "medicine-list": [...]
      },
 ...]
  }
}
```


The analysis system receives the JSON data, and then analyses them to verify if the patient has understood correctly the medication prescription. To do that, it calculates the accuracy of the experience by comparing the prescription with the final state of the pillbox. Equation 8.1 defines the accuracy of the experience, where n and m are the number of days in a week, and the number of periods in a day, respectively. The $MedicineIndicator$ is defined by Equation 8.2, and it concerns the amount of medicines taken (FinalAmount) and the amount of medicines described in the prescription (PrescriptionAmount).

$$Accuracy_{exp} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} MedicineIndicator}{n+m} \times 100 \tag{8.1}$$

$$MedicineIndicator = \begin{cases} 1, & \text{if FinalAmount } \textit{equal} \text{ PrescriptionAmount} \\ 0, & \text{if FinalAmount } \textit{not equal} \text{ PrescriptionAmount} \end{cases} \tag{8.2}$$

After analyzing the experience, the analysis system outputs a JSON appending the analysis results to the original information received. An extract of the resulting JSON is shown as follows:

```
{
"caregiver-initials":"BRA",
"date": "19-1-2018",
"patient-code":"S1_VA70-1AG_19-1-2018",
"patient-type":"AG",
"pillbox-type":"tab",
"prescription-type":"verb",
"Analyses":{
  "prescriptionID":1,
  "experienceAccuracy":58.20
  "day-period":[
     {
      "period":"evening",
      "day":"Sunday",
      "abs":3,
      "coord":7,
      "medicine-list":[
          {
           "analysisResult":"CORRECT",
           "name":"ANIVRAX"
       ...}
     ...]
    ...},
   ...],
    },
...}
```

Then, the analysis system sends the JSON message to different systems in the SoS. One of them is a relational database that aims to support the KM in the SoS by storing the raw information resulting from the patient experiences and the analyses. The database receives the JSON message, extracts the data based on its properties, and stores it in relational tables which share their column names with the JSON properties. Another system is a browser with a proactive interface for caregivers. The Personal Assistant (PA) of the interface receives the JSON and extracts its data, creating a synthesis in natural language to warn the caregiver about the results. If the accuracy of the experience is below a certain threshold, then the PA acts proactively to try to find useful information for supporting the caregiver, for instance, by searching annotations or recommendations

associated with past experiences concerning the same type of prescription, pillbox, or patient. Such an information may be fetched from a knowledge platform which is also the last system receiving the JSON message from the analysis system. The knowledge platform keeps a synthesis of the experience, by extracting the JSON data concerning the analysis results and then translating it into a PDF format. The resulting file is indexed to the caregiver's personal space in the platform by using ontological concepts such as "experience," "prescription," "medicine," or "advice." The caregiver is then able to use the information by accessing the knowledge platform directly or asking her PA to open it in the browser.

The goal of the knowledge platform in the SoS is to manage knowledge along with the relational database, and to provide an environment for caregivers cooperation. When the caregiver is navigating in the knowledge platform she is able to share with other caregivers by using a common sharing space, the synthesis PDF or any other information she wants regarding her experiences. The caregivers accessing the common space can use it, for instance, to collaborate around the shared experiences, for instance, by annotating or recommending them, rating, discussing in forums, writing Wiki articles, or adding other resources such as scientific studies or links to Web pages.

Another interesting possibility for caregivers in the SoS is to ask their PAs to request some kind of analysis of past experiences (see Table 8.2). For instance, assuming a caregiver say to her PA "I want a chart showing the accuracy of past experiences." Then, after analyzing the input in natural language, the PA asks its caregiver for more clarifications. Once it receives the answer it sends a request to the analysis system with the message content in JSON format, containing the parameter "Patient type." The analysis system in turn sends a request to the relational database for retrieving in JSON the accuracy and the patients of all experiences made by the caregiver. Once it has the required information, the analysis system analyzes the data and sends the results in JSON format to the PA. Then, the PA builds a chart with the information and displays it to its master. For caregivers, the information resulted from the analyses can be a useful support for better understanding the results of their experiences, in order to ensure the correct understanding of medication prescriptions by their patients. Moreover, such information may also be a support for collaborating with other caregivers.

Table 8.2: An example of request made by a caregiver to her PA for asking some kind of analysis in past experiences.

Interlocutor	Utterance
...	...
Caregiver	I want a chart showing the accuracy of past experiences.
Personal Assistant	Are you interested in the accuracy of patient or prescription type?
Caregiver	Patient type
Personal Assistant	OK, I got it.
Personal Assistant	Please, find below a chart showing the information:
...	...

Based on such a scenario, we defined the following **SoS functionalities** for the CONSIGNELA-Appli-P:

1. To analyze experience.
2. To store raw information of experience results and analyses.
3. To retrieve raw information of experience results and analyses.
4. To store synthesis of experience results and analyses.
5. To fetch annotations or recommendations associated with synthesis of experiences.
6. To show experience analysis results.

Table 8.3 shows the SoS functionalities with the systems offering them required by the CONSIGNELA-Appli-P to achieve its goal.

Table 8.3: The SoS functionalities requested by systems and the possible providers for them in the CONSIGNELA-Appli-P.

		Provider				
		Virtual pillbox	Browser proactive interface	Analysis system	Knowledge platform	Relational database
Requester	Virtual pillbox	-	-	1	-	-
	Browser proactive interface	-	-	1	5	-
	Analysis system	-	6	-	4	2; 3
	Knowledge platform	-	-	-	-	-
	Relational database	-	-	-	-	-

8.2.4 Guideline point: Select SoS Systems

After defining the SoS functionalities with the systems offering them, we selected the constituent systems in practice. For the CONSIGNELA-Appli-P we followed four criteria for selecting the SoS systems: (i) their suitability to the SoS context; (ii) the SoS functionalities they could provide; (iii) their availability in the research project; and (iv) recommendations from domain experts. We ended by selecting the CONSIGNELA-Appli-R (version V1.1) as the virtual pillbox, the MBA Browser (Appendix F) as the browser with the proactive interface, an MAS system "Experience Analyzer" as the experience analysis system, MEMORAe[6] (Abel, 2015) as the knowledge platform, and MySQL[7] as the relational database.

[6]See `http://memorae.hds.utc.fr/demo/labo/` for more information.
[7]See `https://www.mysql.com/` for more information.

The first system we selected was the CONSIGNELA-Appli-R-V1.1 (Wanderley et al., 2017b) which is the **virtual pillbox** described briefly in Section 8.2.1. In the CONSIGNELA -Appli-R-V1.1 caregivers prepare medication prescriptions and patients (older and Parkinsonian) follow such prescriptions through virtual pillboxes lying on tablets or touch-screen devices. Moreover, the system also allows tracking how patients follow their prescriptions through eye movement by means of eye-tracking devices.

It is important to highlight that both caregivers and patients use the CONSIGNELA-Appli-R-V1.1. The difference is that caregivers have access to more functionalities, for instance, for parameterizing experiences or typing the medication prescriptions. Our choice for selecting the CONSIGNELA-Appli-R-V1.1 as the virtual pillbox was mainly because it met the four criteria above.

First, it fitted well the SoS context, for instance, by providing means for caregivers type prescriptions or advice, and by allowing patients to follow their medication experiences. Moreover, for the SoS functionalities the pillboxes already exported data concerning the prescriptions and their final states after experiences were performed, thus it was a step towards facilitating the functionality of analyzing the experience results. Furthermore, because it was developed in the CONSIGNELA project, the system was fully available to be used and modified by the research team. The CONSIGNELA-Appli-R-V1.1 was developed mainly using C#. Finally, there was a consensus for using the system as one of the constituents of the SoS, mainly because it was specially designed and fitted for the context of the project.

In the CONSIGNELA-Appli-R-V1.1, caregivers can customize patient experiences based on a number of parameters. Figure 8.7 shows the screen of the system used by caregivers to parameterize the experiences. Several different parameters are filled by the caregiver, for instance, the “caregiver initials,” “date,” “patient age,” “patient type,” “prescription type,” “pillbox type,” and “patient code.” The possible values assumed by such parameters are shown in Table 8.1.

Another example of the virtual pillbox is shown in Figure 8.8. In the figure, one may see a medication prescription for fever, created by a caregiver using a tabular format. The prescription is then used by patients to follow a medication regimen through their pillboxes. In the top-left corner there are three medicines, and their pills are distributed along the days of the week and the different periods of a given day.

An example of a patient realizing an experience in her pillbox is shown in Figure 8.9. On the top of the screen there are two boxes. Each one stands for a different medicine and shows the number of pills it contains. Moreover, there is a table showing the seven days of the week (columns) and the periods of the days (rows), i.e. morning, afternoon and evening used by patients to follow their regimens. According to the prescriptions, the patients fill the compartments of the table with the medicines available in the boxes. The main sequence of their actions is: Choose a medicine and open its box, select the

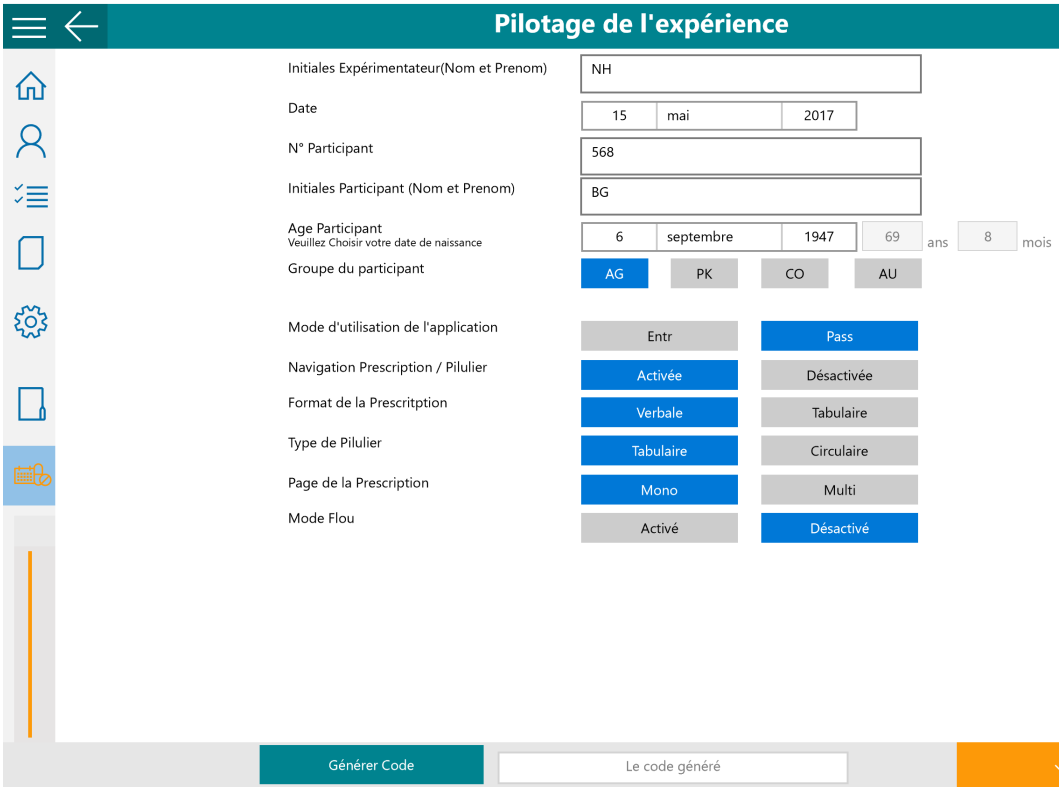


Figure 8.7: The screen used by caregivers for parameterizing the patient experiences in the CONSIGNELA-Appli-R-V1.1.

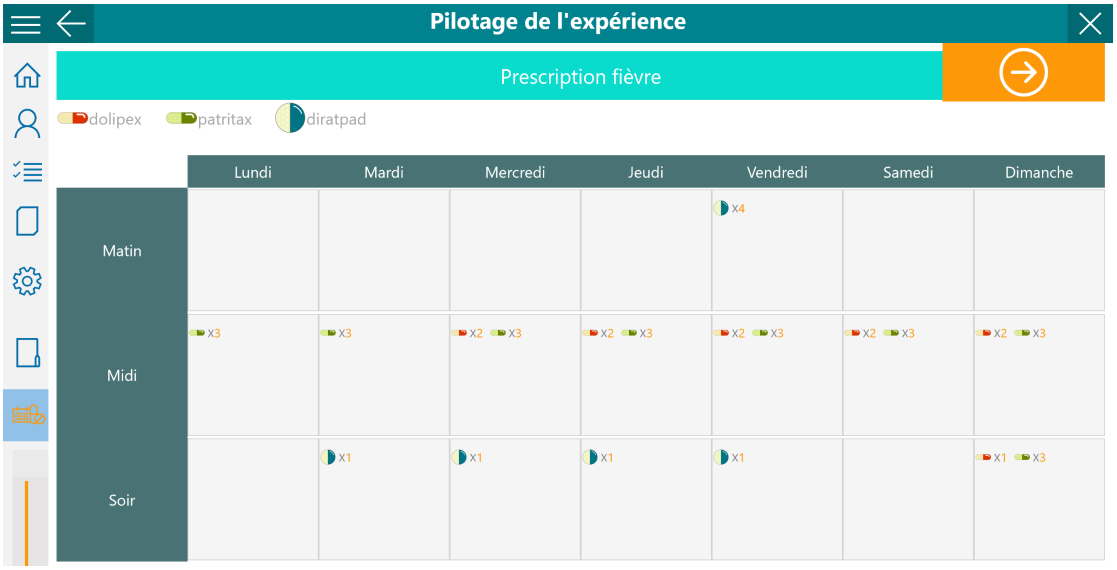


Figure 8.8: An example of medication prescription defined by a caregiver in a tabular format in the CONSIGNELA-Appli-R-V1.1.

appropriate compartment in the table and open it, then take the medicine and put it inside the compartment. After that, close the compartment. If the patient made a mistake during the filling of a medicine, he can select the medicine put incorrectly and throw it into the garbage can. As consequence the medicine will return to its original box

automatically. When patients finish their regimens, the pillbox extracts the actions and indicators of the Table 8.1 storing all of them in JSON format in a local file. Examples of such a JSON are shown in Section 8.2.3.

Figure 8.9: The virtual pillbox interface of the CONSIGNELA-Appli-R-V1.1.

The **browser** system was the MBA Browser (Appendix F) which has a natural language interface capable of receiving proactive behavior. We chose it because it is generic, i.e., we need only to customize dialogues to the SoS domain, and create a User Model (UM) for a domain user (in the case caregivers) receive proactive support (details in Chapter 5). Moreover, another point for our choice was that we have tested it successfully in the ACE4SD SoS.

Another system we selected for the MBA SoS was an agent-based system, namely "**Experience Analyzer**," to analyze information about medication experiences. This is an MAS system specially developed to analyze the list of indicators extracted from the CONSIGNELA-Appli-R-V1.1. The system was developed in Common Lisp through the OMAS platform, as we did for all other agents used in our approach (see Chapter 6). The analyzer system has four Service Agents (SA) each one with its functions (see Figure 8.10). There is a coordinator agent that receives input data in JSON and outputs results in the same format. The coordinator is also responsible for allocating and coordinating the tasks related to the analysis of experiences to the other three SAs. Furthermore, it can make requests external to the system, for instance, to receive data such as experience indicators used by the other SAs to perform their tasks.

For instance, when the MAS system receives a request for analyzing an experience and computing its accuracy, first the coordinator takes the data in JSON and translates to Lisp lists. Then, it breaks the data into three parts, i.e., one concerning the morning

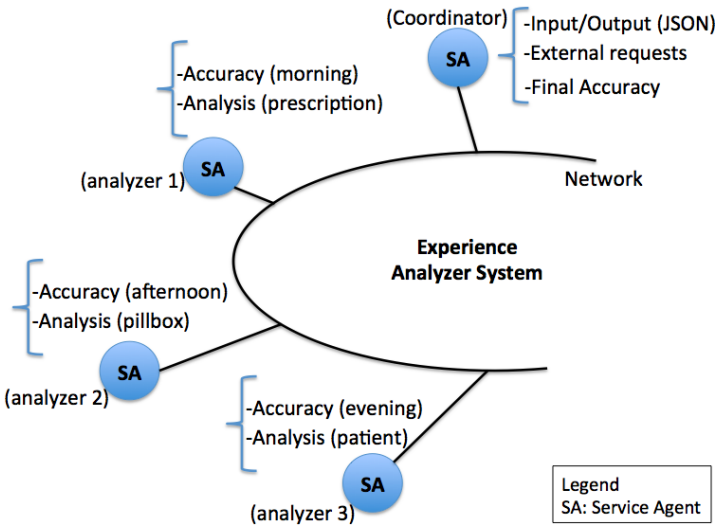


Figure 8.10: The Experience Analyzer system used to performs analyses in the medication experiences.

part of the pillbox, the second part concerning the afternoon part, and the last one concerning the evening part. After that, the coordinator distributes each piece to one of the other three agents. When the three agents receive the information they compute the accuracy (through the Equation 8.1) related to their parts. Once they finish the tasks, they send the answers to the coordinator agent that in turn merge them, computing the final accuracy value and outputting it in a JSON format.

Besides computing a specific part of the accuracy, each of the three agents is also coordinated for analyzing a specific kind of information concerning past experiences. One of them takes care of analysis concerning patients, another of prescriptions, and the last one of pillboxes.

The **knowledge platform** selected was MEMORAe which we have mentioned earlier in Chapter 6 when we used it as the triplestore of the ACE4SD SoS. For the CONSIGNELA-Appli-P SoS, the MEMORAe platform aimed to provide the collaborative environment for caregivers, and to perform Knowledge Management (KM) in the SoS.

The MEMORAe platform was conceived to support organizational memory. The platform was developed to be used in the context of a semantic learning organization in order to facilitate sharing and exchanging information. Through the platform, users can acquire new knowledge and skills by performing different tasks such as read reports, ask questions, share information, annotate documents, among others. In MEMORAe, each user of an organization has her own personal space used to store her resources like documents or images. Moreover, users can have common spaces for exchanging information and sharing their resources. All the resources stored in the platform are indexed by an ontology consisting of concepts and attributes describing the domain of

the organization. Users see such ontology as a knowledge map, interacting with it to retrieve or index resources.

Figure 8.11 shows a general view of the MEMORAe platform for a user that is logged in. In the middle of the platform there is the knowledge map of the organization containing, in this case, concepts such as "Experience," "Caregiver," and "Patient." The rightmost concept in dark orange is called "focus concept" and is the concept active, i.e., the last one clicked by the user. On the left, the user can select among the sharing spaces he has access, in the case, an organization space named "Consignela," and a personal space called "Medecin 1." Moreover, there is a number of external systems that can be used to add resources to her spaces. On the right of the platform, there is a panel for the active sharing space. Within such a panel there are resources indexed by the focus concept.

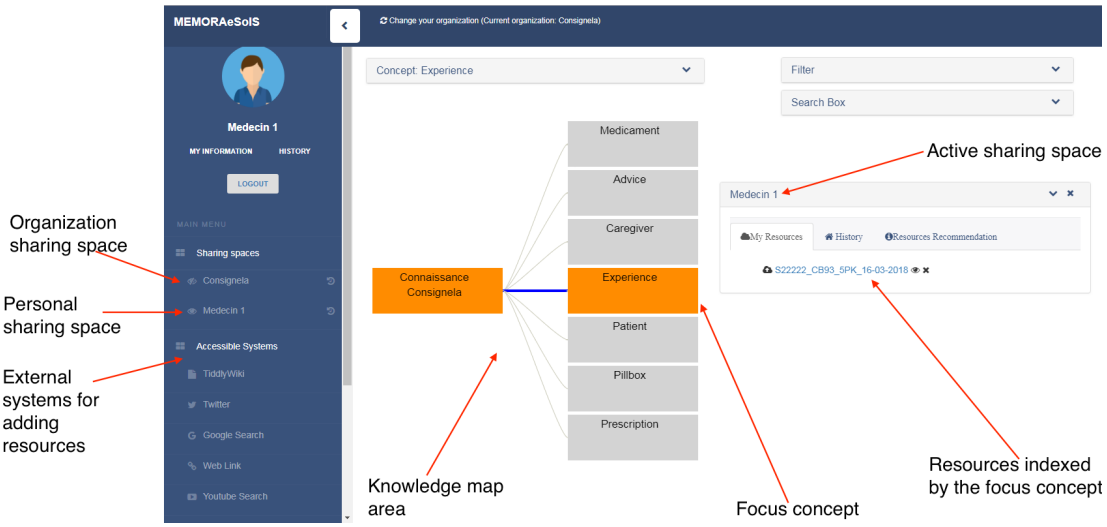


Figure 8.11: A general view of the MEMORAe platform.

The MEMORAe platform offers a number of functionalities for its users cooperate in a sharing space. In the Figure 8.12 we show an example of annotating (commenting) and rating a resource. The document titled "S22222_CB93_5PK_16-03-2018" and indexed by the concept "Experience" of the knowledge map is annotated with a comment, and rated with a note. Note that, the user can also add other sharing spaces for the resource, in order to share it with other users.

We chose MEMORAe as the knowledge platform for the SoS because it also met the four criteria above. For the SoS context it could provide a rich environment for caregiver collaboration and knowledge management. For the SoS functionalities, it could store synthesis of patient experience results and analyses, and also fetch annotations or recommendations associated with such synthesis, coming from different caregivers for supporting each other. Moreover, the MEMORAe platform was free and provided an API in HTTP using JSON format to access its functionalities. In addition, because it was developed by one of the research partners of the CONSIGNELA, it was well-known

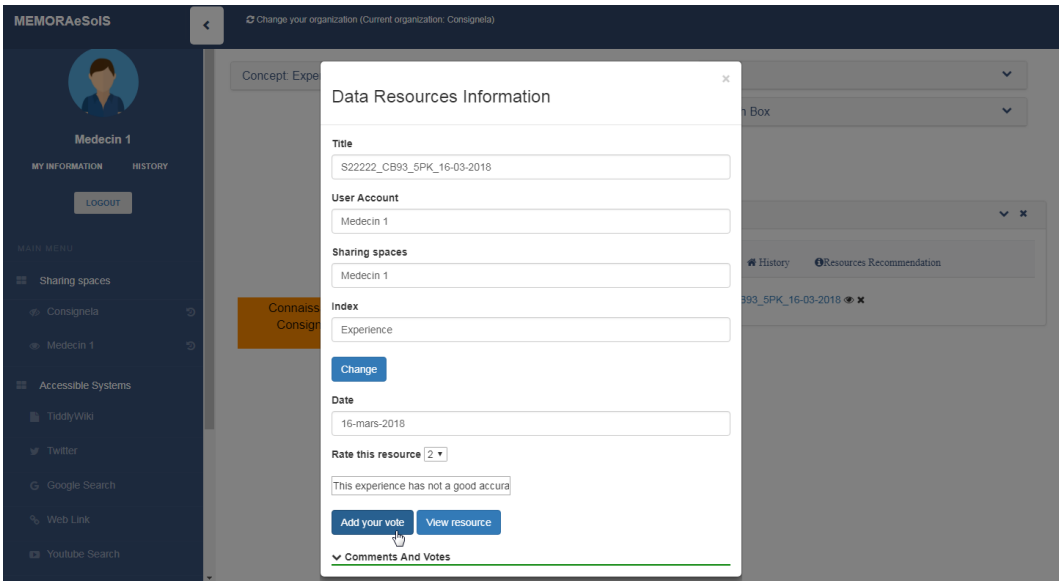


Figure 8.12: An example of annotating and rating a resource in the MEMORAe platform.

and recommended for supporting the collaboration among caregivers and performing the KM in the SoS.

The last system we selected was the MySQL **relational database** which acted as a memory in the SoS. The role of such a database was to support the knowledge capitalization and management in the SoS. The MySQL was responsible for storing and retrieving the raw information resulted from the patient experiences and the analyses performed on them. The choice for such a relational memory was that it is free and very easy to use. Moreover, we have already applied it in the ACE4SD SoS, thus we thought we could benefit from such an experience.

8.2.5 Guideline point: Select Memories

Because we have already selected in the previous step the MySQL database to act as a memory, we skipped this point of the guidelines.

8.2.6 Guideline point: Define SoS Users

For the CONSIGNELA-Appli-P SoS we defined the **patients** and the **caregivers** of the CONSIGNELA project as the SoS users. Patients interact with the SoS by providing the results and requesting analyses for the experiences they realized. Such results are the cooperation provided by patients and their pillboxes to achieve the SoS goal of ensuring the correct understanding of medication prescriptions. On the other hand, caregivers interact with the SoS in some ways. First, they cooperate with their patients by providing the experiences to realize. Moreover, they can ask their PAs to request

analyses of past experiences. In addition, they cooperate with each other by exchanging information of their experiences for ensuring the understanding of the prescriptions.

8.2.7 Guideline point: Extend the OntoMBA Ontology

The next point we followed in our guidelines was to extend the OntoMBA core ontology Appendix A for developing the global ontology of the CONSIGNELA-Appli-P SoS. Our basis to develop such ontology was mainly from modeling the CONSIGNELA-Appli-P domain through scenario described in Section 8.2.3. The resulting global ontology is shown in Figure 8.13. Different concepts were defined such as the organization as "CONSIGNELA," "Caregiver" and " Patient" extending the SoS user, the "Experience" associated with "Advice" and "Prescription" which contains "Medicine," the systems "Pillbox," "Browser," "Relational Database," "Experience Analyzer" and "Knowledge Platform."

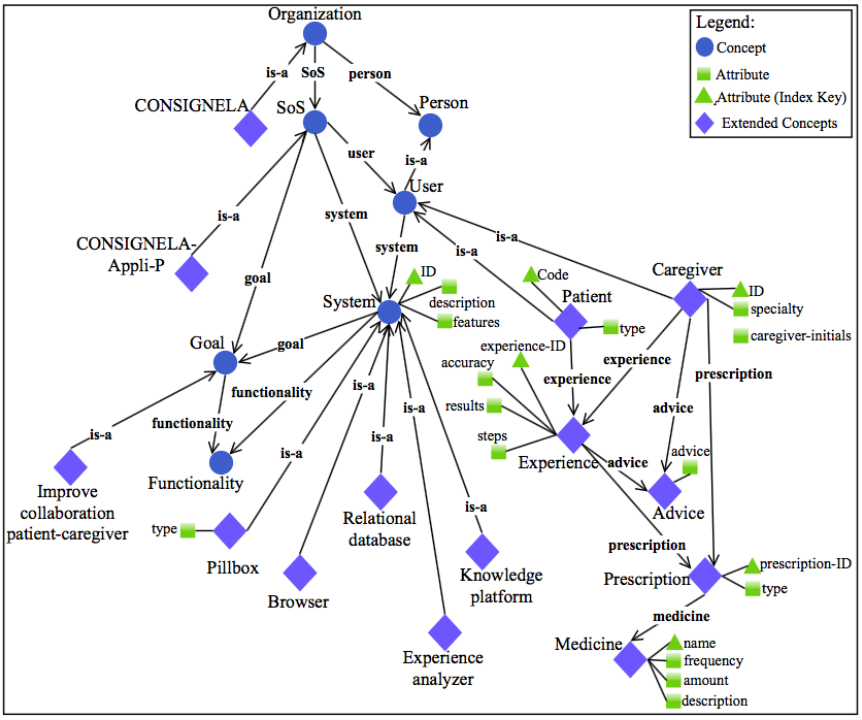


Figure 8.13: An extract of the CONSIGNELA-Appli-P global ontology.

8.2.8 Guideline point: Interface Systems - Proxy Agents

Next, the goal was to interface the selected constituent systems with proxy agents, in order to allow them exchange information and cooperate in the SoS. For interfacing the systems we performed the steps shown in the Figure 8.2 (Section 8.1). Moreover, we need to highlight that the agents used for interfacing the systems were from the OMAS platform and the ontologies we developed were implemented through the MOSS

framework, meaning that regarding agents and ontologies we used the same tools we did when developing the ACE4SD SoS (details in Chapter 6).

First, we connected to proxy agents all the selected systems that were not memories. We began by connecting the CONSIGNELA-Appli-R-V1.1 (virtual pillboxes), then the MBA Browser, next the Experience Analyzer MAS (experience analysis system), and after that MEMORAe (knowledge platform). For all of these systems we used the stub of proxy agent developed during the implementation of the ACE4SD (see Chapter 6). The goal was to try to facilitate our work by reusing proxy agents and reducing the efforts of implementation. For instance, such a stub already provided functions for translating message content from JSON, the format used by the systems above, to Lisp lists (and vice versa). Moreover, the stub also provided standard skills for sending and receiving work orders through a **virtual broker** approach. In addition, as the stub was an instance of the OMAS Transfer Agent it already provided means for connecting systems through different types of standards such as TCP, UDP and HTTP. Thus, by reusing such proxy agents we could facilitate our work and save time for developing the SoS.

For the connection between the **CONSIGNELA-Appli-R-V1.1 pillbox** and its proxy agent, we used TCP connection creating an endpoint in both sides pillboxes and agents. This was easy to do in the virtual pillbox, because it was full developed by research members of the CONSIGNELA and provided an API in C#. After that, we took the SoS global ontology developed in Section 8.2.7 and we derived an ontology for the proxy agent of the pillbox. In the ontology we added concepts related to the SoS functionalities the pillbox could request and provide according to the Table 8.3 of Section 8.2.3. In the case, we added a concept for the SoS functionality of "Analyze experience" (requester). Figure 8.14 shows an extract of the ontology derived for the proxy agent of the virtual pillbox.

Then, for each SoS functionality defined in the ontology we created a skill in the proxy agent to perform the required tasks. The names of the skills were the names of the functionalities in the ontology. For instance, in the proxy agent of the pillbox we created a skill named "Analyze experience" to receive the JSON data resulted from an experience and send a work order for analyzing it. That is, each time a patient touched "I finished" in the tablet of her pillbox, it sent through its endpoint a JSON similar to the one shown in the scenario described in Section 8.2.3.

After that, when the endpoint of the proxy agent received the JSON, it translated the messages to Lisp lists and distributed them to the correct agent skill ("Analyze experience"), based on the functionality name. The "Analyze experience" skill of the pillbox was very simple and just sent a work order (virtual broker) with the experience parameters to the patients' pillboxes to request the patients to begin an experience. An

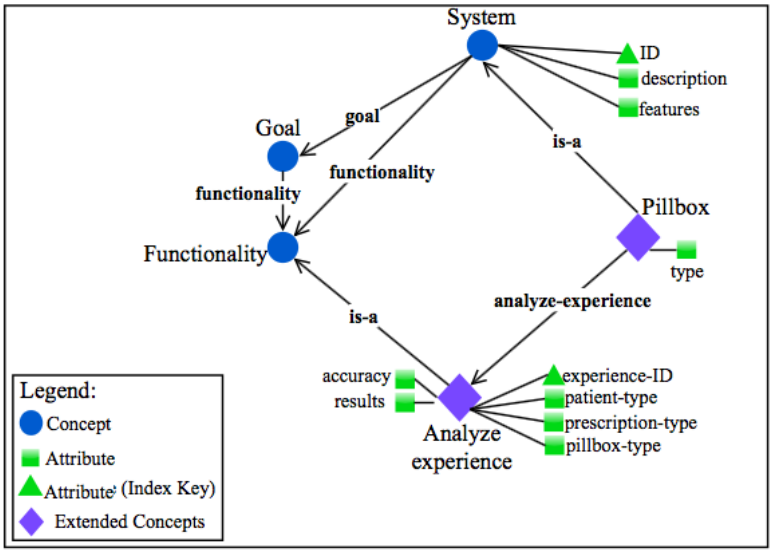


Figure 8.14: An extract of the ontology derived for the proxy agent of the virtual pillbox.

extract of the "Analyze experience" skill of the proxy agents of the pillbox is shown in Common Lisp as follows:

```
...
;; send a work order (Pillbox experience -> Analysis)
(setq message-content
      (append (list `("experience-lisp-list" ,experience-lisp-list))
              (list `("functionality" :analyze-experience))
              (list `("feature" :analyzer))
              (list `("experience-json" ,experience-json))
              (list `("type" :request))))

   (setq msg (make-instance 'omas::message
                            :name :send-work-order
                            :type :request
                            :from :PILLBOX-PROXY
                            :to :PILLBOX-PROXY
                            :action :SEND-WORK-ORDER
                            :args `(:data ,message-content)))
   (send-message msg)
...
```

The second system we connected with a proxy agent was the **MBA Browser**. First, we connected the PA, staff and the local server agents of the browser (see Appendix A) to the same LAN of the proxy agent. Then, because the MBA Browser is an MAS (PA, staff and local server) written in Common lisp through the OMAS platform the connection was made very easily, actually by just exchanging agent-to-agent messages directly. After that, we derived the proxy agent ontology from the SoS global one as we did for the virtual pillbox, but now adding the functionality concepts of "Analyze experience" (requester), "Fetch synthesis-recommendation" (requester), and "Show experience-results"

(provider). Next, we defined skills in the proxy agent for handling the tasks of such functionalities. Note that in this key point of the GAMBAD we do not add the dialogues or proactive behavior to the PA of the browser, as the focus is the interfacing with the proxy agent. We do that after in the "Integrate Proactive Interfaces" key point (see Section 8.1 for details about each key point).

The next system we connected with a proxy agent was the **Experience Analyzer MAS**. The connection with the proxy agent followed the same principle we applied for the MBA Browser, as the analyzer is also an MAS. Then, we derived the proxy agent ontology by adding the functionality concepts of "Analyze experience" (provider), "Store raw information" (requester), "Retrieve raw-information" (requester), "Store synthesis-results" (requester), and "Show experience-results" (requester). Finally, we defined skills for such functionalities in the coordinator agent of the MAS system (see Figure 8.2.4) and in the proxy agent, making them communicate for performing the needed tasks.

Next, we connected the **MEMORAe platform** with a proxy agent. Now, first we prepared the system by taking the SoS global ontology and adding it within MEMORAe for creating an organization (details in Section 8.2.4) named "CONSIGNELA", in order to allow caregivers of the project collaborate with each other by using the platform. Then, for connecting the platform with a proxy agent we did not create or keep an open connection between them, for instance, as we did for the pillboxes by using TCP. Actually, because MEMORAe provides an API in HTTP with the message content in JSON, we called directly its functionalities by using HTTP requests within the proxy agent. The next thing we did to fulfill the SoS requirements (see scenario of Section 8.2.3) was to create a function in the proxy agent for translating JSON data resulting from experience analysis to PDF. To do that we used the CL-PDF[8] library which is free and available for Common Lisp. The goal was to create the PDFs synthesizing the experiences for storing and indexing them in MEMORAe.

After that, we derived the ontology of the MEMORAe proxy agent, by taking the SoS global ontology and adding the functionality concepts of "Store synthesis-results" (provider) and "Fetch synthesis-recommendation" (provider). After that, we defined skills for such functionalities in the proxy agent, and then in the skills we used HTTP requests to call the MEMORAe API for handling its functionalities.

Next, after connecting the systems above, the last system we interfaced with proxy agents was the SoS memory, i.e., the **MySQL database**. The first step we did was to prepare the system by setting up and configuring a new database schema to store data about patient experiences and analyses. We created relational tables with names being concepts and their columns being attributes of the SoS global ontology. For instance, we created the table "experience" containing columns such as "ID," "patient_type," "accuracy," or "results." Then, our goal was to connect it to the proxy agent focused

[8]See `http://www.fractalconcept.com/asp/cl-pdf` for more information.

on a generic memory interface which we developed in the ACE4SD and incorporated in our approach (see Chapter 4 and Chapter 6). However, because we have already used the MySQL database as a memory of the ACE4SD, we could also reuse much more of the memory proxy agent interface. For instance, we had already available the syntax for the SQL statements within the proxy agent. Thus our work was simplified to define the structure of the skills for storing and retrieving raw information of experience results and analyses.

Once the SoS systems were interfaced with proxy agents, they were able to exchange information and cooperate. Note that, we did not implement any broker agent here. This is because we used a virtual broker approach for handling the work orders exchanged between systems. By doing that, we saved time and computational resources for developing the SoS, because we did not need to implement the brokers physically.

Next, we stored the SoS global ontology in the MySQL proxy agent. Moreover, we also derived a second ontology from it defining the SoS functionalities related to the database, and storing it along with the global ontology. For the second ontology we defined the functionality concepts of "Store raw information" (provider), "Retrieve raw-information" (provider). Finally, after that we defined and created the skills for such functionalities in the proxy agent.

8.2.9 Guideline point: Integrate Proactive Interfaces

In the CONSIGNELA-Appli-P SoS we used MBA Browsers as proactive interfaces for caregivers. The goal of such interfaces was to support caregivers to request analyses for patient experiences, and to act proactively for providing useful information such as past experiences or recommendations from other caregivers to improve cooperation and ensure the understanding of the prescriptions by their patients.

To integrate the browsers we followed the steps of the Figure 8.3 (Section 8.1). However, note that because in the CONSIGNELA-Appli-P the MBA Browser was one of the systems selected in the earlier key points of the GAMBAD, we have already performed some steps such as "Use MBA Browser," "Connect Local Server," and "Connect PA and Staff" in the previous key point. Thus, we started this key point with the "Define Skills" step of the Figure 8.3.

Regarding the skills, first we needed to state that for the CONSIGNELA-Appli-P we did not need to define skills in the PA for handling the messages from/to the local server and the proxy agent, as we had already done it in the when we used the MBA Browser in the ACE4SD (see Section 6.4.3 of Chapter 6). Thus, we implemented a first skill in the PA concerning the SoS functionality "Show experience-results," for receiving the JSON data (after passing through the browser proxy agent) sent by the Experience Analyzer

MAS (see Section 8.2.3), containing the analysis results for an experience done by a patient.

The skill extracted data from the JSON such as the accuracy of the experience, and the patient type or age. Then, it created a synthesis of the information in natural language which was presented by the PA to the caregiver. Figure 8.15) shows an example of such a synthesis in natural language.

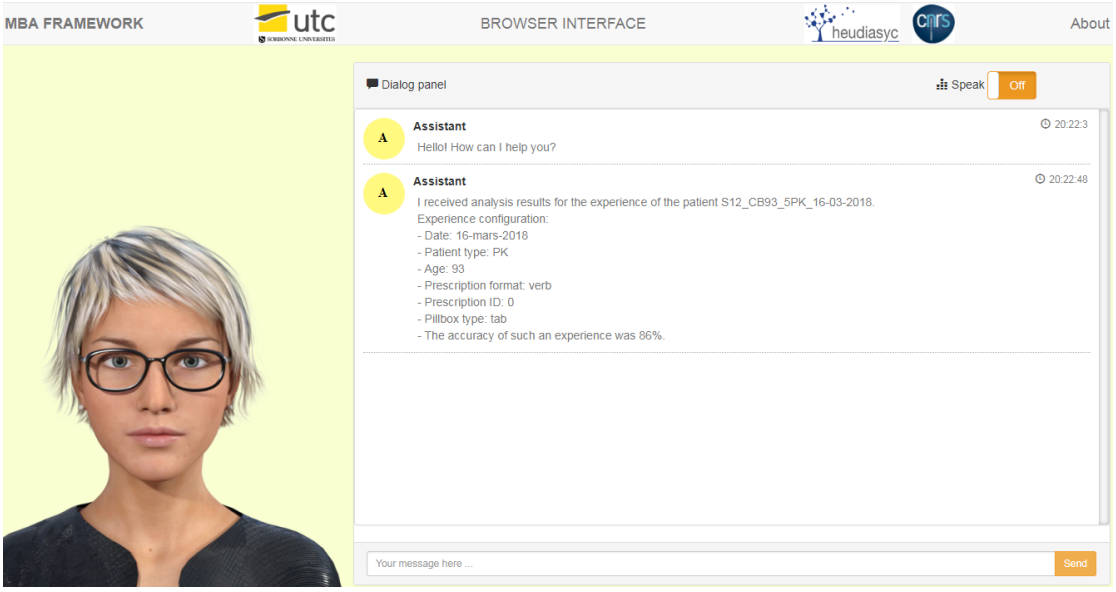


Figure 8.15: The synthesis created by the PA to its caregiver after receiving analysis results for a request of experience.

Moreover, in the skill we also tested the accuracy of the experience for allowing the PA to act proactively to support the caregiver. To implement the PA proactivity, we followed the ProPA method (Chapter 5). First, we took the ontology of the proxy agent interfacing the MBA Browser and derived a User Model (UM), in this case just by adding a single concept "score" for the caregiver. Then, differently from the scores used in the ACE4SD which were dynamic (see Chapter 6), here we used a fixed score limited by a threshold of 60% which was defined by project team members. For instance, if the accuracy of the experience was below the threshold of 60%, then the PA started to act proactively for supporting the caregiver.

When acting proactively, the PA tried to fetch caregivers' comments, i.e., annotations associated with past experiences kept in the MEMORAe platform. To fetch such an information the PA sent a work order requesting the SoS functionality "Fetch synthesis-recommendation" along with indicators of Table 8.1 (Section 8.2.3) such as "prescription-type," "patient-type," "date," or "age." In MEMORAe, the criterion for picking an annotation or comment was based on the rating gave by caregivers. We focused on picking the one with the best rating. On the other hand, in case of only a single comment was available, then we fetched it directly from the platform. Figure 8.16

shows an example of the proactivity of the PA by bringing its caregiver comments after the results of a patient experience.

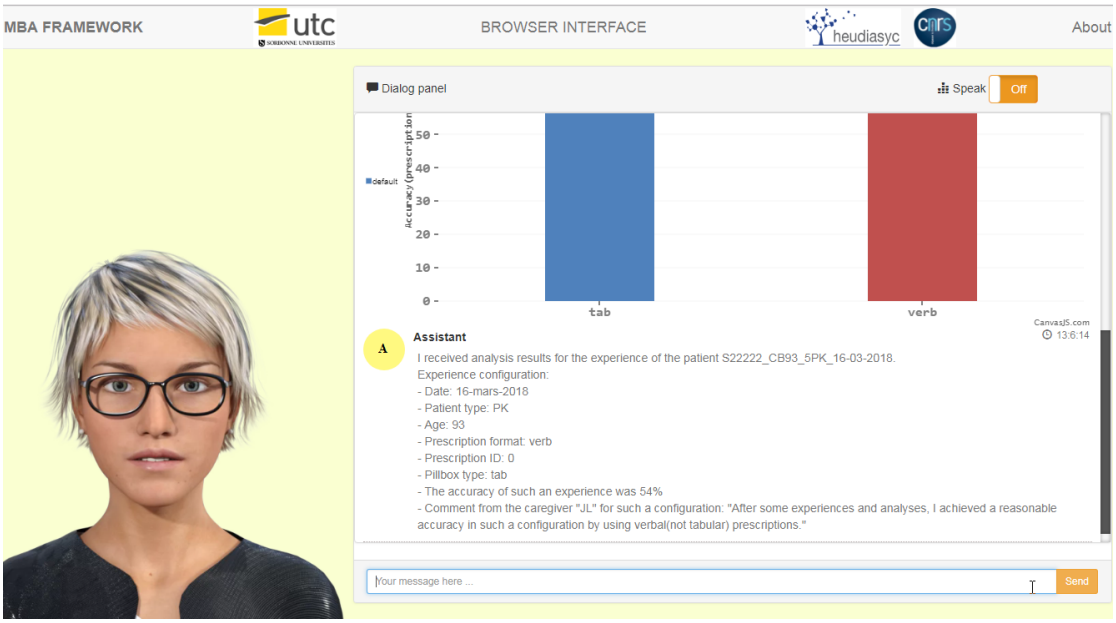


Figure 8.16: The proactivity of a PA shown by bringing comments to its caregiver after the results of a patient experience.

Then, the last step was to develop the PA dialogues for the interaction with the caregiver. We implemented the simple dialogue shown in Section 8.2.3 for handling caregivers' requests for analyzing past experiences of their patients. To develop the dialogue we used the OMAS dialogue manager which we have applied successfully for implementing the dialogues in the ACE4SD (Chapter 6). The result of such a kind of request for analyzing past experiences was a chart such as the one shown in the example of the Figure 8.17.

8.3 Discussion

We have started this chapter by presenting our guidelines for developing MBA SoS. Such guidelines were the result of lessons learned from the development of the ACE4SD SoS along with the MBA architecture. Our main goal in providing the guidelines is to facilitate the work of SoS architects by serving them with a path to follow when developing MBA SoS.

When defining the guidelines we aimed to cover all the requirements needed for building MBA SoS. The results of our reflections and experience has culminated in eight key points ranging from the SoS goal to proactive user interfaces. We ended up presenting the key points of our guidelines in a logical sequence through a very simple method used for guiding SoS architects.

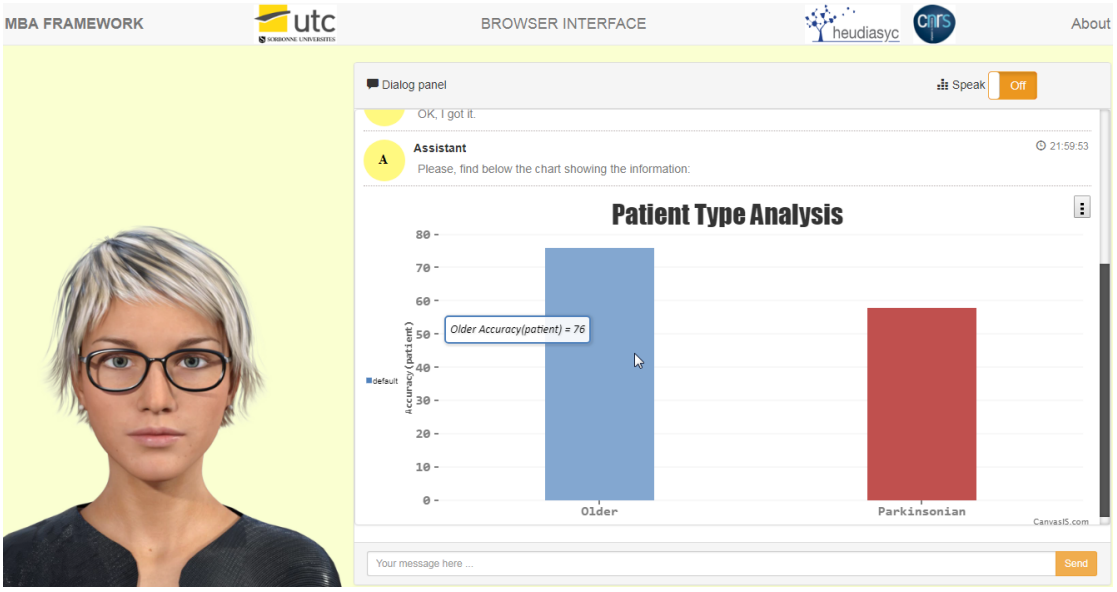


Figure 8.17: A chart showing the results for an analysis request concerning the type of patients.

To test and show the results of using the proposed guidelines, we have used them for building a brand new MBA SoS in the domain of Health Care. Through such a new experience we have drawn a number of conclusions.

Comparing to the ACE4SD SoS which was developed without the guidelines, in the Health Care SoS we could have been much more accurate when performing the tasks required for the development. For instance, because we did not have a clear sequence of steps to follow in the ACE4SD, we defined the SoS ontology at the wrong time, and thus we were obliged to modify and update it many times during the SoS development. On the other hand, for the Health Care SoS we did not experience such a problem, actually we were precise to define the ontology at the correct time, and this allowed us to avoid redoing unnecessary tasks and wasting time.

Moreover, when using the guidelines we were much more focused on the correct tasks at each time. For example, during the development of the ACE4SD we tried to integrate proactive interfaces at the same time we were still interfacing systems with proxy agents. However, this was not a good idea because we found proactive interfaces may rely on the skills of proxy agents for sending and receiving work orders, or derive their ontologies, i.e., User Models, from the proxy agents one. For the Health Care SoS we did not suffer from such a problem, as the guidelines prescribed to interface proxy agents before integrating proactive interfaces.

Furthermore, the expansions of the guideline key points “Interface Systems - Proxy agents” and “Integrate Proactive Interfaces” were very helpful when developing the second SoS. Because these two key points are complex, the fact of providing expansions

showing their steps in sequence has kept us on the right track focusing on the right tasks to do.

In addition to show the benefits of using our guidelines, our goal for developing the Health Care SoS was also to validate the generic feature of the MBA architecture, testing its potential to be applied to different domains. Developing the Health Care SoS helped us to understand how well the MBA architecture adapts and facilitates the development of SoS in distinct domains.

For instance, for adapting the MBA architecture from the collaborative software development to the Health Care domain we needed to:

1. adapt ontologies.
2. interface systems with proxy agents.
3. integrate proactive interfaces, as an option.

The first change needed by the MBA architecture for building an SoS from a distinct domain concerned in developing an ontology modeling it. The effort required for developing such a "global" SoS ontology depends on the domain, or on the number of concepts and attributes needed. However, because our approach already provides a starting point for developing such ontology through the OntoMBA core, the work may be facilitated for SoS architects.

Moreover, we had to interface some systems with proxy agents. However, when performing such a task we were well served and supported by the MBA architecture. For instance, our approach already provides a generic stub of proxy agent resulted from the development of the ACE4SD, and consisting of several facilities such as means for connecting systems through different network standards, translation of message content defined in JSON standard, and skills for handling work orders. Thus, in the Health Care SoS we reused such a stub for interfacing the SoS systems, making our work much easier when compared to the development of the ACE4SD.

When interfacing the systems in the Health Care SoS, our efforts were mainly devoted to implement the skills of the proxy agents for handling the SoS functionalities. Besides, we had to develop ontologies for the proxy agents to handle idiosyncrasies, however such a task was not a big issue because such ontologies were derived from the global SoS one, by just adding concepts related to the SoS functionalities of each system.

A substantial support provided by the MBA architecture was for interfacing the MySQL memory in the Health Care SoS. Here, we benefited from reusing the MySQL proxy agent developed for the ACE4SD. The four basic DB operations of inserting, removing, updating, and selecting using the SQL syntax were already implemented. Thus, our

efforts were focused just in deriving the agent ontology, and adapting the agent skills for storing and retrieving data according to the requirements of the new domain.

Although integrating proactive interfaces are optional in our approach, the MBA Browsers were part of the systems selected to be constituents of the Health Care SoS. Thus, we will state the main changes we made (comparing to the ACE4SD) for adding the proactive interfaces to the new SoS. First, because MBA Browsers are generic SoS user interfaces (Chapter 5; Appendix F), we could reuse it in the new SoS. Using the MBA Browser facilitated carrying SoS information to caregivers, because we could avoid spending efforts in augmenting systems, for instance, by adding new windows. Second, we only needed to implement the skills and dialogues for handling the SoS functionalities and PA interactions required by the caregiver. However, developing dialogues is not an easy task (Chapter 6), but we benefited from a good experience we had when developing the ACE4SD SoS. For implementing the dialogues of the new SoS, we have also used the dialogue manager of the OMAS platform which is the same used to implement our agents. Using the dialogue manager has facilitated our work, letting us concentrated much more on implementing the content of the dialogues than their structures. Finally, for the proactive interfaces we also derived an ontology (User Model), now from the proxy agent one and just by adding a single concept for supporting the proactive behavior of the PA.

Handling work orders in the Health Care SoS was made through a virtual broker approach. This was also a result from our experience with the ACE4SD. By using such an approach we avoided implementing brokers physically, thus reducing implementation efforts and computational resources.

Using our guidelines for developing a new MBA SoS has facilitated our work in a number of ways. Our goal is to provide such results for guiding and reducing the efforts of SoS architects when building MBA SoS. Moreover, developing a brand new MBA SoS in a different domain has validated the generic feature of our approach, showing its potential to be applied to distinct domains. Finally, by generalizing our architecture we could move it towards a framework for building SoS.

Chapter 9

Conclusion and future research

The research in this thesis was done in the context of Systems of Systems (SoS) approached from a systems engineering perspective. It was motivated by the observation that although SoS have become quite popular and have received an increasing amount of attention, most of them are still handcrafted through ad hoc approaches.

Developing SoS using ad hoc approaches is an arduous task for SoS architects. Thus, in this research we designed a domain-independent framework for facilitating the development of an SoS. Our framework is based on a novel architecture we call MBA for Memory-Broker-Agent, consisting of agents, brokers, and memories. The role of agents, or proxy agents as we defined in this research, is to interface a constituent system in an SoS, allowing it to exchange information and cooperate with other constituent systems. Brokers are intermediaries that receive requests for functionalities (or services), try to find providers, and then transfer results back to the callers. Memories are used to capitalize and manage knowledge during SoS operation. Considering such elements for supporting our architecture was not a random choice.

The first element we chose was a Multi-agent system (MAS), which is a key element of our architecture. Although MAS are different from SoS as we stated in this thesis, they also share positive common features due to their cooperative, distributive or evolutionary nature. Thus we decided to add an MAS layer in our architecture and to build the SoS on top of it. Our goal in using such a layer was to agentify the SoS elements and interactions, standardizing their heterogeneous environment.

The second element we added to our architecture were brokers. The role of such intermediaries was to take care of organizing exchanges such as requests for SoS functionalities or transferring results among constituent systems. The main goal that led us using brokers was to avoid connecting constituent systems directly to each other, thus providing a loose coupling among them.

Finally, we decided to add memories to our architecture. The main reason was because there is a strong exchange of information in SoS, resulting from the cooperation among constituent systems. Thus, in our mind it would be important to provide means for capitalizing and managing knowledge, allowing it to be reused during SoS operations.

Main contributions

The main contributions of this research are:

- A domain-independent MBA (Memory-Broker-Agent) framework based on a novel architecture for facilitating the development of Systems of Systems (SoS).
- An original method for building SoS which consists of eight key points describing general guidelines for supporting SoS architects to develop MBA SoS accurately.

Building our first SoS and functional prototype which was the ACE4SD in the domain of collaborative software development allowed us to perform tests and experiments, in order to determine how we could support architects for developing SoS. The results of our experiments with the ACE4SD culminated in the design of the MBA architecture, and later in supporting us to move it towards a framework. Some relevant contributions the ACE4SD SoS brought to our approach were that (i) we discovered we could replace physical broker agents by a virtual broker; and (ii) the development of a proxy agent providing a generic memory interface.

The virtual broker was the result of using conditional addressing allowing to deliver messages only to agents that satisfy certain conditions formulated using the sender ontology and the ontology query structure. The main advantage of using a virtual broker was that the MBA architecture could keep a pure P2P among constituent systems, as it was no longer necessary to have intermediaries implemented physically.

The generic memory interface resulted of implementing and testing proxy agents with different kinds of databases. The main advantage of such an interface is that it provides a structure with an Abstract Statement for handling basic database operations. The result was that we incorporated such an interface to our architecture offering it to facilitate the development of an SoS.

Moreover, the role played by the ACE4SD in our research was even more significant, i.e., it was also used to validate and evaluate the approach proposed in this research. Based on the ACE4SD SoS we evaluated our approach to verify if the MBA architecture could facilitate the development of SoS, regarding the difficulty and effort required when building the interfaces to connect the constituent systems of an SoS. Moreover, in such an evaluation we also considered if the interfaces built through our approach were easier to

be modified or adapted to changes, in order to answer the evolutionary nature of SoS. The results of the evaluation proved that the MBA architecture makes easier building SoS, being less difficulty and demanding less effort when compared to traditional handcrafted approaches. Moreover, our results demonstrated that the MBA architecture supports the evolutionary nature of SoS, building interfaces easier to reuse and modify. Through this evaluation we could prove the hypothesis stated in this research that: *a framework based on a Memory-Broker-Agent architecture could facilitate the development of an SoS, regarding the difficulty and effort needed for building the interfaces to connect its constituent systems.*

Moreover, other noteworthy results we achieved in this research were that our MBA architecture can answer the SoS challenges of interoperability, evolution, robustness, knowledge management, and user dimension (detailed in Chapter 2). Following, we describe how the architecture of our framework answers such challenges.

- Interoperability is provided by the MAS layer and by interfacing constituent systems with proxy agents. This allows constituents use Work Orders based on conditional addressing which benefits from ontologies, performing requests and finding providers semantically.
- Evolution is favored by the loose coupling among constituent systems provided by a virtual broker which is handled by Work Orders, and then letting systems in a pure P2P.
- Robustness is improved by the use of Contract-Net, which allows inserting new components in parallel to existing ones. Moreover, robustness is also supported by using a virtual broker providing loose coupling, and making the SoS not relying on centralized components. Finally, the synchronization among constituent systems may be used to let an SoS coherent after its constituent systems go down.
- Knowledge management must be implemented in the framework of each application but can rely on the generic memory interface provided by our architecture.
- User dimension is taken into account by the possibility of interfacing users to the SoS through Personal Assistant agents, conducting exchanges using natural language in a textual or vocal mode. Personal assistants can be proactive and make use of user profiles to better customize interactions.

The experience of building the ACE4SD SoS has led us to gather key points and create a method called *GAMBAD* for giving directions to develop MBA SoS. The method is our second contribution in this research, and it consists of eight key points describing general guidelines for developing MBA SoS. The goal of the GAMBAD is to guide SoS architects with a logical sequence of steps when building MBA SoS, independent of the domain it is applied.

In this research we used our GAMBAD method for building a brand new SoS in the domain of Health Care. We built an SoS in a French regional project, named CONSIGNELA, aiming to improve the collaboration between patients and caregivers for ensuring the correctly understanding of medication prescriptions. Our goal with that was to validate the generic feature of our MBA architecture, showing its potential to be applied to different domains. By building this new SoS we could see in practice the usefulness of our guidelines. For instance, comparing to the ACE4SD which was developed without such a support, we were much more accurate in the tasks needed to be done at the right time. Thus, we avoided redoing work and we saved time, keeping our efforts in developing the relevant tasks.

Finally, applying the MBA architecture to a new domain allowed us to move our approach towards a generic framework for building SoS.

Future research

There are many possible directions for future work in this research. For instance, the SoS we have built until now consisted of a modest number of systems. It would be interesting to test our MBA framework using a larger number of constituent systems in the SoS built. Moreover, testing our approach in new problems and domains is also desirable as each domain has its idiosyncrasies. By doing that, we plan to find possible gaps, and then perform the needed corrections and improvements in our framework.

Moreover, enhancements can be made for the current extended functionalities provided by our approach. For example, at present in the proactive interfaces an SoS architect needs to define a specific score for the users interacting with PAs. In the future, our goal is to define a generic score as an option for supporting SoS architects to save time when integrating the proactive interfaces. Furthermore, for the functionality of synchronization among constituent systems, although implemented was not fully tested. To this end, we have already proposed a method for testing our synchronization (details in Appendix B), and now we would like to conduct intensive tests.

In addition, currently our approach is only capable of providing means for supporting Knowledge Management (KM), for instance, through a generic memory interface. Nonetheless, in future we intend to provide a new feature or a truly integrated system in our approach for allowing it to manage knowledge autonomously, i.e., without being dependent on third-party systems. This would actually be an option for SoS architects, i.e., they will also be able to continue using their own means for KM. However, our goal is to facilitate the tasks and reduce the efforts of the architects, already providing such a means.

Finally, we aim to move our MBA framework towards a generic platform for developing SoS. Our goal is to provide a full environment for building MBA SoS on the top of a Multi-agent layer. In that way, architects will be better supported, focusing and concentrating their efforts on developing their SoS, instead of consuming endeavors with more technical issues.

Appendix A

OntoMBA: The MBA core ontology

In this appendix we show the OntoMBA ontology provided by the MBA framework to support SoS architects when developing the ontologies in an MBA SoS. The goal of the OntoMBA is to be used as a point of departure by SoS architects when developing the ontologies of an MBA SoS. Thus, the OntoMBA is a core ontology and intends to be extended and customized according to the domain, requirements and specifications of the SoS being developed. Figure A.1 shows the OntoMBA ontology.

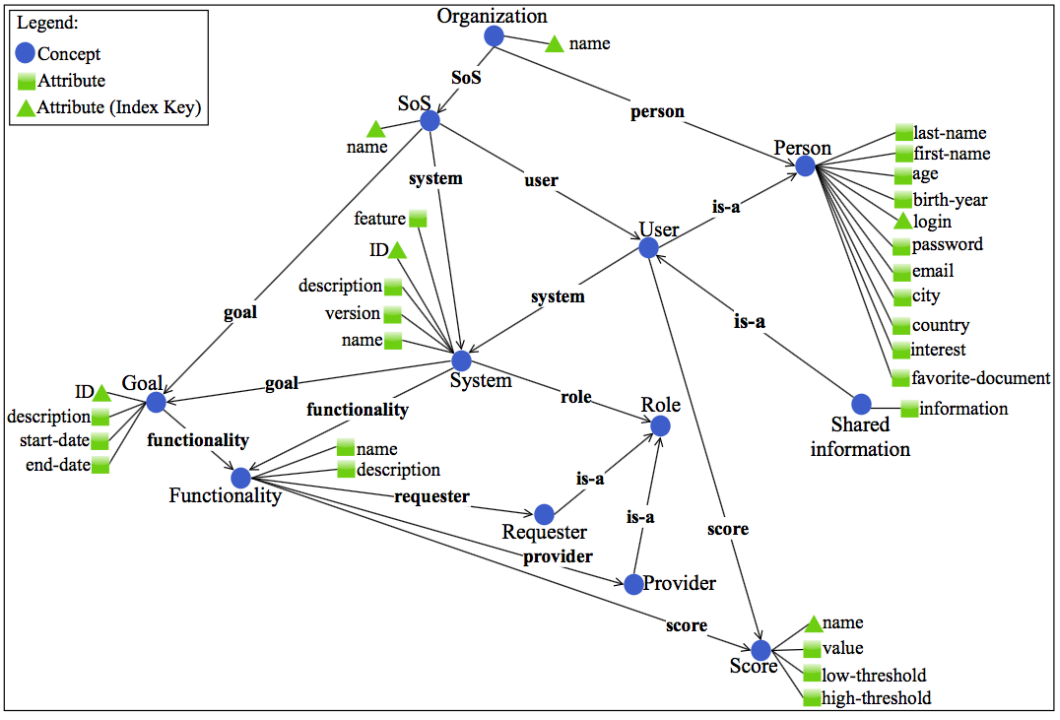


Figure A.1: The OntoMBA core ontology.

Appendix B

Synchronization among MBA systems

This appendix details the synchronization among constituent systems proposed by the MBA framework. The goal of the synchronization is to support the robustness of an MBA SoS, trying to let it coherent after its constituent systems go down. This can be important as once systems are reconnected to the SoS their state may not be anymore consistent with the rest of the SoS, and thus information provided by them can be incoherent and spread over all the SoS. Having said that, we need to state that the synchronization provided by our approach is intended to be used for systems that change their internal states when cooperating in an SoS. For instance, a database could be an example of such a kind of system as it inserts or removes data, changing its internal state.

B.1 Example of MBA SoS

To better understand the synchronization provided by the MBA framework, we use the very simple SoS shown in Figure B.1. This example of SoS concerns the domain of collaborative software development, and it consists of three constituent systems: a database based on key-value data model, a database of relational data model, and a source code versioning system. The two databases are considered legacy systems offering the same SoS functionalities for storing and retrieving meta-data information about code versions such as the name of the developer who created it, the date, or the code version ID. Conversely, the versioning system is more specialized and keeps full structures of the different versions of code produced by teams during software development.

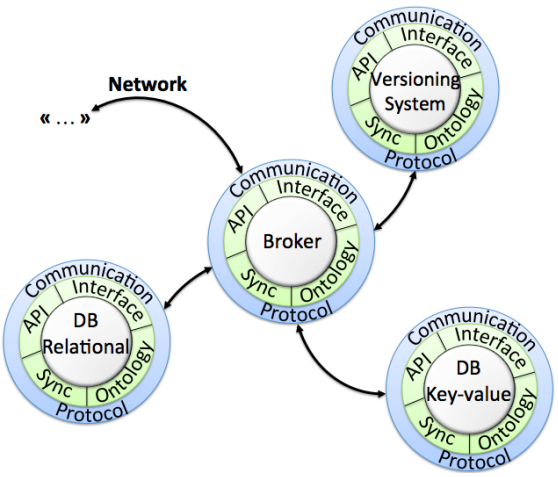


Figure B.1: The SoS used as example for the MBA synchronization.

B.2 ExAS: The Extended Abstract Statement

In this section we present an important element of our synchronization called *Extended Abstract Statement*. This is actually the Abstract Statement from our generic memory interface (see Chapter 4) extended by a *label*. To recall, an Abstract Statement is a generic Database (DB) statement for performing four basic operations such as insert, remove, update, and select data. The goal of a label is to define a unique identifier for an Abstract Statement. For instance, when synchronization is used in an MBA SoS like in our example above (Section B.1), the systems that change their internal states (like the DBs) receive work orders with a label appended to their message content. The label consists of who sent the work order (*from*), and the *universal time* captured at the moment it was sent. Then, when such systems execute an Abstract Statement, the label received is used as a new parameter of it. Formally, the Extended Abstract Statement (ExAS) can be represented by the following EBNF:

```
<Extended Abstract Statement> ::= <label> <operator> <class> <data>
<label> ::= <from> . <universal-time>
<operator> ::= <insert> | <update> | <remove> | <select>
<data> ::= {<property> (<value> | {<abstract-statement>})}
```

B.3 SyncMBA: A method for synchronizing MBA systems

In this section we present the *SyncMBA* which is a method for performing the synchronization among MBA systems. The method is performed according to the following assumptions.

- The synchronization is used by systems that change their internal states when cooperating with an SoS.
- For each system intended to be synchronized in an SoS, there is at least one other system in such an SoS offering the same SoS functionalities it does.
- Each system requiring to use the synchronization need to enable a "synchronization goal" in their proxy agent interface.

Figure B.2 shows the SyncMBA which consists of two activities, named "Run Normal Operation" and "Synchronize."

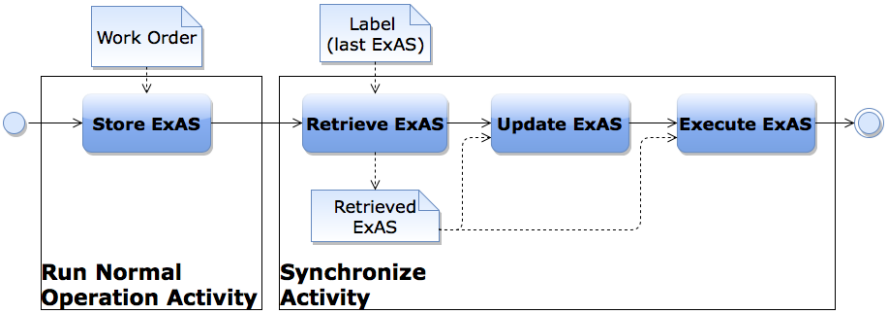


Figure B.2: The SyncMBA method used for the synchronization among constituent systems of an MBA SoS.

The "Run Normal Operation" activity describes the steps performed by the synchronization when the constituent systems that change their internal state are in operation, i.e., not down. The activity consists of a single step, named "Store ExAS," which receives a work order as input. In this step, after receiving a work order, the changing state systems store in their proxy agents an Extended Abstract Statement (ExAS) each time a normal Abstract Statement is executed. The goal is to use such ExAS after in the SyncMBA when synchronizing the constituent systems. To give an example, when the key-value DB of our example (Section B.1) receives a work order for storing metadata about code version, it executes an Abstract Statement for performing such a task, and then it also stores an ExAS (with the label parameter received) in the proxy agent interfacing the DB. The following EBNF could represent such an ExAS stored in the proxy agent of the key-value DB.

```
<Extended Abstract Statement> ::= <label> <operator> <class> <data>
<label> ::= "Versioning System" . "3724172590"
<operator> ::= "Insert"
<class> ::= "Meta-data"
<data> ::= {"Version-ID" "0.2" ...}
```

The next activity of the SyncMBA, named "Synchronize," intends to synchronize constituent systems reconnected to the SoS after going down. The activity consists of the steps "Retrieve ExAS," "Update ExAS" and "Execute ExAS."

The "Retrieve ExAS" step aims to let constituents that went down retrieve ExAS from constituents which stayed in operation during the same time. The input of this step is the label of the last ExAS received by a constituent that was down. The goal is to support the constituent which was reconnected to the SoS to receive the ExAS it lost during the time it was down. To do that, when the constituent is reconnected its proxy agent triggers a goal (what it plans to do) which sends a work order to all the constituents offering the same SoS functionalities its system does. This means that all constituents requiring to use the synchronization need to enable this goal in their proxy agent interface (see our assumptions for the SyncMBA).

The work order sent in the goal uses conditional addressing allowing to deliver messages only to agents that satisfy certain conditions (see *Real broker vs. Virtual broker* in Chapter 6 for details). The work order contains the label of the last ExAS received by the system that was down, and it requests all the ExAS from the time associated to such a label until the current universal time. When the proxy agents of the possible providers for such ExAS receive the request, they process it ordering chronologically the ExAS by the universal time associated to their labels, and then they send the results to the proxy agent of the requester.

To illustrate, assuming that the key-value DB went down for 5 minutes. Then, when it was reconnected to the SoS its proxy agent sent a work order, containing the label of the last ExAS received along with the current universal time, to all the constituents that were "database." In the case, the only one capable of receiving such a work order was the relational DB. Once, the proxy agent of the relational DB has received the work order, first it retrieved and ordered chronologically the ExAS required. Then, it sent the ExAS to the proxy agent of the key-value DB. Note that, if there are more than one system to retrieve the ExAS, than the proxy agent of the requester will take the first answer (see *work order protocol* in Chapter 4). The output of this step of the "Synchronize" activity are the ExAS received by the proxy agent requester.

The next step is "Update ExAS" which aims to update the ExAS contained in the proxy agent of a system that went down. In the "Update ExAS" step, the proxy agent takes the ExAS retrieved in the previous step, and then it appends them to the end of its ExAS. The result after that is the proxy agent holding a chronological up to date list of ExAS.

The last step of the "Synchronize" activity and the SyncMBA method is named "Execute ExAS." In this step, the goal is to take the ExAS outputted in the previous step, and then execute each one of them in chronological order for synchronizing a constituent

that was down. For instance, the proxy agent of the key-value DB will execute chronologically each one of the ExAS received. The expected result after doing that is the DB synchronized and coherent with the current state of the SoS.

B.4 Example of synchronization

The sequence diagram of the Figure B.3 shows the example we described above in Section B.3 for the 5 minutes down faced by the key-value DB.

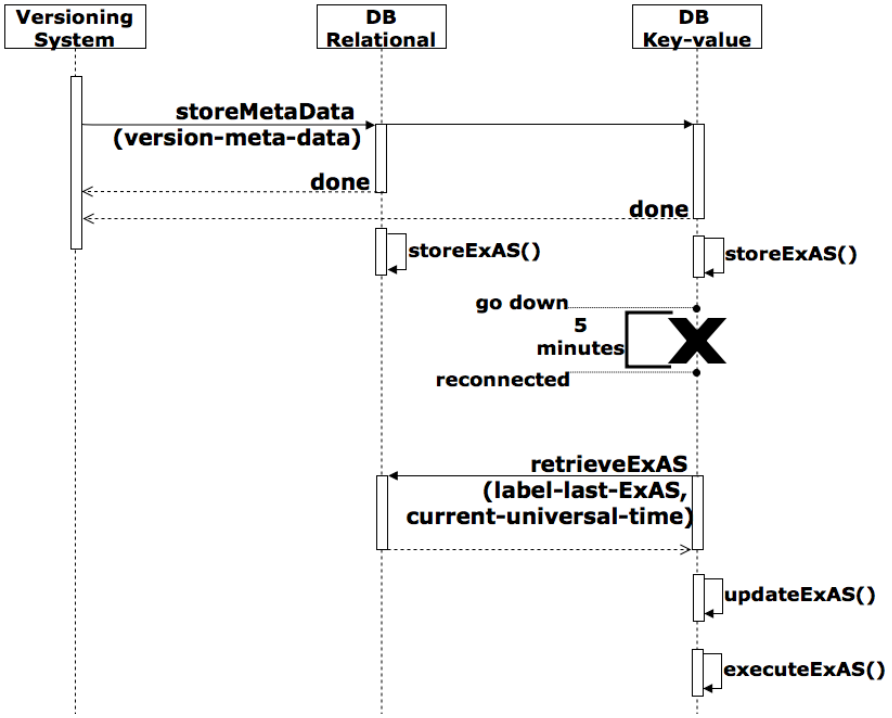


Figure B.3: A sequence diagram to illustrate the synchronization among constituent systems.

B.5 Implementation

We used the following tools for building the MBA SoS of the example described in Section B.1:

- Redis[1] as the key-value database.
- MySQL[2] as the relational database.
- GitHub[3] as the code versioning system.

[1]See `https://redis.io/` for more information.
[2]See `https://www.mysql.com/` for more information.
[3]See `https://github.com/` for more information.

- OMAS[4] platform for developing our agents (Barthès, 2011).

The techniques used for implementing this MBA SoS, for instance, interfacing systems with the proxy agents, were the same we did for our ACE4SD SoS (detailed in Chapter 6).

The Extended Abstract Statements (ExAS) were *lists* in Common Lisp that in turn is language implementing the OMAS platform. For storing and retrieving the ExAS from the proxy agents, we used a slot in the agent which was a memory area represented by a *stack*. The operations in such a stack were realized using the Common Lisp *push* and *pop* commands. For retrieving the universal time we used the Common Lisp *get-universal-time*[5] function.

Next, we describe a method, named *MeTRo*, that we propose for testing the robustness of an MBA SoS (Section B.6) based on the synchronization among constituent systems.

B.6 MeTRo: Method for Testing the Robustness of MBA SoS

The method presented in this section is currently a proposition for testing the robustness of an MBA SoS. The focus is to test the MBA robustness based on the *synchronization* functionality and the *loose coupling* feature provided by our approach.

To analyze the robustness of an MBA SoS, the architect performs the steps of a method we call *MeTRo*:"Method for Testing Robustness" which is shown in Figure B.4. The MeTRo tests the robustness of an MBA SoS taking into account its loose coupling, and the coherence of constituent systems when they are reconnected to the SoS after going down. The method consists of the activities "Record," "Reset," "Replay" and "Compare."

Because the MeTRo uses the SyncMBA method, the assumptions we have made for such a method are also valid here. Thus, we recall that: (i) the synchronization is used by systems that change their internal states when cooperating with an SoS; and (ii) for each system intended to be synchronized in an SoS, there is at least one other system in such an SoS offering the same SoS functionalities of it.

The "Record" activity consists of a single step named "Record Work Order I." The goal of this step is to record all the work orders exchanged in an MBA SoS during its life cycle. Thus, as we are describing a method for testing purposes, we recommend the SoS architect to build a brand new MBA SoS in a given scenario with any number of systems (considering the assumptions made for the MeTRo). There is no constraint for

[4]See `http://www.utc.fr/~barthes/OMAS/` for full documentation.
[5]See `http://cl-cookbook.sourceforge.net/dates_and_times.html` for more information.

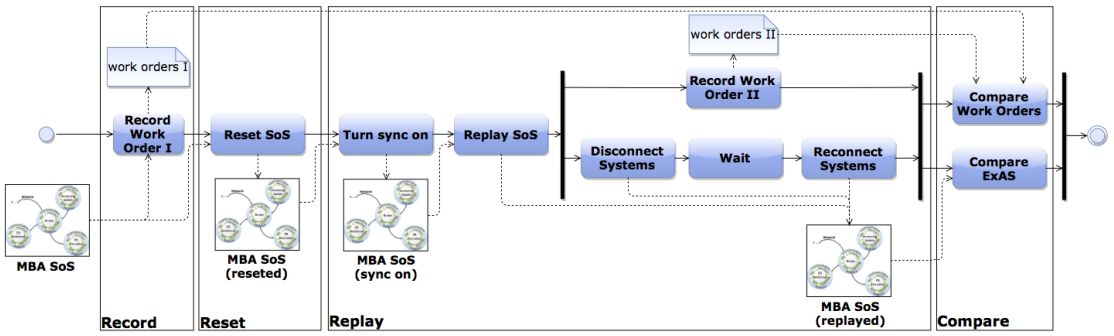


Figure B.4: The MeTRo: "Method for Testing Robustness" used for analyzing the robustness of an MBA SoS.

the length of the SoS life cycle, meaning that it is defined by the architect. The input of this step is the SoS used to record the work orders. When the SoS is in operation all the work orders exchanged between constituent systems are recorded, for instance, using SoS memories or a log agent. The output of this step is the SoS and the work orders recorded from it.

The next activity of the MeTRo named "Reset" has also a single step called "Reset SoS." This step aims to reset the SoS outputted in the previous activity. The goal is to prepare the SoS to be used in the next activities of the MeTRo. By resetting the SoS we mean clear all the data and information stored on it, putting it and its constituent systems to their initial state (as they were before being used). Among the data deleted by the architect when resetting the SoS, one may have information stored in memories and in logs. The output of this step is the SoS reseted to its initial state.

The "Replay" activity intends to replay (put in operation again) the SoS reseted in the previous activity, testing then the SoS robustness through the loose coupling and the synchronization among constituent systems. This activity consists of the steps of "Turn sync on," "Replay SoS," "Disconnect Systems," "Wait," "Reconnect Systems," and "Record Work Order II."

In the step "Turn sync on" the architect aims to turn the synchronization on in constituent systems of the SoS (note the assumptions made for the MeTRo). The goal is to let the constituents with the synchronization enabled coherent with SoS after they go down, in order to support the SoS robustness. To do that, the architect enables a synchronization goal in the proxy agents of the constituent systems she intends to allow the synchronization. The input in this step is the SoS that has been reseted in the "Reset" activity of the MeTRo, and the output is the SoS with the synchronization turned on in the proxy agents.

In the step "Replay SoS," the architect intends to put the SoS outputted in the previous step in operation by throwing chronologically the work orders recorded in the "Record" activity of the MeTRo. The purpose is to "run" the SoS using the data collected during

its life cycle, preparing it to test its robustness. The input of this step are the work orders outputted in the "Record" activity of the MeTRo.

Next, when the SoS is being replayed the step "Record Work Oder II" aims to record the work orders exchanged among constituent systems. The main goal is to collect such work orders for using them in the next activity named "Compare" of the MeTRo. To record the messages the architect could use the same technique adopted in the "Record" activity, for instance, by using the SoS memories or a log agent. The output of this step are the recorded work orders.

In parallel with the step "Record Work Oder II" occurs the step "Disconnect systems." In this step, the architect chooses constituent systems for being disconnected from the SoS to test its robustness through the synchronization. Note that the systems chosen need to be among the ones with the synchronization turned on in the "Turn sync on" step. Moreover, not all the systems with the synchronization on can be disconnected at the same time. That is, it needs to remain at least one constituent offering the same functionalities for the systems disconnected. After choosing the systems, the architect is required to disconnected them from the SoS. The goal is to simulate such systems as they were going a down. The input of this step is a list of the constituent systems chosen to go down.

Then, in the next step, named "Wait," the architect waits for some time letting the messages be replayed in the MBA SoS. The wait time is defined by the architect, and there is no restriction imposed by the MeTRo. The goal of waiting is to let the remaining constituent systems interacting in the SoS, at the same time the work orders they exchange are recorded. This kind of interaction is important to understand later in the MeTRo how an MBA SoS behavior after its systems go down. The input in this step is the time, for instance, the number of seconds, the architect will wait.

Next, in the step "Reconnect systems" the architect reconnects to the SoS the systems that were considered down. Because the synchronization was enabled in the step "Turn sync on," after the architect reconnects the systems, their proxy agents will try to synchronize automatically from other constituents that were continuously in operation (see the *SyncMBA method* in Section B.3). The goal of this step is to try to let the constituent systems that were down coherent with the current state of the SoS. The input of this step is the list of system that were down.

Note that, the fact of disconnecting and reconnecting constituent systems is related to the coupling of the SoS. We analyze such a coupling in the last activity of the MeTRo, in order to understand how an MBA SoS behavior when connecting and disconnecting constituents.

When all the work order have been thrown during the replay, we perform the last activity of the MeTRo is "Compare" which intends to compare and analyze information outputted in the previous activities, in order to understand the robustness of the MBA SoS. This activity consists of the steps of "Compare Work Orders" and "Compare ExAS."

In the "Compare Work Orders" step, the architect compares the work orders resulted from the "Record" and the "Replay" activities of the MeTRo. The goal is to understand how the loose coupling provided by our approach impacts on the the robustness of an MBA SoS. For instance, to analyze if work orders have been lost when connecting or disconnecting constituents. Our first answer would be that they have not been lost, because in an MBA SoS constituents are not connected to each other, thus when connecting or disconnecting one, the remaining continue to operating normally, i.e., they do not need to be modified, stopped or reinitialized. The input of this step is the work orders recorded in the "Record" and "Replay" activities of the MeTRo.

The last step of the "Compare" activity and also of the MeTRo is the "Compare ExAS" step, the architect aims to compare the Extended Abstract Statements (ExAS) of the constituents which were synchronized with the systems which provided the ExAS for such synchronizations. The goal is to be analyze and be sure that constituents became coherent with the SoS after the synchronizations. The input of this step is the SoS after being replayed, and the ExAS are retrieved from the proxy agents of its constituent systems.

Appendix C

CoQAS: Code Quality Analysis System

The Code Quality Analysis System (CoQAS) is shown in Figure C.1. It is composed of some elements interacting between them, say a "Calculator" agent, three evaluator agents ("A", "B" and "C") and a "Synthesizer" agent.

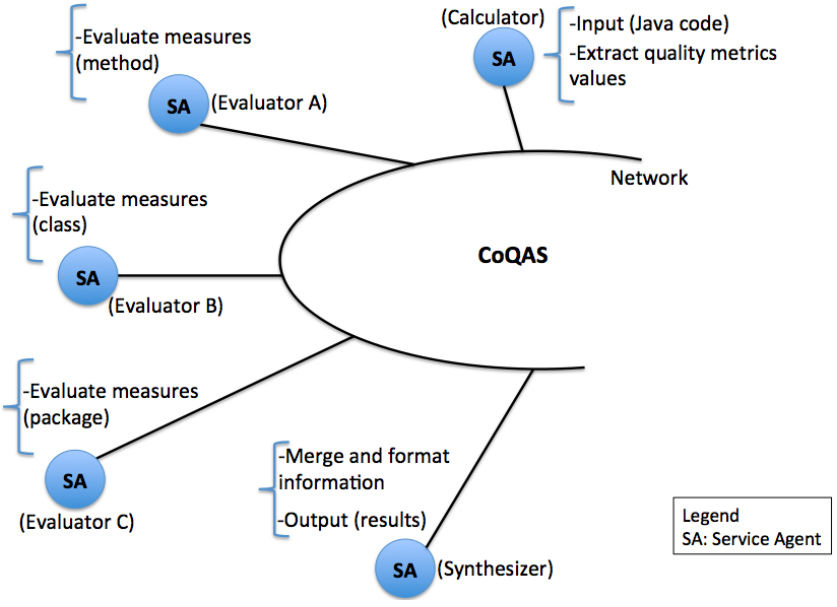


Figure C.1: The Code Quality Analysis System and its elements interacting with each other.

The "Calculator" agent receives Java source code as input, and then it extracts automatically values of Java quality metrics for such a code. To extract the metric values, the "Calculator" uses the Metrics Plugin (Metrics, 2018) which is an application for measuring Java source code through several quality metrics (see Table C.1). The evaluator agents are responsible to evaluate each one a group of the measured values. The "Evaluator A" evaluates values of method metrics, the "Evaluator B" evaluates values of class

metrics, and the "Evaluator C" evaluates values for package metrics. The evaluation is performed by using well-studied quality metric thresholds (Martin (1994); Filó et al. (2015)). Finally, the synthesizer agent intends to merge and format all the information, outputting the results in Common Lisp list format.

Table C.1: The software quality metrics extracted by the Metrics Plugin in the Code Quality Analysis System.

Code quality metrics
McCabe's Cyclomatic complexity metric - MCC (McCabe, 1976)
Weighted Methods per Class metric - WMC (Chidamber and Kemerer, 1994)
Lack of Cohesion in Method metric - LCOM* (Chidamber and Kemerer, 1994), (Sellers, 1996)
Nested Block Depth - NBD (Martin, 2003)
Depth of Inheritance Tree - DIT (Chidamber and Kemerer, 1994)
Number of Children - NOC (Chidamber and Kemerer, 1994)
Number of Overridden Methods - NORM (Sellers, 1996)
Specialization Index - SIX (Sellers, 1996)
Method Lines of Code - MLOC (Sellers, 1996)
Number of Attributes per Class - NOA (Sellers, 1996)
Number of Static Attribute - NSF (Harrison et al., 1997)
Number of Static Methods - NSM (Harrison et al., 1997)
Number of Parameter - NOP (Harrison et al., 1997)
Number of Interfaces - NOI (Harrison et al., 1997)
Number of Package - NOP (Harrison et al., 1997)
Afferent Coupling - Ca (Martin, 2003)
Efferent Coupling - Ce (Martin, 2003)
Instability metric - I, named here as RMI (Martin, 2003)
Abstractness - A, named here as RMA (Martin, 2003)
Normalize Distance from Main Sequence - D (Martin, 2003)

Appendix D

PeWeS^2: Personalized Web Search System

The PeWeS^2 provides personalized results to web search requests, performing meta-search on the top of well-known search engines. The system is shown in Figure D.1, and it is composed of some elements such as "Query" agent, "Models" agent, "Contextualize" agent, "Search" agent and "Rank" agent.

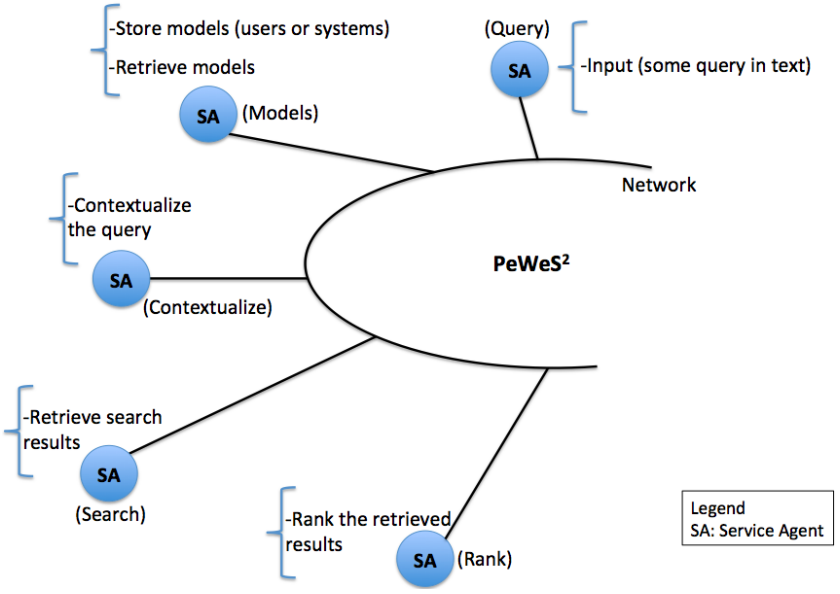


Figure D.1: The Personalized Web Search System and its elements interacting with each other.

The "Query" agent aims to receive and process a query from a user or some other system. The "Models" agent stores user or domain models, describing for instance, the interests, the favorite documents, and the search history of users or systems. The "Contextualize" agent tries to contextualize the query. The "Search" agent is responsible

to retrieve results from a well-known search engine such as Bing[1]. Finally, the "Rank" agent intends to rank the retrieved results.

When the "Query" agent receives a query, for example, a list of terms, it processes it, converting to lower case and removing extra spaces. Then, it sends a message to the "Models" agent to retrieve the user or domain model concerning the requester.

In the "Models" agent, we are currently adopting that the model comprises the interests, the favorite documents and the search history of the requester. The structure of the model is represented by the following ontology in Common Lisp:

```
;;;=============================================================================
;;;                                                                    Document
;;;=============================================================================
;; It is some document such as a webpage.

(defconcept "Document"
    (:doc
     "Some document such as a webpage.")
  (:att "url" (:entry) (:unique)) ;; unique (the "url ID" provided by the crawler)
  (:att "display-url") ;; the url used by users to retrieve the document
  (:att "name" (:entry))
  (:att "dictionary") ;; list of the most important terms describing the document
  )


;;;=============================================================================
;;;                                                                    Favorite
;;;=============================================================================
;; The favorite is a list of documents
;; Ex.: ("document-1" "document-2" ...)

(defconcept "Favorite"
    (:doc
     "The favorite documents of someone.")
  (:rel "document" (:to "Document"))
  )

;;;=============================================================================
;;;                                                                     History
;;;=============================================================================
;; The history describing the web searches made by someone
(defconcept "History"
    (:doc
     "The history concerning the searches made by someone.")
  (:att "date" (:entry))
  (:att "query") ;; ex.: "term-1" "term-2" ... "term-n"
  (:att "contextualized-query") ;; ex.: "term-1" "domain-1" "term-2" "domain-2"
                                ;;      ... "term-n" "domain-n"
  (:rel "document" (:to "Document")) ;; list of documents retrieved by the
                                     ;; contextualized query
  )

;;;=============================================================================
```

[1] See `https://www.bing.com` for more information.

```
;;;                                                                          Interest
;;;=============================================================================
;; The user's interests
;; Interest = ("interest-name" ("domain-1" "domain-2" ...))
(defconcept "Interest"
    (:doc
     "The interests of someone.")
  (:att "name" (:entry)) ;; the interest name
  (:rel "domain" (:to "Domain")) ;; the domain(s) or context(s) it belongs to.
  )
```

One may note that one of the attributes in a document is a list of terms, named dictionary, which is composed of the most important terms in the document. The most important terms are defined by the TF-IDF (Term Frequency-Inverse Document Frequency) (Salton, 1989) measure applied to the document terms, and regarding the corpus of documents retrieved with it during a search. Currently, for testing purposes, we arbitrarily take the first 256 most important terms as the dictionary attribute in a document. The goal of the dictionaries is to serve as a kind of representation of their documents, in order to facilitate the manipulation of information and save processing time during a search.

After retrieving the model, the "Query" agent sends a message with the information to the "Contextualize" agent. The contextualization of a query is defined as follows:

Definition 1. A query is said to be *contextualized* if a term from the interests or favorites of a requester's model can be correlated to it.

The goal of the contextualization is to improve the accuracy of the query, going deeper in the search to retrieve more precise and consistent results according to the requester's interests. To contextualize a query, first the "Contextualize" agent sends a message asking the "Search" agent to perform a search using the query inputted in the PeWeS^2. We call this preliminary search as *pre-search*. The goal is to retrieve a primary corpus of documents to serve as a basis to increase the accuracy of the query. Then, when receiving back the documents, the "Contextualize" agent looks for the term with the highest TF (Term Frequency) in the interests and favorites of the requester's model. The term with the highest TF will be used to contextualize, being appended to the original query.

In the next step, the "Contextualize" agent sends a message to the "Search" agent to perform a search, now using the query that has been contextualized. The documents retrieved by the agent are then sent to be ranked by the "Rank" agent. In the ranking process, first the TF-IDF is computed for the terms of the interests and favorites from the requester's model, regarding each of the documents retrieved. In the sequence, we sum the TF-IDFs of the model terms in each document. Finally, a ranking is performed in descending order, putting the documents with the highest TF-IDF on the top.

In the last step, the "Query" agent receives the documents ranked and it sends them to the be viewed by the requester. Moreover, the "Models" agent receive the documents, in order to update the model. The "Models" agent extracts the documents' dictionaries and update the model history. It is up to the requester to update his interests and favorite documents.

Appendix E

EclipA^2: Eclipse Augmented by Agent

The EclipA^2 is a system used by developers for coding, for instance, through Java. Figure E.1 shows the EclipA^2 and its elements. The system consists of the Eclipse IDE[1] augmented with a plugin, a local version control system, a Personal Assistant (PA), and a staff agent which is a Service Agent (SA).

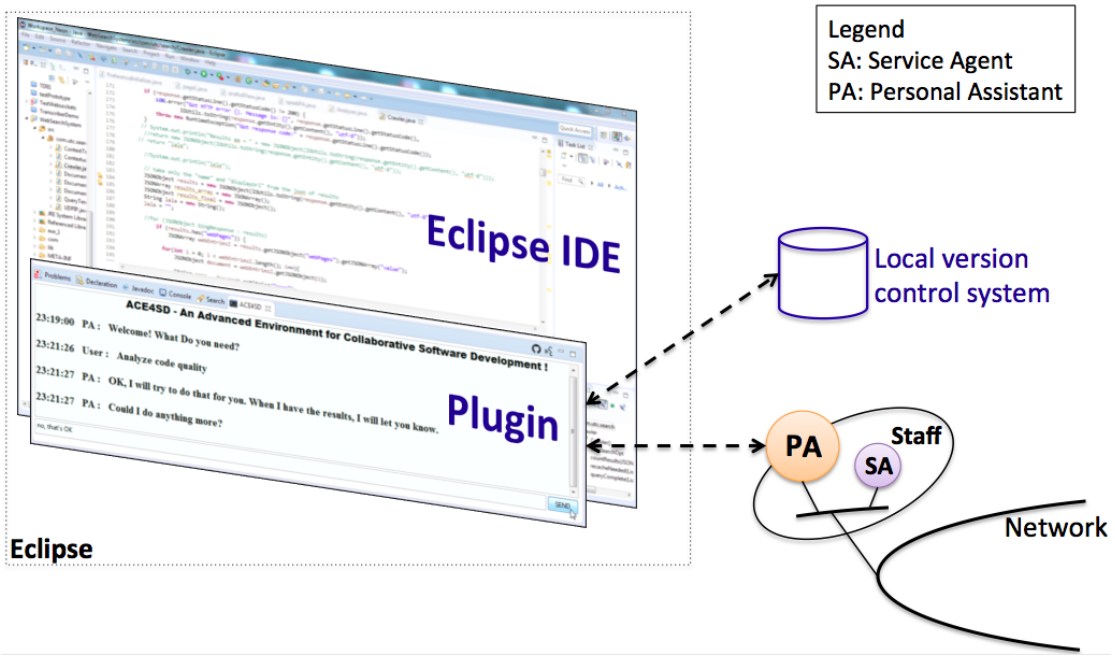


Figure E.1: The EclipA^2 system and its components.

The Eclipse IDE is used for coding and the plugin is a view that augments it, allowing a developer to interact with her PA in natural language (NL). The PA follows the idea of a digital butler (see Chapter 3) and in the context of Systems of Systems (SoS), it

[1]See `https://www.eclipse.org/` for more information.

can assist its user, for instance, to request SoS functionalities. The staff agent supports the PA with more specialized services, for example, it may store User Models (UM) to help the PA to provide customized support to its user. The local version control system which could be Git[2] aims to store locally code versions produced by the developer.

Figure E.2 shows in detail the EclipA[2] interface used by a given software developer. The interface is provided by the Eclipse IDE augmented with a plugin. The interface consists of dialogue panels showing the history of the utterances exchanged between a developer and her PA. The user can interact with her PA in NL by typing the utterance in a box, or by speaking it when clicking on a button. The interface also provides a button that directs the developer to her space in the GitHub[3] (online code versioning system).

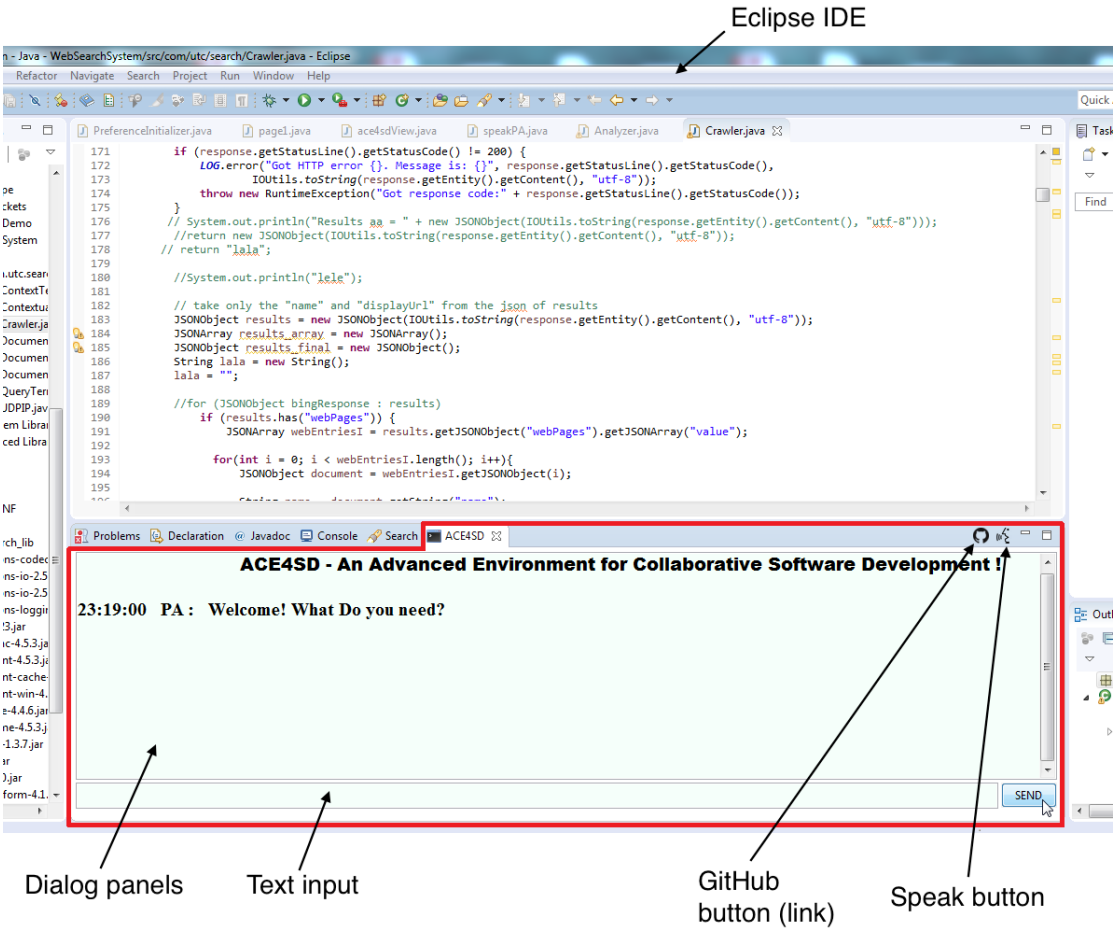


Figure E.2: The EclipA[2] interface which is the Eclipse IDE augmented with a plugin.

[2]See `https://git-scm.com` for more information.
[3]See `https://github.com` for more information.

Appendix F

MBA Browser

The MBA Browser is a system for sending and retrieving information from a given System of Systems (SoS). The interaction between a user and the system is made through Natural Language (NL) written or spoken. Figure F.1 shows the MBA Browser system and its components. The system consists of a browser, a local server handled by a Service Agent (SA), a Personal Assistant (PA), and a staff agent which is an SA. The browser component has an interface acting as a front end with the user, and a vocal interface in the back end for providing speech-to-text and text-to-speech. The local server is responsible for the exchanging of messages between the browser and the other elements in the system. The PA is a digital butler for assisting the user, and the staff agent aims to support the PA with more specialized services, as we stated such an idea in the Chapter 3.

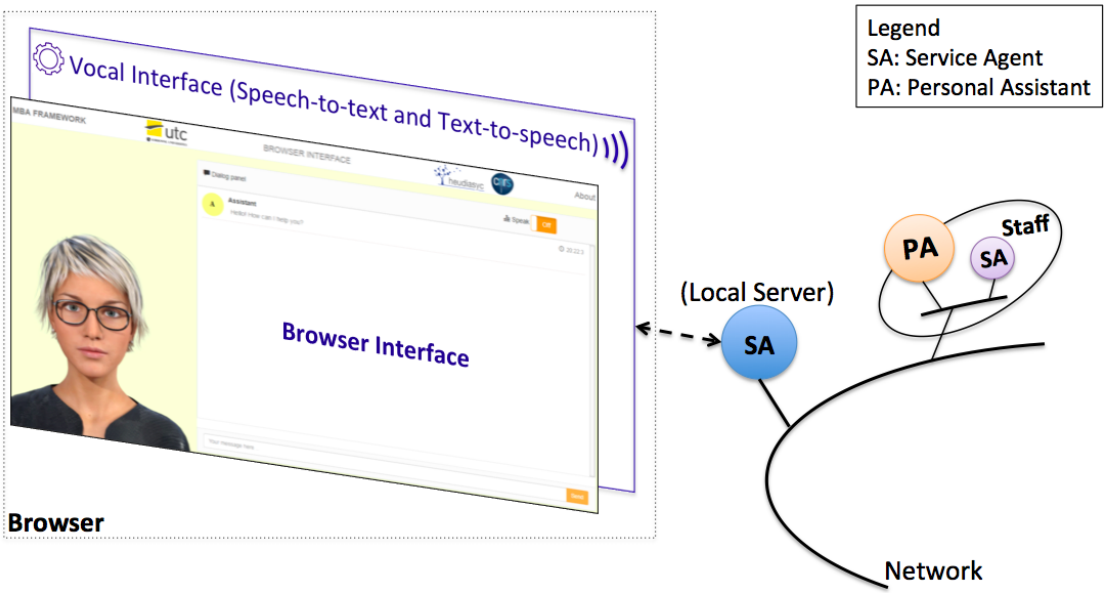


Figure F.1: The MBA Browser system and its components.

Figure F.2 shows in detail the browser front end interface for the user interaction. The interface has a dynamic avatar[1] that moves the head and the face when reproducing text-to-speech. Moreover, dialogues panels show in NL the history of the interactions between user and PA. The interaction can be realized vocally by switching a button (in the top-right), or written by typing the text in a box.

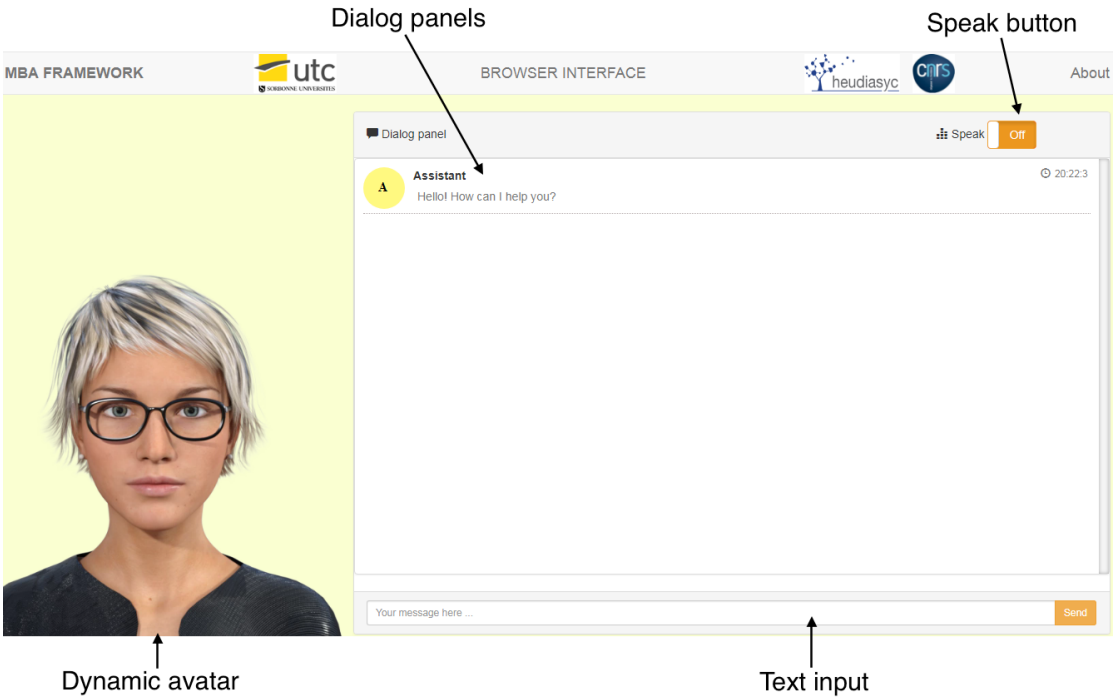


Figure F.2: The front ent interface of the MBA Browser.

[1] See `https://www.botlibre.com` for more information.

Bibliography

Abel, M.-H. (2015). Knowledge map-based web platform to facilitate organizational learning return of experiences. *Computers in Human Behavior*, 51:960–966.

Ackoff, R. L. (1971). Towards a system of systems concepts. *Management science*, 17(11):661–671.

Agarwal, S., Pape, L. E., Dagli, C. H., Ergin, N. K., Enke, D., Gosavi, A., Qin, R., Konur, D., Wang, R., and Gottapu, R. D. (2015). Flexible and intelligent learning architectures for sos (fila-sos): Architectural evolution in systems-of-systems. *Procedia Computer Science*, 44:76–85.

Arar, Ö. F. and Ayan, K. (2016). Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies. *Expert Systems with Applications*, 61:106–121.

Audi, R. (1999). The cambridge dictionary of philosophy. pages 898–899.

Bai, L. (2013). System of systems engineering and geographical simulation: Towards a smart tourism industry information system. In *Advanced Communication Technology (ICACT), 2013 15th International Conference on*, pages 1015–1018. IEEE.

Baldwin, W. C. and Sauser, B. (2009). Modeling the characteristics of system of systems. In *System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on*, pages 1–6. IEEE.

Barthès, J.-P. A. (2011). Omas a flexible multi-agent environment for cscwd. *Future Generation Computer Systems*, 27(1):78–87.

Barthès, J.-P. A. (2013). Improving human-agent communication using linguistic and ontological cues. *International Journal of Electronic Business*, 10(3):207–231.

Bertalanffy, L. v. (1969). General system theory: Foundations, development, applications.

Bettenburg, N. and Hassan, A. E. (2013). Studying the impact of social interactions on software quality. *Empirical Software Engineering*, 18(2):375–431.

Biran, Y., Collins, G., Azam, S., and Dubow, J. (2017). Federated cloud computing as system of systems. In *Computing, Networking and Communications (ICNC), 2017 International Conference on*, pages 711–718. IEEE.

Bonsignour, O. and Jones, C. (2011). *The Economics of Software Quality*. Addison Wesley.

Booher, H. R. (2003). *Handbook of human systems integration*, volume 23. John Wiley & Sons.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons.

Borg, M., Olsson, T., and Svensson, J. (2017). Piggybacking on an autonomous hauler: Business models enabling a system-of-systems approach to mapping an underground mine. In *Requirements Engineering Conference (RE), 2017 IEEE 25th International*, pages 372–381. IEEE.

Boulding, K. E. (1956). General systems theory—the skeleton of science. *Management science*, 2(3):197–208.

Bray, M., Brune, K., Fisher, D. A., Foreman, J., and Gerken, M. (1997). C4 software technology reference guide - a prototype. Technical report, Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh, PA, USA.

Brooks, R. A. (1990). Elephants don't play chess. *Robotics and autonomous systems*, 6(1-2):3–15.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing.

Callebert, L., Lourdeaux, D., and Barthès, J.-P. (2016). Trust-based decision-making system for action selection by autonomous agents. In *Computer Supported Cooperative Work in Design (CSCWD), 2016 IEEE 20th International Conference on*, pages 4–9. IEEE.

Cambridge University (2013). Cambridge University study. http://insight.jbs.cam.ac.uk/2013/financial-content-cambridge-university-study-states-software-bugs-cost-economy-312-billion-per-year/. Online; accessed 08 February 2018.

Carlock, P. G. and Fenton, R. E. (2001). System of systems (sos) enterprise systems engineering for information-intensive organizations. *Systems engineering*, 4(4):242–261.

Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493.

Coleman, D. (1992). Assessing maintainability. In *Proceedings of the Software Engineering Productivity Conference*, pages 525–532.

Corsello, M. A. (2008). System-of-systems architectural considerations for complex environments and evolving requirements. *IEEE Systems Journal*, 2(3):312–320.

Curry, E. (2012). System of systems information interoperability using a linked dataspace. In *System of Systems Engineering (SoSE), 2012 7th International Conference on*, pages 101–106. IEEE.

Dauby, J. P. and Upholzer, S. (2011). Exploring behavioral dynamics in systems of systems. *Procedia Computer Science*, 6:34–39.

Davendralingam, N. and DeLaurentis, D. (2013). A robust optimization framework to architecting system of systems. *Procedia Computer Science*, 16:255–264.

DeLaurentis, D. (2007). Role of humans in complexity of a system-of-systems. In *International Conference on Digital Human Modeling*, pages 363–371. Springer.

DeLaurentis, D. and Callaway, R. K. C. (2004). A system-of-systems perspective for public policy decisions. *Review of Policy Research*, 21(6):829–837.

DeLaurentis, D. A. (2005). A taxonomy-based perspective for systems of systems design methods. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 1, pages 86–91. IEEE.

DiMario, M. J. (2006). System of systems interoperability types and characteristics in joint command and control. In *System of Systems Engineering, 2006 IEEE/SMC International Conference on*, pages 6–pp. IEEE.

DoDAF (2004). Department of Defense (DoD) Architecture Frameworking Group. Version 1, volume II: Product Descriptions, February 2004.

Eckstein, J. and Schultz, B. R. (2018). Introduction to data management and database design. *Introductory Relational Database Design for Business, with Microsoft Access*, pages 43–51.

Enembreck, F. (2003). *Contribution à la conception d'agents assistants personnels adaptatifs*. PhD thesis. Thèse de doctorat dirigée par Barthès, Jean-Paul Technologies de l'information et des systèmes Compiègne 2003.

Enos, J. R., Mansouri, M., and Nilchiani, R. (2017). Applying social network analysis to systems of systems: The case of the 2016 puerto rico power outage. In *System of Systems Engineering Conference (SoSE), 2017 12th*, pages 1–6. IEEE.

Ferber, J. (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading.

Ferrario, E., Pedroni, N., and Zio, E. (2016). Evaluation of the robustness of critical infrastructures by hierarchical graph representation, clustering and monte carlo simulation. *Reliability Engineering & System Safety*, 155:78–96.

Filó, T., Bigonha, M., and Ferreira, K. (2015). A catalogue of thresholds for object-oriented software metrics. In *Advances and Trends in Software Engineering, 2015 The First International Conference on*, pages 48–55. IARIA.

Ford, S., Rauschecker, U., and Athanassopoulou, N. (2012). System-of-system approaches and challenges for multi-site manufacturing. In *System of Systems Engineering (SoSE), 2012 7th International Conference on*, pages 1–6. IEEE.

Foster, S., Miyazawa, A., Woodcock, J., Cavalcanti, A., Fitzgerald, J., and Larsen, P. G. (2014). An approach for managing semantic heterogeneity in systems of systems engineering. In *System of Systems Engineering (SOSE), 2014 9th International Conference on*, pages 113–118. IEEE.

Freitas, A., Oliveira, J. G., O'riain, S., Da Silva, J. C., and Curry, E. (2013). Querying linked data graphs using semantic relatedness: A vocabulary independent approach. *Data & Knowledge Engineering*, 88:126–141.

Friedrich, T., Hilbes, C., and Nordt, A. (2017). Systems of systems engineering for particle accelerator based research facilities: A case study on engineering machine protection. In *Systems Conference (SysCon), 2017 Annual IEEE International*, pages 1–8. IEEE.

Fuckner, M., Barthès, J.-P., and Scalabrin, E. E. (2014). Using a personal assistant for exploiting service interfaces. In *Computer Supported Cooperative Work in Design (CSCWD), Proceedings of the 2014 IEEE 18th International Conference on*, pages 89–94. IEEE.

Ge, B., Hipel, K. W., Yang, K., and Chen, Y. (2014). A novel executable modeling approach for system-of-systems architecture. *IEEE Systems Journal*, 8(1):4–13.

Gomez, M., Kim, Y., Matson, E., Tolstykh, M., and Munizzi, M. (2015). Multi-agent system of systems to monitor wildfires. In *System of Systems Engineering Conference (SoSE), 2015 10th*, pages 262–267. IEEE.

Gonçalves, M. B., Oquendo, F., and Nakagawa, E. Y. (2015). A meta-process to construct software architectures for system of systems. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1411–1416. ACM.

Guariniello, C. and DeLaurentis, D. (2014). Communications, information, and cyber security in systems-of-systems: Assessing the impact of attacks through interdependency analysis. *Procedia Computer Science*, 28:720–727.

Halstead, M. H. (1977). *Elements of software science*, volume 7. Elsevier New York. ISBN: 0-444-00205-7.

Han, S. Y., Marais, K., and DeLaurentis, D. (2012). Evaluating system of systems resilience using interdependency analysis. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 1251–1256. IEEE.

Harrison, R., Counsell, S., and Nithi, R. (1997). An overview of object-oriented design metrics. In *Software Technology and Engineering Practice, 1997. Proceedings., Eighth IEEE International Workshop on [incorporating Computer Aided Software Engineering]*, pages 230–235. IEEE.

Hershey, P., Wang, M.-C., and Toppin, D. (2013). System of systems for autonomous mission decisions. In *System of Systems Engineering (SoSE), 2013 8th International Conference on*, pages 129–134. IEEE.

Hu, J., Huang, L., Chang, X., and Cao, B. (2014). A model driven service engineering approach to system of systems. In *Systems Conference (SysCon), 2014 8th Annual IEEE*, pages 136–145. IEEE.

Ingram, C., Payne, R., and Fitzgerald, J. (2015). Architectural modelling patterns for systems of systems. In *INCOSE International Symposium*, volume 25, pages 1177–1192. Wiley Online Library.

Ingram, C., Payne, R., Perry, S., Holt, J., Hansen, F. O., and Couto, L. D. (2014). Modelling patterns for systems of systems architectures. In *Systems Conference (SysCon), 2014 8th Annual IEEE*, pages 146–153. IEEE.

ISO/IEC/IEEE 15288 (2015). Systems and Software Engineering-System Life cycle Processes. International Organization for Standardization. Geneva, Switzerland.

Jackson, M. C. and Keys, P. (1984). Towards a system of systems methodologies. *Journal of the operational research society*, 35(6):473–486.

Jacob, F. (1974). The logic of living systems: a history of heredity. Allen Lane.

Jamshidi, M. (2017). *Systems of systems engineering: principles and applications*. CRC press.

Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38.

Johnson, J. and Gheorghe, A. V. (2013). Antifragility analysis and measurement framework for systems of systems. *International Journal of Disaster Risk Science*, 4(4):159–168.

Jokinen, J., Ambat, M. B., and Lastra, J. L. M. (2016). Condition monitoring for distributed systems with reconfigurable user interfaces and data permissions. In *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*, pages 5705–5710. IEEE.

Jones, A., Kendira, A., Moulin, C., Barthes, J.-P. A., Lenne, D., and Gidel, T. (2012). Vocal interaction in collocated cooperative design. In *Cognitive Informatics & Cognitive Computing (ICCI* CC), 2012 IEEE 11th International Conference on*, pages 246–252. IEEE.

Keating, C., Rogers, R., Unal, R., Dryer, D., Sousa-Poza, A., Safford, R., Peterson, W., and Rabadi, G. (2003). System of systems engineering. *Engineering Management Journal*, 15(3):36–45.

Kilian, J. C. and Schuck, T. M. (2016). Architecture and system-of-systems design for integrated missile defense. In *System of Systems Engineering Conference (SoSE), 2016 11th*, pages 1–6. IEEE.

Kumar, P., Merzouki, R., and Bouamama, B. O. (2017). Multilevel modeling of system of systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

Lana, C. A., Souza, N. M., Delamaro, M. E., Nakagawa, E. Y., Oquendo, F., and Maldonado, J. C. (2016). Systems-of-systems development: Initiatives, trends, and challenges. In *Computing Conference (CLEI), 2016 XLII Latin American*, pages 1–12. IEEE.

Lander, S. E., Corkill, D. D., and Staley, S. M. (1996). Designing integrated engineering environments: blackboard-based integration of design and analysis tools. *Concurrent Engineering*, 4(1):59–71.

Lewis, G., Morris, E., Simanta, S., and Smith, D. (2011). Service orientation and systems of systems. *IEEE software*, 28(1):58–63.

Lhommet, M., Lourdeaux, D., and Barthès, J.-P. A. (2012). Foule sentimentale: influence des caractéristiques individuelles sur la contagion émotionnelle. *Revue d'Intelligence Artificielle*, 26(3):281–308.

Madni, A. (2011). Integrating humans with and within software and systems: Challenges and opportunities (invited paper). *CrossTalk J Defense Software Eng May/June*.

Madni, A. M. (2010). Integrating humans with software and systems: Technical challenges and a research agenda. *Systems Engineering*, 13(3):232–245.

Madni, A. M. and Sievers, M. (2014). System of systems integration: key considerations and challenges. *Systems Engineering*, 17(3):330–347.

Maier, M. W. (1996). Architecting principles for systems-of-systems. *INCOSE International Symposium*, 6(1):565–573.

Maier, M. W. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284.

Martin, R. (1994). Oo design quality metrics. *An analysis of dependencies*, 12:151–170.

Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*. Prentice Hall PTR.

Mazzetti, P., Puglisi, G., D'Auria, L., Roncella, R., Reitano, D., Merenda, R., and Nativi, S. (2017). The med-suv virtual research environment for enabling the geo geohazard supersites in italy. *Earth Science Informatics*, pages 1–13.

McCabe, T. J. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*, (4):308–320.

McGuire, J. G., Kuokka, D. R., Weber, J. C., Tenenbaum, J. M., Gruber, T. R., and Olsen, G. R. (1993). Shade: Technology for knowledge-based collaborative engineering. *Concurrent Engineering*, 1(3):137–146.

Metrics (2018). Eclipse Metrics Plugin. `http://metrics2.sourceforge.net`. Online; accessed 31 March 2018.

MODAF (2005). MODAF Partners, MOD Architecture Framework. Technical Handbook, Version 1.0, August 2005.

Moschoglou, G., Eveleigh, T., Holzer, T., and Sarkani, S. (2012). A semantic mediation framework for architecting federated ubiquitous systems. In *System of Systems Engineering (SoSE), 2012 7th International Conference on*, pages 485–490. IEEE.

Myers, G. J. (1977). An extension to the cyclomatic measure of program complexity. *ACM Sigplan Notices*, 12(10):61–64.

Nakagawa, E. Y., Gonçalves, M., Guessi, M., Oliveira, L. B., and Oquendo, F. (2013). The state of the art and future perspectives in systems of systems software architectures. In *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems*, pages 13–20. ACM.

Nativi, S., Craglia, M., and Pearlman, J. (2013). Earth science infrastructures interoperability: the brokering approach. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 6(3):1118–1129.

Nativi, S., Mazzetti, P., Santoro, M., Papeschi, F., Craglia, M., and Ochiai, O. (2015). Big data challenges in building the global earth observation system of systems. *Environmental Modelling & Software*, 68:1–26.

Negroponte, N. (1996). *Being digital*. Vintage.

Nielsen, C. B., Larsen, P. G., Fitzgerald, J., Woodcock, J., and Peleska, J. (2015). Systems of systems engineering: basic concepts, model-based techniques, and research directions. *ACM Computing Surveys (CSUR)*, 48(2):18.

Nomaguchi, Y. and Fujita, K. (2017). A framework of system of systems design with scenario, multi-agent simulation and robustness assessment. *Procedia CIRP*, 60:133–138.

Olivier, J. P., Balestrini-Robinson, S., and Briceno, S. (2014). Approach to capability-based system-of-systems framework in support of naval ship design. In *Systems Conference (SysCon), 2014 8th Annual IEEE*, pages 388–395. IEEE.

Oman, P. and Hagemeister, J. (1994). Construction and testing of polynomials predicting software maintainability. *Journal of Systems and Software*, 24(3):251–266.

Oquendo, F. (2016). Formally describing the software architecture of systems-of-systems with sosadl. In *System of Systems Engineering Conference (SoSE), 2016 11th*, pages 1–6. IEEE.

Pape, L. and Dagli, C. (2013). Assessing robustness in systems of systems meta-architectures. *Procedia Computer Science*, 20:262–269.

Paraiso, E. C. and Barthès, J.-P. A. (2006). An intelligent speech interface for personal assistants in r&d projects. *Expert Systems with Applications*, 31(4):673–683.

Pérez, J., Díaz, J., Garbajosa, J., Yagüe, A., Gonzalez, E., and Lopez-Perea, M. (2015). Towards a reference architecture for large-scale smart grids system of systems. In *Proceedings of the Third International Workshop on Software Engineering for Systems-of-Systems*, pages 5–11. IEEE Press.

Radatz, J., Geraci, A., and Katki, F. (1990). Ieee standard glossary of software engineering terminology. *IEEE Std*, 610121990(121990):47.

Ramos, M. (2000). *Structuration et évolution conceptuelles d'un agent assistant personnel dans les domaines techniques*. PhD thesis. Thèse de doctorat dirigée par Barthès, Jean-Paul Contrôle des systèmes Compiègne 2000.

Salton, G. (1989). Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley*.

Sanduka, I. and Obermaisser, R. (2014). Model-based development of systems-of-systems with real-time requirements. In *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*, pages 188–194. IEEE.

Saunders, T., Croom, C., Austin, W., Brock, J., Crawford, N., Endsley, M., et al. (2005). System-of-systems engineering for air force capability development. *US Air Force Scientific Advisory Board*.

Schlager, K. J. (1956). Systems engineering-key to modern development. IRE Trans. Eng. manag. vol. EM-3, no. 3. pp. 64-66.

Selberg, S. A. and Austin, M. A. (2008). 10.1. 1 toward an evolutionary system of systems architecture. In *INCOSE International Symposium*, volume 18, pages 1065–1078. Wiley Online Library.

Sellers, B. H. (1996). Ojbect-oriented metrics. measures of complexity.

Shen, W., Norrie, D. H., and Barthès, J.-P. (2003). *Multi-agent systems for concurrent intelligent design and manufacturing*. CRC press.

Siemieniuch, C. and Sinclair, M. (2012). 11.4. 1 systems of systems (sos) engineering: a view from the inside looking out. In *INCOSE International Symposium*, volume 22, pages 1653–1670. Wiley Online Library.

Siemieniuch, C. E. and Sinclair, M. A. (2014). Extending systems ergonomics thinking to accommodate the socio-technical issues of systems of systems. *Applied ergonomics*, 45(1):85–98.

Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on computers*, (12):1104–1113.

Stary, C. and Wachholder, D. (2016). System-of-systems support—a bigraph approach to interoperability and emergent behavior. *Data & Knowledge Engineering*, 105:155–172.

Stein, J.-P. and Ohler, P. (2017). Venturing into the uncanny valley of mind—the influence of mind attribution on the acceptance of human-like characters in a virtual reality setting. *Cognition*, 160:43–50.

Sycara, K. P. (1998). Multiagent systems. *AI magazine*, 19(2):79.

Tran, H. T., Domerçant, J. C., and Mavris, D. N. (2016). A network-based cost comparison of resilient and robust system-of-systems. *Procedia Computer Science*, 95:126–133.

Tumini, M., Nagel, O., Molina, M. P., and Althaus, R. (2017). Microbiological assay with bacillus licheniformis for the easy detection of quinolones in milk. *International Dairy Journal*, 64:9–13.

Uhrmacher, A. M. and Weyns, D. (2009). *Multi-Agent systems: Simulation and applications*. CRC press.

Vandenbergh, É., Heurley, L., Hainselin, M., Quaglino, V., and Mouras, H. (2017a). Filling a pillbox from a medication prescription displayed on a tablet: Impact of presentation formats in young and older adults. In *Journal of Gerontology and Geriatric Research*, volume 6:3 (Suppl).

Vandenbergh, É., Wanderley, G. M. P., Abel, M.-H., Barthès, J.-P., Hainselin, M., Moulin, C., Mouras, H., and Heurley, L. (2017b). Consignela-appli-r.1.0. a research tool for studying in real time electronic medication prescription comprehension in older adults and patients with parkinson's disease. In *Journal of Health and Medical Informatics*, volume 8:3 (Suppl).

Varga, P., Blomstedt, F., Ferreira, L. L., Eliasson, J., Johansson, M., Delsing, J., and de Soria, I. M. (2017). Making system of systems interoperable–the core components of the arrowhead framework. *Journal of Network and Computer Applications*, 81:85–95.

Vierhauser, M., Rabiser, R., Grünbacher, P., Seyerlehner, K., Wallner, S., and Zeisel, H. (2016). Reminds: a flexible runtime monitoring framework for systems of systems. *Journal of Systems and Software*, 112:123–136.

Von Bertalanffy, L. (1967). *Robots, men, and minds: Psychology in the modern world*. George Braziller.

Walden, D. D., Roedler, G. J., Forsberg, K., Hamelin, R. D., and Shortell, T. M. (2015). *Systems engineering handbook: A guide for system life cycle processes and activities*. John Wiley & Sons.

Wanderley, G. M. P., Abel, M.-H., Barthès, J.-P., and Paraiso, E. C. (2017a). A core architecture for developing systems of systems. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 141–146.

Wanderley, G. M. P., Abel, M.-H., Paraiso, E. C., and Barthès, J.-P. A. (2018). MBA: A system of systems architecture model for supporting collaborative work. *Computers in Industry*, 100:31–42.

Wanderley, G. M. P., Vandenbergh, É., Abel, M.-H., Barthès, J.-P. A., Hainselin, M., Mouras, H., Lenglet, A., Tir, M., and Heurley, L. (2017b). CONSIGNELA: A multi-disciplinary patient-centered project to improve drug prescription comprehension and execution in elderly people and parkinsonian patients. *Telematics and Informatics*.

Wasson, C. S. (2015). *System engineering analysis, design, and development: Concepts, principles, and practices*. John Wiley & Sons.

Wickramasinghe, N. and Schaffer, J. L. (2006). Creating knowledge-driven healthcare processes with the intelligence continuum. *International Journal of Electronic Healthcare*, 2(2):164–174.

Wirth, N. (1977). What can we do about the unnecessary diversity of notation for syntactic definitions? *Communications of the ACM*, 20(11):822–823.

Wong, R. K., Chi, C. H., Yu, Z., and Zhao, Y. (2014). A system of systems service design for social media analytics. In *Services Computing (SCC), 2014 IEEE International Conference on*, pages 789–796. IEEE.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152.

Zeigler, B. P., Mittal, S., and Hu, X. (2008). Towards a formal standard for interoperability in m&s/system of systems integration. In *GMU-AFCEA Symposium on Critical Issues in C4I*.

Zhou, B., Dvoryanchikova, A., Lobov, A., and Lastra, J. L. M. (2011). Modeling system of systems: A generic method based on system characteristics and interface. In *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pages 361–368. IEEE.