AVERTISSEMENT

LIENS

Code la Propriété Intellectuelle – Articles L. 122-4 et L. 335-1 à L. 335-10
Loi n° 92-597 du 1er juillet 1992, publiée au *Journal Officiel* du 2 juillet 1992

http://www.cfcopies.com/V2/leg/leg-droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse Capitole

École doctorale : **Droit et Science Politique**

Présentée et soutenue par

BOUZIAT Teddy

le 20/11/2017

A Cooperative Architecting Procedure for Systems of Systems Based on Self-Adaptive Multi-Agent Systems

---

Discipline : Informatique

Spécialité : Domaine STIC : Intelligence Artificielle

Unité de recherche : Institut de Recherche en Informatique de Toulouse (UMR 5505)

Directeur de thèse : Pierre GLIZE, Chargé de Recherche, Université de Toulouse

JURY

Rapporteurs	M. Javier BAJO, Associate Professor, Universidad Politecnica de Madrid M. Michel OCCELO, Professeur, Université Grenoble Alpes
Suffragants	M. Vincent CHEVRIER, Professeur, Université de Lorraine M. Frédéric AMBLARD, Professeur, Université de Toulouse
Co-Encadrants	Mme Valérie CAMPS, Maître de conférences, Université de Toulouse Mme Stéphanie COMBETTES, Maître de conférences, Université de Toulouse
Directeur(s) de thèse	M. Pierre GLIZE, Chargé de Recherche, Université de Toulouse

Teddy Bouziat

A Cooperative Architecting Procedure for Systems of Systems Based on Self-Adaptive Multi-Agent Systems

Directeur : Pierre Glize, Ingénieur de Recherche HDR, CNRS
Co-Encadrante : Valérie Camps, Maître de Conférences, UPS
Co-Encadrante : Stéphanie Combettes, Maître de Conférences, UPS

Abstract

Since the World War II, researchers have tended to develop methodologies and tools to build and control the development of more and more complex systems and projects. This inter-disciplinary research area has been called Systems Engineering (SE) and continues to be developed nowadays. In 1990, the fall of USSR led the US Department of Defense (DoD) to re-think its defense doctrine and to switch from a one opponent confrontation to a globalization of conflicts with a huge variety of scenarios. Its idea was to re-use and join its defense systems by producing a huge, decentralized and adaptive defense system that is composed of existing and independents (complex) systems. This is the apparition of the System of Systems (SoS) concept. After 2000's, this concept spreads in civil domains such as crisis management or logistic systems. More precisely, a SoS is a complex system characterized by the particular nature of its components: these latter, which are systems, tend to be managerially and operationally independent as well as geographically distributed. This specific characterization led to re-think research areas of classic SE such as definition, taxonomy, modeling, architecting and so on. SoS architecting focuses on the way independent components of a SoS can be dynamically structured and can change autonomously their interactions in an efficient manner to fulfill the goal of the SoS and to cope with the high dynamics of the environment. This PhD thesis mainly focuses on two SoS research areas: 1) SoS modeling and 2) SoS architecting. To achieve the first point, we propose a new model called SApHESIA (SoS Architecting HEuriStIc based on Agent). We have used set theory and ABM (Agent-Based Model) paradigm to define this model that takes into account the characteristics of SoS. Secondly, we propose a new SoS architecting procedure based on the Adaptive Multi-Agent System (AMAS) approach that advocates full cooperation between all the components of the SoS through the concept of criticality. This criticality is a metric that represents the distance between the current state of a component and its goals. In this procedure, the SoS architecture evolves over time to self-adapt to the dynamics of the environment in which it is plunged, while taking into account the respective local goals of its components. Finally we instantiate this model and this procedure through 4 examples from different domains (military, logistics and exploratory missions) and validate the feasibility, the efficiency, the effectiveness and the robustness of the SoS architecting procedure we have developed and proposed.

Teddy Bouziat

UNE PROCÉDURE D'ARCHITECTURE COOPÉRATIVE POUR LES SYSTÈMES DE SYSTÈMES BASÉE SUR UNE APPROCHE PAR SYSTÈME MULTI-AGENT AUTO-ADAPTATIF

Directeur : Pierre Glize, Ingénieur de Recherche HDR, CNRS
Co-Encadrante : Valérie Camps, Maître de Conférences, UPS
Co-Encadrante : Stéphanie Combettes, Maître de Conférences, UPS

Résumé

Depuis la seconde guerre mondiale, l'ingénierie des systèmes a permis le développement de méthodologies pour contrôler le développement de systèmes et de projets de plus en plus complexes. En 1990, la chute de l'URSS a provoqué un changement de doctrine militaire aux Etats-Unis en passant d'une confrontation bipolaire à une mondialisation des conflits comportant une grande variété de menaces. Sa nouvelle doctrine était de faire collaborer ses systèmes de défense existants pour produire un système de défense de haut niveau, décentralisé, adaptable et composé de systèmes indépendants. C'est l'apparition du concept de Système de Systèmes (SdS).

Cette thèse de doctorat propose un nouveau modèle de SdS appelé SApHESIA (SoS Architecting HEuriStIc based on Agent), ainsi qu'une nouvelle méthodologie d'architecture. Cette nouvelle méthodologie est basée sur une coopération complète entre tous les composants du SdS, lui permettant d'évoluer de lui-même afin de faire face à des événements inattendus de son environnement tels que des menaces. Enfin, ce travail est testé à travers 4 exemples issus de différents domaines (militaire, logistique et exploratoire).

Je dédie cette thèse à mon père
qui m'a donné goût à la science

Remerciements

Dans le meilleur des cas, si tout va bien, je serais docteur quand vous lirez ces lignes. Arriver au bout de cette thèse fut plus difficile que je ne le pensais. Un doctorat est un travail de longue haleine et contrairement à mes idées reçues, le résultat d'un travail en équipe. Je commencerai donc par remercier les principaux contributeurs scientifiques de cette thèse, mon directeur de thèse, Pierre, pour son expertise scientifique, son écoute ainsi que sa capacité à trouver des solutions inattendues aux moments opportuns. Je remercie mes deux co-directrices Stéphanie et Valérie pour les mêmes raisons, ainsi que pour les longues heures de corrections d'articles et du manuscrit. La qualité de ces derniers est due en grande partie à votre souci du détail.

Je remercie les rapporteurs de cette thèse Javier Bajo et Michel Occello pour avoir accepté de relire et de commenter ce manuscrit. Je remercie Vincent Chevrier pour avoir accepté de faire partie du jury en tant qu'examinateur. Je me permets de remercier tout particulièrement Frédéric Amblard, qui a accepté à la dernière minute de faire partie de ce jury. Je le remercie aussi pour sa sympathie qui a rendu les séances de corrections du C2I agréables.

Je remercie tous les membres de l'équipe SMAC temporaires et permanents, Françoise, Frédéric, Jean-Paul, Carole, Guy, Benoit, Jean-Pierre (merci pour les jeux post-apo!), Marie-Pierre, Chihab que je remercie pour ses conseils pédagogiques et de rédaction, Elsy, Christine Régis, Rubén, Christine Maurel et Sylvie. Votre présence et votre gentillesse ont été plus que bénéfique sur mon moral durant cette thèse. Je remercie les membres de l'équipe pédagogique du département informatique d'UT1, Nathalie, Sylvie, Julien, Umberto, Laurent, David, Jean-Marc, Chantal, Maryse, Éric, Yves, Alain et Stéphane.

Je remercie particulièrement les doctorants de SMAC, les nouveaux comme les anciens. Au cours de ces années la plupart sont devenus des amis et m'ont permis de passer des soirées mémorables, d'avoir des discussions passionnantes sur l'existence de la conscience chez les parpaings, sur le libre arbitre des crustacées, etc... Je les remercie aussi pour m'avoir faire découvrir des activités louches mais très amusantes comme les murder partys et pour les sessions de jeu de rôles. Je remercie particulièrement Valérian, irremplaçable organisateur d'évènement, faux je-m'en-foutiste et réel soutien. Arcady, pour être comme il est, lécheur de joue à ses heures perdues et membre de la guilde des moines uni-bras. Jérémy, vrai scientifique, casseur de chevilles ayant le cardio d'une grenouille fumeuse. Bob, franctireur de l'humour, franc tout court et savant vidéo-ludique. Jupi, maître incontestable du jeu de plateau et future game designer de renom. Charles, douteur maladif et MJ génial. Natacha, voleuse professionnelle efficace en combat. Sameh, vraie-fausse tricheuse dans tous les styles de jeu de l'univers, que j'adorais aller embêter à UT1. François, créateur de mélodies par lecteur de disquettes (entre autres!). Victor, expert en Taï-chi et en contradiction. Tom, génie de la punchline ayant un vrai métier et pas un post-doc. Luc Pons, archimage social et joueur de oud niveau 6. Luc Duzan, savant fou cycliste de l'elastic search. Maxime, ancien traumatisé repenti par ses profs de programmation. Florent, membre très haut d'Ingress (pas vraiment en vrai), esquiveur de poteau débutant. Elhadi, offreur de ventilateur-horloge de renom. David, vampire badass lanceur de 14d6 par attaque. Augustin, possesseur d'une attaque "Rire intensif" provoquant un état d'irritabilité avancé chez ses adversaires. Alexandre, plombier et découpeur de boutons professionnel. Seb, créateur de cellules qui contrôleront bientôt l'humanité. John, à qui le bonnet jaune va si bien et

faiseur de banane flambée approximative. Tanguy, alias Ignus le thaumaturge aka le grand mage de feu. Axel, compagnon vidéo ludique, de création et encore plus à venir. Une pensée particulière va aux différents stagiaires que j'ai vu passer dans l'équipe, qui ont participé de près ou de loin à ma montée en skill à Avalon. Je remercie aussi les doctorants tout frais arrivés, encore pleins d'enthousiasme, qui feront sans aucun doute de très bons docteurs: Maroun, JB, Bruno, Davide et Walid. Finalement, je remercie tout particulièrement Raja pour son soutien sans faille et qui, grâce à nos discussions, a permis à cette thèse d'aboutir.

Je remercie particulièrement mon ami André-Luc Beylot, pour ses cours d'une grande qualité, pour son amitié et ses conseils qui ont permis en premier lieu mon entrée en thèse.

Je remercie la famille de cœur, Didou, opérationnel sur les daurades et les poivrons sautés à la sauce soja, moins sur la mini-nes. Méla, pour sa joie de vivre, son humour et ses pétages de plombs. Laure, pour sa bonne humeur et son rire qui, combiné à celui d'Augustin (cf. plus haut) pourrait provoquer des mini-trous noirs. Bensou pour son côté fifou et les raclées à la pétanque.

Je tenais à remercier Camille, mon ami d'enfance, pour sa présence sans faille et pour les sessions nocturnes d'Amiga sur Populous, Eye of the Beholder, ADS Bateau, HeroQuest, SilkWorm, bref pour mes premiers souvenirs vidéo ludiques.

Je remercie ma famille proche à commencer par mon père qui sans le savoir a fait de moi un scientifique en me partageant sa curiosité enthousiaste. Je le remercie également pour m'avoir aidé sur les devoirs de maths et les sessions de feux d'artifices. Je remercie Renée qui a toujours été présente, pour sa gentillesse et les photos d'enfance. Je remercie ma mère pour son soutien et sa présence, notamment lors de mon départ difficile à Toulouse. Je remercie infiniment mes sœurs pour l'aide et votre présence réconfortante au cours de ces années, vous êtes tout simplement les meilleures. Je remercie aussi Pierrick, mon beau-frère, pour sa gentillesse et son virus des jeux vidéo. Je remercie mes neveux et nièces, Linda, Loïc, Justine et Inès, vous voir tout simplement grandir est un réel privilège et je suis réellement émerveillé de ce que vous êtes devenus. Je remercie la famille proche de Leslie, qui ont leur place dans ce paragraphe. Je remercie Philippe et Corinne, pour leur gentillesse et pour tous les moments qu'ils ont pris pour nous aider. Enzo, futur start du boum-boum, que je remercie pour les discussions musicales et les découvertes d'humoristes facebookeurs.

Je terminerai en demandant pardon à toutes les personnes que j'aurai oublié de remercier, ma tête étant prise encore par la présentation de l'oral. Oups, il me reste quand même une personne à remercier, moi-même. Ben oui quoi c'est la page des remerciements, pas des remerciements aux autres. Non ? Bon d'accord. Teddy donc, contente-toi de devenir un game designer de renom.

Trêve de plaisanteries, Leslie, je pense sincèrement que tu n'as pas conscience d'à quel point tu as ta place dans la réussite de ce doctorat. Si tu es à la toute fin, c'est parce que tu es la plus importante contributrice. Par biens des aspects, tu m'as aidé à mener à bien ce travail et tu as même été relectrice téméraire de ce document. Humainement, être à tes côtés est un privilège et une chance pour moi, je tenais à te remercier donc à la fois pour ta contribution à la réussite de cette aventure et te remercier par avance à la réussite des prochaines, quelles qu'elles soient. Je finis en te souhaitant la plus grande réussite possible pour ton propre doctorat et j'espère pouvoir t'apporter ne serait-ce que la moitié de ce que tu as fait pour moi durant ces années. Merci pour tout.

Contents

Introduction

Problems to solve have never been so complex. To design useful systems in this context, designers use more and more often a combination and/or an aggregation of heterogeneous systems that have been independently designed but are interdependent. The work of this thesis fits in the intersection between the area of Systems of Systems (SoS) and Adaptive Multi-Agent Systems (AMAS) and proposes to study what these two fields can mutually provide to each other.

Context of Study

Modern societies have never been so complex. We experience this complexity more and more often by the apparition of international issues that are huge threats for mankind: Economic crisis as 2008 sub-primes, global response to health threats such as Ebola, global warming and ethnic violence are a few but modern concrete examples. In 'The collapse of complex societies' [Tainter, 1988], the author explains a possible consequence of these threats called *"The doom and gloom scenario"*. This scenario explains that a society having reached a certain level of complexity will collapse if it faces huge economic and/or environmental changes as the Maya and the Roman civilizations. The author argues that the complexity of these societies leads to an inflexibility of their structures and consequently an unsuitability to changes. This "non-adaptation to changes" leads to the collapse of these civilizations. This inflexibility seems to come from the sophistication that conducts to a huge, interlocking dependent network of roles and responsibilities unable to change. The author finishes by writing that the state of our occidental societies is the same that Romans and Maya ones. The aim is then to avoid this scenario. As [Henshaw et al., 2013] explains, our societies have two resources that the previous ones did not have: Information Technology and better knowledge about our world, especially in engineering science. To cope with and solve these complex problems, science tends to deal with complex systems. As a first approximation, the difficulty of "understanding" complex systems comes from their own nature: the interconnections and the mutual influences between heterogeneous parts of the systems (physical, biological, social, economic and technological) lead to what seem to be unpredictable phenomena. Research seeks to answer global questions like how to analyze, design, develop, manage, simulate and use complex systems for the good of our societies [Baldwin and Sauser, 2009]. One part of this research area is focused on human-made complex systems giving rise to the research area of System Engineering (SE).

Managing Complexity through System Engineering (SE)

System Engineering (SE) appeared nearly after the World War II, as the academic and industrial response to the increasing complexity of engineering issues. This complexity leads to the needs of new methodologies to realize and conceptualize complex systems in a rational way. The adjective **complex** (for a system in SE) has the following definitions (ISO/IEC/IEEE 2010):

(a) An adjective describing a system's design or code that is difficult to understand because of numerous components or relationships among components.

(b) An adjective describing a system or component that has a design or implementation that is difficult to understand and verify.

As an example (of complex systems), military and astronomy science began to think about huge projects as the Apollo program. In this program, the realization of the lunar module is a good example of complex system because of the numerous components and interrelations. Its construction led to budget overruns, project management and testing problems. Then, from the early 50's to nowadays, researchers from various areas proposed new methodologies to realize systems in a controlled and efficient way (for example to control cost and quality of a project or a system). As a result of 50 years of research, SE is well documented and the creation of the International Council on Systems Engineering (INCOSE) is nowadays the responsible of its evolution. INCOSE still proposes new processes, models, methodologies to realize more and more complex systems. Finally, during the last 50 years, SE has enabled to analyze, manage, realize, and simulate systems by giving interdisciplinary tools such as processes and methodologies. For INCOSE [BKCASE Editorial Board, 2014], *" 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, then proceeding with design synthesis and system validation while considering the complete problem."*

From Classical Systems Engineering to Systems of Systems Engineering (SoSE)

Early in the SE history, the term **system of systems** was used but through a theoretical point of view. The first was [Boulding, 1956] whose idea is to obtain a *"spectrum of theories"* greater than the sum of its parts by imagining a *"gestalt"* in theoretical construction. It is only in 1989 that the Strategic Defense Initiative uses the term **System of Systems** in its modern sense: the integration of several existing complex systems to obtain a huge system owning capabilities that are not in the subsystems. The main idea is to switch the point of view of complexity (in the sense of SE) from the study of **one complex system** to a **network of integrated complex systems**. In the 90's, this change of paradigm in system engineering is confirmed in the literature: The objective was to address *"[...] shortcomings in the ability to deal with difficulties generated by increasingly complex and interrelated system of systems"* [Gorod et al., 2008]. Moreover, [Jamshidi, 2008b] argues that there was *"a growing interest in a class of complex systems whose constituents are themselves complex"*. The aim is to *"achieve synergy between these independent systems to achieve the desired overall system performance"*. This change of paradigm in

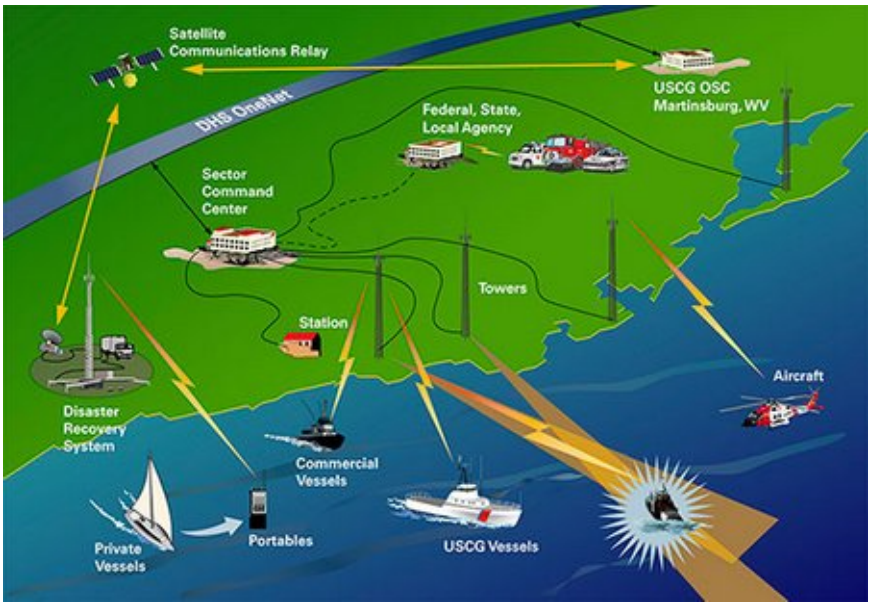


Figure 1 – Search And Rescue SoS Example

SE leads to re-think about all its methodologies, tools and processes: this is the beginning of SoS Engineering (SoSE). SoS Engineering is believed to implement and analyze more effectively large, complex, independent, and heterogeneous systems working (or made to work) cooperatively. Finally in the early 2000, numerous projects of SoS have proposed to study different research areas (COMPASS, T-AREA-SoS and so on). As a first example, figure 1 shows a SoS for Search And Rescue (SAR) missions and shows the use of heterogeneous systems such as commercial vessels, aircraft and satellites integrated together to fulfill SAR missions. Each individual system has not the ability to do alone its own SAR mission because of its limited perceptions and action fields, which is the reason why the integration of these systems in a SoS is important.

Importance of SoS Research

Scientists have already begun to work on new theoretical foundations, new methodologies and engineering tools to face the complexity of designing in a good way this kind of systems. After 20 years of research, the study of SoS and all research areas in SoS are still widely open. In 2008, [Jamshidi, 2008b] argues that *"No engineering field is more urgently needed in tackling SoS problems than SE. On top of the list of engineering issues in SoS is the engineering of SoS, leading to a new field of SoSE. How does one extend SE concepts like analysis, control, estimation, design, modeling, controllability, observability, stability, filtering, simulation and so on."* More recently, [Henshaw et al., 2013] referenced 113 research areas: Concept, definition, taxonomy, model, methodology of implementation, simulation and so on. They argue that *"The highly, and increasingly, connected nature of modern society means that the existence of SoS is an unavoidable factor of modern life, driving the need for better techniques to analyze, design for, manage, and retire SoS and the individual systems that contribute to SoS"*. Indeed, all classical

SE concepts are re-thought to match with new challenges of SoSE. Furthermore, in 2015, [Axelsson, 2015] argued that even after 3000 papers and 20 years of research: *" [...] there are signs of immaturity within the research area, with only limited use of systematic empirical methods those are common in other domains, and also that new research results are not building systematically on previous research."* Then, current research on SoS focuses on a large variety of problems [Selberg and Austin, 2008] to develop new methods of engineering or architecting the parts of the SoS (called component systems).

Generic SoS Model and Dynamic Architecture

Generic SoS models exist but have some limitations concerning their definition of the SoS environment and the interactions of the component systems.

Concerning [BKCASE Editorial Board, 2014], the challenges in architecting SoS come from the managerial and operational independence of component systems of the SoS. These inherited component systems are not designed to fulfill the global goal of the SoS but the combination of individual component systems can lead to the emergence of unexpected behavior. The independence of the component systems leads to re-think the methodologies of architecting: contrary to SE where the architecting of a system was mostly static, in a SoS this one is dynamic.

As a first definition, SoS architecting research focuses on how, in an efficient manner, a SoS can have a dynamic, network-centric and collaborative architecture [Jamshidi, 2008a] and how the dynamic architecture can lead to the emergence of new capabilities. SoS literature shows that Agent-Based Modeling (ABM) is a natural way to develop this new kind of architecture [Azani, 2008] [Caffall and Michael, 2009]. More precisely, ABM and Agent-Based Simulation (ABS) enable to describe and test new architecture dynamics by varying the behavior of component systems in the SoS. [Azani, 2008] proposes a set of principles for efficient SoS Architecting based on open systems such as the **Open interface principle** stating that *"Open systems have permeable boundaries that allow them to exchange mass, energy, and information with other systems"* or the **Synergism principle** stating that *"that the cooperative interaction between component systems has a greater effect in their combined efforts than the sum of their individual parts. Essentially, this is what gives rise to emergence"*. Several architecting methodologies exist but to our knowledge none respects all of these principles.

Contribution

We propose to contribute to the SoS research area by proposing a new generic SoS model called SApHESIA (SoS Architecting HEurIStIc based on Agent) based on set theory which takes into account the SoS main characteristics found in literature and extend the notion of environment and interactions between component systems. We also propose to contribute to SoS architecting by developing a new SoS architecting heuristic respecting the principles for efficient SoS Architecting based on open systems such as the **Open interface principle** and the **Synergism principle** and where the interoperability between component systems is supposed to be achieved. This heuristic, based on a multi-agent approach called AMAS

(Adaptive Multi-Agent System), uses cooperation between agents enabling them to self-organize and self-adapt to the dynamics of the environment and where the adequate functionality of the system emerges from the interactions of the agents. Another contribution is a set of architecting tools based on SApHESIA enabling to test our architecting heuristic on several applications. As our work is focused on SoS architecting, we evaluated our work on SoS architecting problems found in literature and self-made problems because no real case studies with usable data have been found.

Organization of the Document

This thesis is divided in three main parts:

- the first one, related to the state of the art, focuses in a first time on SoS definitions and existing generic SoS models. We propose a working SoS definition and highlight limitations about existing generic SoS models. Existing SoS architecting heuristics as well as their limitations are also presented. Then, a differentiation between classical SoS simulations and SoS Architecting Heuristic Simulations (SAHS) and existing paradigms of simulation for SAHS are introduced. Finally, The AMAS approach is presented.
- the second one, related to the contribution, presents in a first time the new generic SoS model (called the SApHESIA model) we have defined to fulfill the limitations addressed in the state of the art. Then, our new SoS architecting heuristic based on cooperation between component systems is exposed. Finally, the SApHESIA procedure we implemented for SAHS is presented.
- the last one, related to the evaluation of our contribution, first presents the simulation of a SoS problem called the Missouri Toy Problem to evaluate if our heuristic is able to solve it. Then, an experimentation concerning an UAV obstacle avoidance system is exposed to compare our approach with satisficing games that are the basis of the SoS collaboration formation (another SoS architecting heuristic presented in the state of the art). Experimentation about SoS coupling is introduced through the coupling of the two first experimentations. Finally, the last chapter presents the problem of the criticality adaptation and proposes an algorithm to solve this issue.

Finally, we end this document by concluding our work and presenting perspectives.

Part I

State of the Art

1 SoS Definitions and Existing Generic SoS Models

This first chapter presents the foundations as well as the concepts used in this thesis. More precisely, the aim of this chapter is triple: (i) to enable the reader to understand what a SoS is; (ii) to propose a SoS definition useful for our work; (iii) to present and discuss about existing generic SoS models.

In this way, we will focus in a first part on SoS literature concerning definitions and characterizations. This part is needed because a lot of different SoS definitions have been given during the last 20 years and at this time no definition of SoS is widely accepted. After the presentation of existing summaries and reviews of these definitions, we propose a working definition of SoS that will be a basis for this thesis work. During this first part, several other concepts that have not consensual accepted definitions are also presented.

Then, this chapter proposes a state of the art about generic SoS models and their limitations. The presentation of generic SoS models is needed because they are the basis of SoS architecting heuristics. The limitations are exposed through evaluation criteria we defined thanks to the proposed definition of SoS of the first part. These limitations serve as arguments to propose a new generic SoS model introduced in chapter 5 of this document.

1.1 Main Characteristics, Classifications and Definitions of SoS

As written in the introduction, concepts of SoS and SoSE (SoS Engineering) appeared when Systems Engineering (SE) focused on how to interconnect a group of more or less independent complex systems to fulfill a higher goal. Then, first research works have consisted in collecting real cases of SoS and trying to find a general definition as well as a taxonomy attached to this concept. Unfortunately, no consensual definition of SoS exists nowadays. Indeed, a lot of definitions has been given by many authors coming from different areas [Jamshidi, 2008b], [DoD, 2006], [CARLOCK and FENTON, 2001], [Manthorpe, 1996]. Some of them are not coherent with each other and sometimes too domain specific or unclear. Then, these definitions do not enable to propose a generic definition and common characteristics of SoS. Thus, research efforts have been put on SoS taxonomy and theoretical foundations for finding a generic definition by:

- analyzing case studies and trying to sum up what are the SoS main characteristics

[Maier, 1998];

- collecting and trying to generalize existing definitions [Jamshidi, 2008a];

- analyzing SoS literature and finding generic SoS properties [Boardman and Sauser, 2006].

In this section we propose an overview of the main research about definitions and characteristics of SoS.

1.1.1 Main Characteristics of SoS

Maier in [Maier, 1998] was the first to characterize SoS by giving five main characteristics, accepted by researchers working on SoS ([Jamshidi, 2008a],[Selberg and Austin, 2008]), whose distinguish a SoS from a traditional complex system. The first two characteristics are expressed in this definition: *"A system-of-systems is an assemblage of components which individually may be regarded as systems, and which possesses two additional properties: (1)* ***managerial independence of the components*** *and (2)* ***operational independence of the components****."*
Then, through some examples, Maier finishes by giving the last three characteristics : **the geographical distribution**, **the emergent behavior** and **the evolutionary development processes**.
The **operational independence** means that a component system removed from the SoS continues to operate independently (i.e., it can continue to work even if it does not belong anymore to the SoS).
The **managerial independence** means that the component systems are managed by independent users: *" [...] the component systems not only can operate independently, they do operate independently"* [Maier, 1998]. In other words, a component system takes its own decisions concerning what it has to do.
The **geographical distribution**, has no strict definition in literature. But this characteristic exists because the nature of the interfaces between components is not the same in a SoS than in a classical system. In a classical system, components are less geographically distributed enabling a tight coupling through *"power, material as well as communication interfaces with limited bandwidth and delay issues"* [Maier, 1998]. In a SoS, the geographical distribution disables material and power interfaces between component systems.
The **emergent behavior** of the SoS is defined as the production (at the SoS level) of an overall behavior which was not implemented (and so not predicted) in the component systems. As an example, Maier gives the Internet : *"The Internet exhibits a rich set of emergent behaviors represented by the complex distributed applications that run on top"* [Maier, 1998].
Finally, the **evolutionary development** is defined as a dynamic development that takes care and integrates the changes (of structure, of use and so on) that occur within the SoS during time. For example, it can be the change of communication processes such as, with the same example of the Internet, the change from the IPV4 protocol to the IPV6 one.

1.1.2 SoS Classification

Maier also proposes a classification of SoS [Maier, 1998] that has been extended by Dahmann and Baldwin in [Dahmann et al., 2009]. This classification lays on the existence and the coercive power of a central management. If this one exists, it is presented as "the head" of a SoS. The term coercive power represents the degree of subordination of a component system in a SoS: a strong management power means a strong subordination of component systems. Generally, when a central management exists, it defines the SoS objectives and component systems have to fulfill them. This classification is explained in the following paragraphs, going from the one with the most centralized management to the less one.

Directed SoS: The SoS has been created to fulfill a clearly established goal and component systems are totally subordinated to the SoS. A directed SoS is fully centrally managed by the stakeholders of the SoS (i.e., persons who own the SoS). Component systems have always the possibility to operate independently, but their normal operational mode is subordinated to the central managed purpose, even if the stakeholders of the component systems have their own objectives. It implies that component systems can operate for the SoS at their own detriment (i.e., against their own objectives).

Acknowledged SoS: The component systems and the SoS have their own objectives and their own defined resources (human, raw materials,...). SoS tends to fulfill its objectives by leveraging functionalities of component systems or changes that may be, for example, the addition of a new functionality. The participation of a component system to the SoS is based on cooperative agreements between the SoS and the component system. These cooperative agreements can be based, for example, on resource sharing. Modern military operations are often instantiated with this type of SoS.

Collaborative SoS: The collaborative SoS has no centralized objective but component systems share a common interest and are voluntarily governed to support this interest. If a central management organization exists, this one has no coercive power on component systems. Finally, the belonging of a component system to the SoS is often negotiated between the component system and the SoS. For example, the IETF (Internet Engineering Task Force) is a group of scientists, industrialists, academics having a common interest: to build an efficient structure to develop their activity (the Internet). In this way, they work on and propose standards but IETF has no power to enforce these standards.

Virtual SoS: The SoS has neither central management nor central purpose. A large scale behavior can emerge (and may be desired) and the SoS must rely on shared mechanisms between component systems to maintain it. As an example, Maier [Maier, 1998] gives the World Wide Web: *"the control has been exerted only through the publication of standards for resource naming, navigation, and document structure. Web sites choose to obey the standards or not at their own discretion"*.

It appears that this classification is related to the degree of component systems inde-

	Directed	Acknowledged	Collaborative	Virtual
Central management existence	++	++	+	- -
Central management power	++	+	-	–
Component systems autonomy	-	+	++	++
Clear objective at SoS level	++	+	-	–
Large-Scale behavior emergence	-	-	+	++

Table 1.1 – Classification summarization of SoS

pendence (also called autonomy) from SoS. In a directed SoS, component systems have less autonomy than in a virtual one. More generally, the classification of SoS types can be characterized with the following properties:

- the existence of a central management for the SoS;
- the subordination of component systems to this central management;
- the component systems autonomy which is the capacity for a component system to decide itself for itself;
- the definition of a clear objective at the SoS level;
- the emergence of a large-scale behavior.

Table 1.1 summarizes the types of SoS according to the presented properties. The legend means that the evaluation criterion for the type is totally present (++), almost totally present (+), almost totally absent (-), totally absent (- -).

This classification is important for the study or the creation of SoS architecting heuristic. The notion of SoS architecting heuristic, developed in chapter 2, can be briefly defined here as finding how component systems can work together by changing their interactions to fulfill the SoS goals with effectiveness (i.e., adequate to accomplish the SoS goals) and efficiency (i.e., functioning in the best possible manner with the least waste of time and effort). Proposing an architecting heuristic for a directed SoS is not the same work than for a collaborative one, because the existence of a central management between a directed SoS seems to forbid the use of a total decentralized decision heuristic. As another example, [Maier, 1998] presents the consequences of a misclassification of a directed SoS into a collaborative one: the persons representing the central management of the SoS can take decisions that may not have the expected effects because of a lack of coercive power.

This classification shows that the SoS study is a difficult issue: a SoS can (or not) have a central management, the component systems are managerially and operationally independents but at the same time they can be subordinated to the SoS central management. This classification makes harder the choice of a SoS model and a SoS architecting heuristic.

1.1.3 Existing Definitions of SoS

In 2010, Crossley [Crossley and Professor, 2004] distinguished the notion of SoS from the notion of Family Of Systems (FoS) found in Department Of Defense (DoD) documents. A SoS is *"a set or arrangement of interdependent systems that are related or connected to provide a given capability."* A family of systems is *"a set or arrangement of independent (not interdependent) systems that can be arranged or interconnected in various ways to provide different capabilities."* But this distinction is not really applied in the main literature of SoS. For example, in another DoD document of 2006 [DoD, 2006]: *"Finally, this study did not include Family of Systems (FoS) because these too are not well defined, and have their own varying characteristics to that of a SoS"*. In the same document, 32 definitions of SoS are given. These definitions come from different sources: journals, conference proceedings, academic studies, industry studies and so on. We will not give here all of them but this consequent number leads to an important conclusion: SoS definition depends on the current needs of the person who gives the definition. Jamshidi has the same conclusion after a survey and a discussion on the 6 following definitions in [Jamshidi, 2008a]. These definitions concerns SoS, SoSE and Enterprise SoS, which are strong related notions. In his paper, he argues that *"the field has a large vacuum from basic definition, to theory, to management and implementation"*. This work enables to have a large view of the SoS paradigm.

Definition 1 Systems of Systems exist when there is a presence of a majority of the following five characteristics: operational and managerial independence, geographic distribution, emergent behavior, and evolutionary development [Jamshidi, 2008a].

Definition 2 Systems of Systems are large-scale concurrent and distributed systems that are comprised in the class of complex systems [CARLOCK and FENTON, 2001].

Definition 3 Enterprise Systems of Systems Engineering is focused on coupling traditional systems engineering activities with enterprise activities of strategic planning and investment analysis [CARLOCK and FENTON, 2001].

Definition 4 System of Systems Integration is a method to pursue development, integration, interoperability, and optimization of systems to enhance performance in future battlefield scenarios [R.S, 2000].

Definition 5 SoSE (SoS Engineering) involves the integration of systems into systems of systems that ultimately contribute to the evolution of the social infrastructure [Lukasik, 1998].

Definition 6 In relation to joint war-fighting, system of systems is concerned with interoperability and synergism of Command, Control, Computers, Communications, and Information (C4I) and Intelligence, Surveillance, and Reconnaissance (ISR) Systems [Manthorpe, 1996].

Definition 1, the most generic among the six, is based on Maier characteristics, but is very poor: it gives little information about component systems and the environment. Are the component systems tends to be complex or not? What about the dynamics of the environment?
Definition 2 considers a SoS as a complex system. In this context, complex has to be understood as *"an adjective describing a system or component that has a design or implementation that is difficult to understand and verify (ISO/IEC/IEEE 2010)"*. But once again, component systems complexity is not addressed and he describes component systems as **concurrent systems**. **Concurrent** is not used in the SoS literature. For example, Maier in [Maier, 1998] defines component systems behavior as **cooperative**. Even if the term cooperation is not defined, this behavior seems semantically far from the notion of concurrence.
Definitions 3, 4 and 5 do not directly concern SoS. Definition 3 only deals with Enterprise SoSE, it is then a really domain specific definition. Definition 4 mentions the SoS integration which is a phase of SoS development applied in military area. Definition 5 concerns the already presented notion of SoSE. Definition 6 is a military oriented SoS definition contextualized by joint war-fighting that is a problematic of the 90's of the US defense focusing on how to use efficiently different independent systems in a global joint force.

Each definition has its own specificity, but all of them are too domain specific and do not enable to generalize SoS. Thus, Jamshidi proposes a definition that received a substantial attention: *"A SoS is an integration of a finite number of component systems which are independent and operable, and which are networked together for a period of time to achieve a certain higher goal."*

This definition is generic because there is no reference to any particular domain. Nevertheless, this definition seems to be incomplete: there are few details about component systems and the environment the SoS evolves in. The next sections analyzes the characterization of SoS in order to improve this last definition.

1.2 Characterization of SoS

Another approach to define SoS is to find their generic characteristics. Indeed, Gorod in [Gorod et al., 2008] argued that a characterization is more optimal to understand, study and model SoS. But, as for SoS definitions, different authors as Sauser [Boardman and Sauser, 2006] and Bjelkemyr [Bjelkemyr et al., 2007] argued that there is no consensus of what the main characteristics of SoS are: some authors use and expand the characteristics of Maier, some others do not. Nevertheless, there have been some convergences between these characterizations. In this way, [Boardman and Sauser, 2006], [Baldwin et al., 2015] and [Bjelkemyr et al., 2007] propose a review of SoS literature and propose characteristics to distinguish SoS from classic systems.

1.2.1 ABCDE Characteristics

In [Boardman and Sauser, 2006], authors extract from analysis of the SoS literature five main characteristics used to reason about the differences between classical systems (composed of parts) and SoS (composed of component systems) through these characteristics. These are

the ABCDE (**Autonomy, Belonging, Connectivity, Diversity** and **Emergence**) characteristics:

Autonomy is a characteristic derived from the operational and managerial independence of Maier. It refers to the independence of the systems from each other and from the SoS itself. Authors argue that most of the time parts of classical systems do not have autonomy on their own, or if they have, they do not use it and fully serve the autonomy of the classical system. For example, a brake in a car has no autonomy by itself. The specificity of a SoS, concerning autonomy, is that the component systems are autonomous and use their autonomy in order to fulfill the goal of the SoS.

Belonging of a component system is a characteristic representing the balance between its Autonomy and the loss of a part of its independence for the benefit of the SoS goals. It is also the ability to accept assistance from others component systems and the need to make a valued contribution to the goals of another component system. Then, the parts of a classical system belong totally to it as *"family members belong to the same family"* [Boardman and Sauser, 2006]. Once again, a car brake belongs to a given car and cannot be used in another one. In SoS, component systems have the choice to belong or not to a SoS by negotiating and evaluating what the belonging can give to fulfill their own purpose. The existence of belonging negotiation depends on the SoS type. For example, a directed SoS directly decides the belonging of a component system (i.e., the SoS central management).

Connectivity is the *"capability to form connections as needed to benefit the entity"* (meaning here a component system) [Boardman and Sauser, 2006]. In classical SE (Software Engineering), it is a golden rule to hide huge connectivity between parts by merging them to reduce connectivity between major parts of the classical systems. In SoS, connectivity is chosen by component systems and evolves during time contrary to classical systems that have static connectivities between parts.

Diversity references the notion of variety in a system. Variety is a direct reference to the law of requisite variety of Ashby [Ashby, 1956]: to maintain its stability, a system must have a level of variety as least equals to the variety of the environment (the outside of the system) it evolves in. This variety enables to respond to *"rampant uncertainty, persistent surprise and disruptive innovation"* [Boardman and Sauser, 2006] (i.e., the changes in the environment). Diversity is related to the notion of heterogeneity. In classical systems, a good practice in SE consists in reducing it through the principle of parsimony, also called Occam's razor. The aim is to reduce as much as possible diversity in a system in order to reduce redundancies, ambiguities and inconsistencies. In SoS, this diversity is a desired characteristic because the more diversity in a SoS the more it can cope with the dynamics of the environment. Diversity can be increased for example, by adding new component systems into the SoS.

Emergence means the apparition of phenomena or sophistication in a system that is not issued from the simple sum of its parts. Steve Johnson in [Johnson, 2001] reports emergence

Element	Classical System	System of Systems
Autonomy	Parts of the system have no autonomy or have ceded it in order to grant autonomy to the system.	Autonomy is exercised by component systems in order to fulfill the purpose of the SoS.
Belonging	Parts are akin to family members; they did not choose themselves but came from parents. Belonging of parts is in their nature.	Component systems choose to belong on a cost/benefits basis; also in order to cause greater fulfillment of their own purposes, and because of belief in the SoS purpose.
Connectivity	Prescient design, along with parts, with high connectivity hidden in elements, and minimum connectivity among major subsystems.	Dynamically supplied by component systems with every possibility of myriad connections between component systems, possibly via a net-centric architecture, to enhance SoS capability.
Diversity	Managed, i.e., reduced or minimized by modular hierarchy; parts' diversity encapsulated to create a known discrete module whose nature is to project simplicity into the next level of the hierarchy.	Increased diversity in SoS capability achieved by released autonomy, committed belonging, and open connectivity.
Emergence	Foreseen, both good and bad behavior, and designed in or tested out as appropriate.	Enhanced by deliberately not being foreseen, though its crucial importance is, and by creating an emergence capability climate, that will support early detection and elimination of bad behaviors.

Table 1.2 – Classical systems vs SoS [Boardman and Sauser, 2006], [Baldwin et al., 2015]

to be *"the movement from low-level rules to higher-level sophistication"*. Classical systems tend to show foreseen emergent behaviors (i.e., behaviors that can be determined by specifying interactions among constituent systems [Benites Gonçalves, 2016]). In SoS, the aim is to create conditions where both foreseen and unforeseen emergent behaviors (i.e., behaviors that are unplanned and can dynamically appear during SoS operation [Benites Gonçalves, 2016]) appear to cope with the dynamics of the environment.

This characterization enables to strongly distinguish classical systems from SoS. Table 1.2 from [Boardman and Sauser, 2006] and [Baldwin et al., 2015] sums up this distinction between classical systems and SoS through these ABCDE characteristics. The SoS literature analysis and the definition of the five characteristics are a huge effort to propose a generic vision of what a SoS is. Nevertheless, little information addresses the environment the SoS evolves in: how it can be defined? What are the characteristics of this one? Concerning the component system, what is the level of complexity of this one compared to the SoS?

1.2.2 Characterization through SoS Concept Maps

To improve understanding of SoS, [Bjelkemyr et al., 2007] proposed a concept map showing to discriminate: **Complex Systems, Non-Complex Systems, Non-SoS Complex Systems** and **SoS**.

According to his opinion, these four classes enable to classify any system. The main criterion for this classification is *"the degree of exhibition of complex properties"*. These complex properties are composed of the 5 Maier's criteria defined in section 1.1.1 (Managerial and operational independence of component systems, geographical distribution, emergent behavior and evolutionary development of the SoS) and completed by the notions of **heterogeneity** of the component systems, **networking** and **trans-domain**. Concerning **heterogeneity**, author defines it as component systems that are *"of significantly different nature, different elementary dynamics that operate on different time scales."* Concerning **networking**, author writes that *"networks define the connectivity between independent systems in the SoS through rules of interaction."* **Trans-domain** property means that an effective study of SoS requires unifying knowledge across fields of study: engineering, economy, policy, and operations.

These properties enable to distinguish Complex Systems from Non-Complex Systems: a Non-Complex System does not exhibit complex properties, has clear boundary and purpose, instead of a Complex System that must display some of these properties. Then, Complex Systems can be divided into **Non-SoS Complex Systems** and **SoS Complex Systems** (or simply SoS). A Non-SoS Complex Systems exhibits a few of complex properties and a SoS exhibits most of them. Finally, a SoS is composed of Non-SoS Complex Systems (i.e., not recursively composed of SoS). Authors argue that component systems can show complex properties but have not the same level of complexity than the SoS. Once again, we can point out that the SoS environment is not defined, not even addressed.

Finally, Gonçalves in [Gonçalves et al., 2014] extends the concept map by developing notions contained in the concept map of [Bjelkemyr et al., 2007] and adding new ones coming from the definitions presented before. The emergent behavior (that is not developed in [Bjelkemyr et al., 2007]) of the SoS comes from the cooperation between the component systems through their ability to connect with each other thanks to their operational independence. The environment of the SoS exists but is not developed. Gonzales introduces also the concept of Software-Intensive System of Systems (SiSoS) that is a subclass of SoS where component systems are strongly composed of software. This notion of SiSoS will not be reused in the rest of the document because we want to propose the most generic SoS model as possible.

1.3 Working Definition of SoS

Up to the end of this document, we propose the following working SoS definition which sums up what we found in the literature. We try to be as close as possible to existing defini-

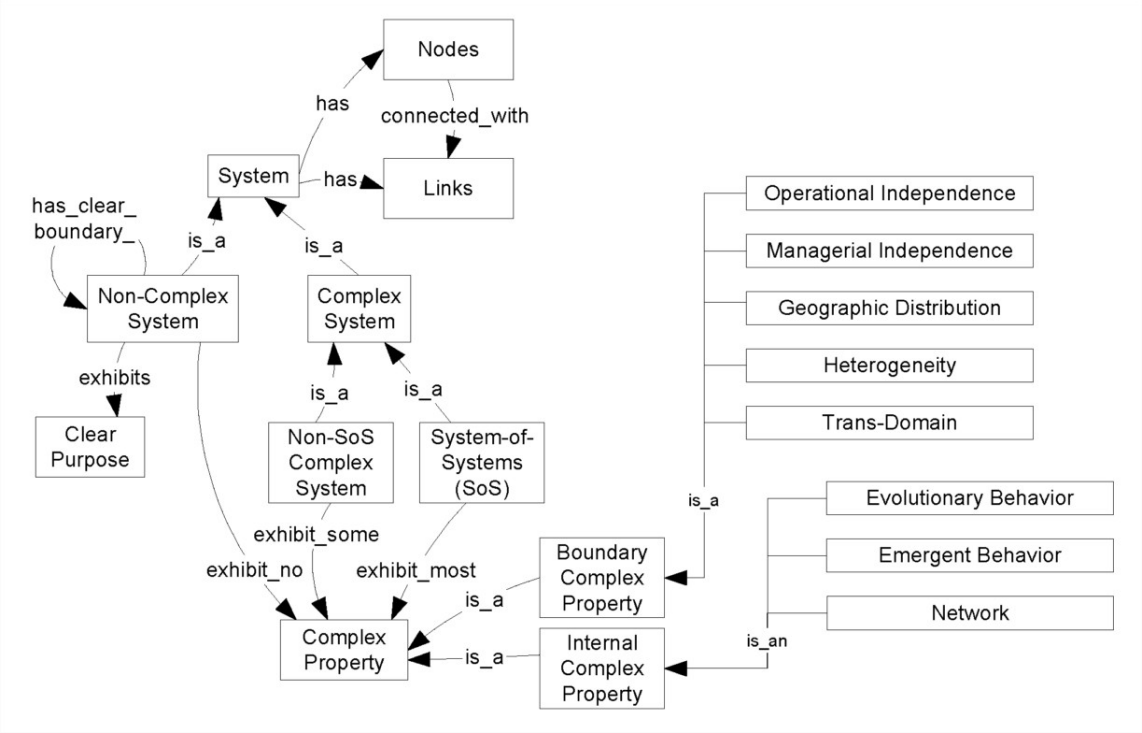


Figure 1.1 – Concept map of a SoS [Bjelkemyr et al., 2007]

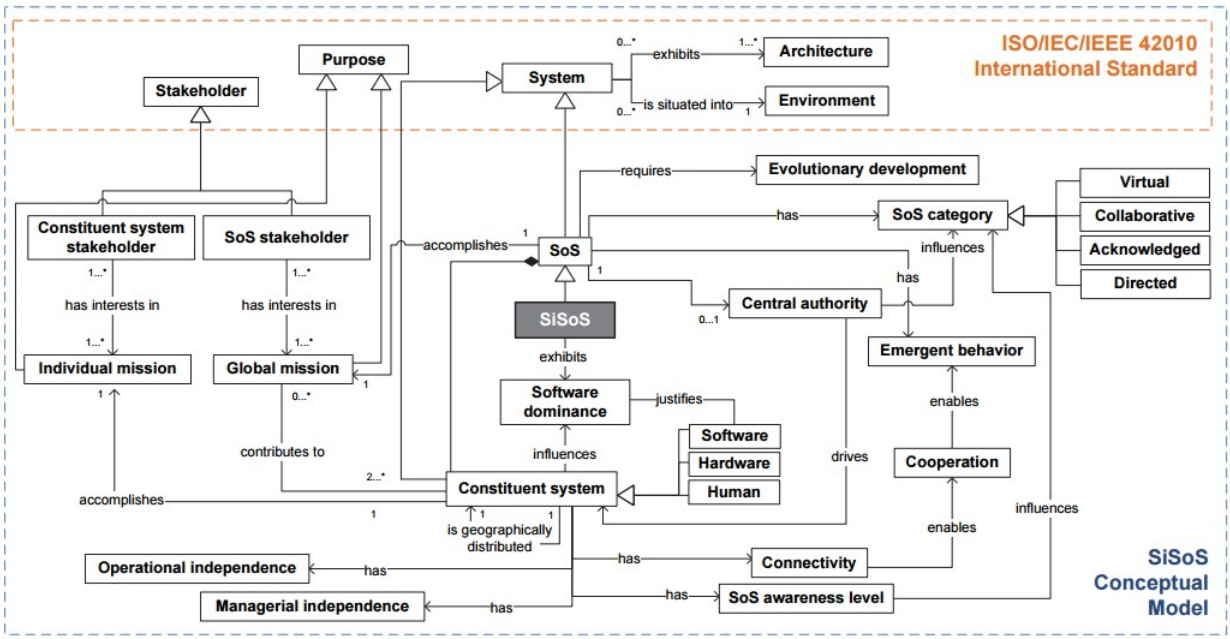


Figure 1.2 – Concept map of a SiSoS [Gonçalves et al., 2014]

tions and characterizations:
A SoS is a system composed of systems (called component systems) that tends to be complex and tend to evolve in a dynamic environment. A SoS can have a central management with its own objectives and can use subordination to force component systems to act as desired. Component systems and SoS have the Maier characteristics:

- operational and managerial independence of the component systems (also called the autonomy of a component system);
- geographical distribution of the SoS;
- interactions and cooperation of component systems tend to provoke emergent behavior at the SoS level;
- evolution of the SoS during time: interactions are dynamics and enable openness (component systems can join or leave the SoS at runtime).

Component systems tend to be less complex than the SoS they evolve in, but can show complex properties such as operational and managerial independences. Component systems tend to be heterogeneous in terms of capabilities (i.e., actions) from each other and interact with each other through their capabilities or their environment. The belonging of a component system to the SoS depends on the type of the SoS. In a directed SoS, the central management chooses if the component system belongs to the SoS. In an acknowledged SoS, this is a mutual agreement. In collaborative and virtual SoS, the existence of interactions defines the belonging of a component system to the SoS. The SoS tends to show emergent behaviors that can be foreseen or not. The emergent behavior is enabled by the cooperation of the component systems through their interactions. The environment in which the SoS evolves is dynamic and composed of systems (not own by the SoS) that work independently from the SoS and may have concurrent objectives from the SoS ones. If a SoS represents a firm and its suppliers, the environment can be, for example, composed of concurrent firms.

1.4 Existing Generic SoS Models

As explained in the introduction, finding a generic SoS model is the basis to propose a new SoS architecting heuristic. Firstly, this section enumerates and justifies the evaluation criteria chosen from literature to evaluate generic SoS models. Secondly, it presents two generic SoS models found in the literature and discusses their limitations.

1.4.1 Evaluation Criteria for SoS Model

We propose nine criteria to model SoS to evaluate existing generic SoS models and show their limitations. The five first criteria concern the ability to model the following characteristics of a component system (that represent the Maier's criteria):

1. **Heterogeneity**;

2. **Managerial independence**;
3. **Operational independence**;
4. **Geographical distribution**;
5. **Interactions between component systems**.

Finally, the following criteria concern the ability to model the SoS and the environment:

6. **SoS model:** it concerns the ability to model SoS as an autonomous entity with its own goals. Indeed in *Directed* or *Acknowledged* SoS, a central management exists and has to be modeled.
7. **Dynamic environment model:** it concerns the ability to model a dynamic environment. It has been chosen because a SoS evolves in a dynamic environment.
8. **Global expressiveness:** it concerns the ability to express interesting problems. It has been chosen because a model has to be expressive enough to model interesting SoS. For instance, it is better if a model allows to model precise examples. In other words, a too simple model will fail to model concrete and interesting examples.
9. **Metric definitions:** it concerns the ability to define useful metrics such as for example, the global cost or the performance of the SoS. A model has to enable the evaluations of instances created with it.

1.4.2 The Basis of Generic SoS Model: Agent-Based Model Paradigm

As existing generic SoS models are based on Agent-Based Model, this section introduces this paradigm. [Bonabeau, 2002] defines ABM as a kind of model uses to describe the system as a collection of entities called agents. In [Gilbert, 2008], an ABM can be defined as a *computational method that enables a researcher to create, analyze, and experiment with models composed of agents that interact within an environment*. In [Squazzoni, 2012], agents can *contain heterogeneous variables, parameters, and behavior. Agents could interact by exchanging information and via communication protocols, and can react to the environment, learn, adapt, and change rules of behavior. Modelers can therefore equip computational agents with cognitive and behavioral properties typical of human agents, while the environment (i.e., social structures and institutions) can be programmed to mimic the real social world in varying degrees of detail.* Then, each of these agents is able to make its own decision from its behavior, the knowing of its situation and its environment. The decision of an agent leads to interactions with other agents or with its environment. The behavior ranges from simply reactive behavior to cognitive one.

The natural vision of SoS as a set of component systems linked to fulfill a higher goal is close to the ABM paradigm. For example, the notion of managerial independence of a component system is really close to the decision autonomy of an agent. The notion of operational independence of component system is close to the autonomy of action of an agent. Finally, interactions between component systems (such as information or resources exchanges, providing services and so on) can be modeled by interactions between agents. Indeed, in ABM,

	SoS	ABM
Parts	Component systems	Agents
Autonomy of the parts	X (Managerial and operational)	X
Dynamic interactions between parts	X	X
Complexity of the parts	X	X
Heterogeneity of the parts	X	X
Important notion of environment	X	X

Table 1.3 – Concepts similitude between SoS and ABM paradigm

an agent has the ability to perceive, to communicate and to exchange information with other agents. It is clear that ABM paradigm is a natural way to model SoS. These similarities of concepts (summed up in table 1.3) explain why ABM has been used to study SoS and propose new ways to architecture them. The following sections focus on 2 generic models based on ABM.

1.4.3 Existing Generic Model for SoS

There are a few generic SoS models in literature. Based on set theory and ABM, the following models enable to have formal definitions of what is a SoS. Nevertheless, we discuss their limitations through the evaluation criteria defined in section 1.4.1.

a) Generic Model of SoS Based on Set Theory

This model, introduced in [Baldwin and Sauser, 2009], uses set theory and first order logic. It has been created to model the five keys characteristics developed in [Boardman and Sauser, 2006] (see section 1.2.1) that are **Autonomy, Belonging, Connectivity, Diversity** and **Emergence**. In this model, a component system S_i is defined as a tuple:

$$S_i = \{A_i, G_i, E_i\}$$

with:

- A_i, a set of actions;
- G_i, a set of goals;
- E_i, a set of non-defined elements.

A_i, G_i are sets of atomic elements. There is no detail about what are exactly these elements but they represent respectively the available actions and the goals of a component system. E_i is a set of undeclared elements where a subset $C_i \subset E_i$ (called the set of connectivities and composed of couples of component systems) represents the connections between component systems.

Finally, a system of systems $S*$ is defined as a set of sets:

$$S* = \{S_1, ..., S_n, G*\}, n \in \mathbb{N}, G* \neq \emptyset$$

With :

- $G*$ is the set of the SoS goals;

- $S_1, .., S_n$ are component systems.

With this model, some metrics [Baldwin and Sauser, 2009] are used to proportion the level of the main characteristics presented in section 1.2.1: **Autonomy, Belonging, Connectivity, Diversity** of each system (**Emergence** and **Connectivity** being not represented). The level of autonomy of S_i is defined as :

$$Autonomy_i \equiv |A_i|$$

The justification of this definition is that if a system wants to fulfill its own goals, a system has to have the most actions as possible (i.e., the more actions a component system has, the more autonomy it has).
Belonging of a component system in a SoS represents the balance between autonomy and the loss of a part of its independence for the benefit of the SoS Goal (section 1.2.1). Then, the belonging of a component S_i to the SoS $S*$ is defined as:

$$Belonging_i \equiv \frac{|f(A_i) \cap G*|}{|A_i|}$$

Where:

$$f : A_i \rightarrow G* | (\forall g \in G*)(\exists A_i \in S_i)(\exists a \in A_i)(f(a) = g)$$

$f(A_i)$ represents the goals of $S*$ for which S_i is able to contribute with its actions A_i. *Belonging* is then represented by the proportion of what the component system can do for the SoS. More the component system can achieve SoS goals through its actions, the more it belongs to the SoS. Then, this result is normalized with the number of actions of S_i ($|A_i|$). Finally, the level of belonging is high when actions of S_i tends to fulfill SoS goals.
Thus, authors define a metric on the **Diversity** of a SoS with all the different sets of available actions A_i of each component system i ($i \in Z^+$). This following definition ensures that the A_i have multiple elements and they are different from each other:

$$Diversity \equiv A_i \neq \emptyset \wedge |A_i| > 1, i \in \mathbb{Z}^+$$

b) Agent-Based Wave Model

In [Acheson et al., 2012], authors propose a generic SoS model based on ABM, which enables the use of a formation heuristic based on a genetic algorithm. In this model, each component system S_i is defined as a set of sets:

$$S_i = \{c_i, p_i, willingness_i, ability_i\}$$

With :

- ▷ c_i, a capability, which is similar to an action;
- ▷ p_i, a performance on the capability c_i;
- ▷ $willingness_i$, a metric representing the willingness of S_i to cooperate with the SoS ;
- ▷ $ability_i$, a metric representing the ability of S_i to cooperate with the SoS.

$willingness_i$ and $ability_i$ are used in a function f that is not described but that enables to know if S_i will cooperate with the SoS (i.e., if the component system will belong to the SoS or not). Then, a component system with a high $ability_i$ tends to be more cooperative with the SoS.

A SoS is represented by :

$$SoS = \{C, W, P\}$$

With:

- ▷ C, a set of desired capabilities c_i;
- ▷ W, a set of weights concerning the set of desired capabilities C;
- ▷ P, a set of desired performances on capabilities in the set C.

Desired capabilities are the capabilities that designer wants the SoS to achieve. It is composed of a subset of component systems capabilities. The P set enables to express a certain level of desired performance on desired capabilities C for the SoS. As an example, let's pretend that one of the SoS objectives is to have the capability c_3 with a performance of 80%. Then, it can be modeled by adding c_3 in C and $p_3 = 0.8$ in P. W contains weight w_i on capabilities modeling the willingness for the SoS to have a capability c_i (i.e., if w_i is high, c_i is highly wanted by the SoS).
The environment is slightly addressed and detailed by a set of external factors such as *SoS funding*, *Threats* and *National priorities*. These factors do not have formal descriptions. Even if it is not explicitly written in [Acheson et al., 2012], SoS funding seems to model the amount of funds allowed by stakeholders for the SoS functioning. This fund is used to be compared with the global cost of the SoS, which seems to be the sum of component systems costs (the information of how these costs are modeled is not given). Finally, there is no example of use of threats and national priorities factors.

c) Discussion

The first generic model enables to model a component system through the set S_i. The concept of heterogeneity is represented thanks to the set of actions A_i that can be different

from component systems. Operational and managerial independences are respectively represented by A_i and G_i. But, the concept of action is not used and not really described to be used in an example. An action is presented as being just an "object" without a clear definition on how to use it. The set of connections C_i enables to model interactions. Nevertheless, connections are only links with another component system and are never used, for example, in metric definitions. Moreover, it is a totally generic model and it enables to calculate metrics as autonomy, belonging, and diversity of a component system. It is useful to know the level of diversity of a SoS because the most a SoS shows diversity, the most it can cope with the dynamics of the environment. Then, a SoS model is presented but there is no environment model and this is a strong limitation because a SoS evolves in a dynamic environment. Concerning global expressiveness, the model is poor because the action is represented without the notions of preconditions and/or effects of an action. Then, there is no possibility to model action dependence between component systems.
Concerning the Agent-Based Wave model, it enables to model component system through its operational independence (through capability) but fails with managerial independence because goals are not defined for component systems. Nevertheless, the model may use metrics to evaluate SoS like the performance to reach (through P_i). Concerning global expressiveness, each component system is reduced to only one capability, which seems far from the reality of the SoS where each component system has several capabilities. This model does not enable to model interactions between component systems. Even if an environment is defined, it is not possible to express its dynamics. Indeed, there is no possibility of expressing dependencies between the actions of the component systems.
Finally, Table 1.4 sums the whole criteria for both models. The legend is the following: the evaluation criterion for the model is totally filled (++), almost partially filled (+), almost partially absent (-), totally absent (- -).

The two SoS models presented here have a lack concerning their realism and overall fail concerning our evaluation criteria. Especially the set theory model seems hard to use to study real cases of SoS because of a lack of expressiveness. Furthermore, the interactions between component systems in both models are not used and the notion of environment is not mentioned (or not dynamic). This analysis leads us to propose a new generic SoS model to fill these lacks.

	Generic set theory model	Based Waved Model
Component system heterogeneity	++	+
Managerial independence	++	-
Operational independence	++	++
Geographical distribution	- -	- -
Interactions between component systems	-	- -
SoS model	++	++
Dynamic environment model	- -	-
Global expressiveness	-	-
Metric definitions	+	+

Table 1.4 – Evaluation of existing generic SoS model

2 Existing Architecting Heuristics and Simulation Tools for SoS

In the previous chapter, we have presented the notion of SoS through existing definitions and characteristics with a focus on two generic SoS models. This chapter introduces the notion of architecting heuristics for SoS and how to simulate them. In this way, this chapter focuses on:

- ▷ enabling the reader to understand what the notion of SoS architecting is;
- ▷ presenting chosen criteria to evaluate existing SoS architecting heuristics;
- ▷ studying the existing SoS architecting heuristics through evaluation criteria we define;
- ▷ presenting the notion of SoS Architecting Heuristic Simulation (SAHS).

In this way, the first part focuses on the definition of SoS architecting. Then, a selection of SoS architecting evaluation criteria are presented and justified. Then, two architecting heuristics are presented and discussed, chosen because they rely on generic SoS models.

Concerning SAHS, after a general introduction to simulation in computer science, the main challenges about SoS simulation are discussed. Then, the chapter focuses on SAHS and evaluation criteria will be extracted from this analysis. At a first time, these criteria are classified in two classes: according to their usefulness for SoS simulation and for SAHS. Then, these chosen SAHS criteria are used to evaluate existing tools for SAHS. Finally, a discussion highlights the scientific limitations that will be handled in the contribution of this document.

2.1 Architecture & Architecting of SoS

In the area of SoSE (SoS Engineering), SoS architecture refers to the set of component systems of the SoS and their interactions. [Wang et al., 2014] defines SoS architecture as *"an arrangement for the set of constituent systems, rules and behaviors that govern an individual system's functions. Architecture also describes how these systems' capabilities contribute to a larger goal."*

As previously said, a SoS evolves in a dynamic environment and its goals can change during time whose cause the SoS to change the set of the component systems and the inter-

actions between them. For example, a satellite (being a component system) used to transmit a signal between a radar station and a combat unit (both considered as component systems) can be periodically out of range because of its displacement. Thus, to overcome this difficulty, the radar and the combat unit have to find a new component system to relay the signal, implying a modification of the architecture. To sum up, a SoS cannot have a static architecture during time, it has to adapt its architecture during time.

The process to propose dynamically SoS architectures during time is called in this document SoS architecting. In [Azani, 2008], *"SoS architecting will develop executable architectures with dynamic representation and reconfiguration capability enabling continuing SoS performance and effectiveness analysis under various conditions and procedures. Static architecture representations and products are not effective tools for analysis of SoS evolving structure and emerging behavior."* To drive the development of SoS Architecting, [Azani, 2008] propose to use six core principles:

- Open interface principle - Open systems have permeable boundaries that allow them to exchange mass, energy, and information with other systems;
- Synergism principle – The notion that designates that the co-operative interaction between component systems has a greater effect in their combined efforts than the sum of their individual parts. Essentially, this is what gives rise to emergence;
- Self-government principle - This implies that the SoS maintains and develops its internal order without interference from external sources. This could be through cybernetic control, homeostasis, or self-organization;
- Emergence principle - In this case, this refers to the occurrence of novel and coherent structures, patterns, and properties during the self-organization of the SoS;
- Conservation principle – This principle states that energy and mass (material) are conserved within the SoS;
- Reconfiguration principle – This refers to the SoS reconfiguring and adapting itself to sustain against changes in its environment.

Finally, SoS Architecting heuristic consists in finding how component systems can work together and change their interactions to fulfill the SoS goals efficiently and effectively. Due to the dynamics of the environment and the large set of solutions area, finding the optimal architecture during time for a SoS is nearly impossible. In other words, the time of resolution can be higher than the time of a change in the environment. Thus, the use of heuristics in SoS architecting has been proposed in literature in order to find a satisfactory architecture for a given state of the environment. To find such an architecture, designers can use the managerial and operational independence of component systems to propose decision strategy (like collaboration) at the component system level. For directed and acknowledged SoS, designers use central management to coordinate and/or use directly component systems resources. For example, central management can decide to add or remove component systems during functioning. Based on and using classical AI tools, we present two existing

architecting heuristics for SoS and their limitations; they have been chosen because they are the most developed and documented.

2.2 Criteria for the Characterization of Architecting Heuristics

We define six evaluation criteria to evaluate the overall quality of SoS architecting heuristics. They are inspired from literature and from the six core principles of [Azani, 2008]. First, a SoS architecting heuristic has to be tested and simulated through different scenarios based on a model that can be easily **computed** and should be as much as **generic** as possible to be used in various domains as possible. As we study systems evolving in a dynamic environment, the **dynamicity** of the architecting heuristic is important. From the open interface principle, a SoS has to have an open architecture (i.e., component systems can be added or removed at runtime), then the **openness** of the heuristic is important. As a huge amount of component systems can be involved, the **computational cost** of the heuristic is important. From the self-government principle, SoS may or not be modeled with a central control (i.e., a SoS can be directed, acknowledged, collaborative or virtual). Then, the heuristic should have the ability to work in a **decentralized** manner. Finally, from the synergism principle, the use of **cooperation** between component systems in an efficient SoS architecting heuristic seems important. These criteria cover characteristics of SoS architecting and will be used to evaluate properly existing heuristics. We describe each one of the criteria we have defined in the following paragraph:

Computationality: The computationality of an architecting heuristic is the ability to be implemented on a computer. If a heuristic describes in a too much literal way its functioning, it becomes a hard task to implement it on a computer and may lead to mistakes or misunderstanding during implementations.

Genericity: The genericity of an architecting heuristic is its ability to be used in various domains. An architecting heuristic is not generic if it uses domain specific information to work. For example, if a heuristic architecting can be used only for the military domain, this one is not considered as a generic heuristic.

Dynamicity: The dynamicity of an architecting heuristic is its ability to enable the SoS to adapt to the changes occurring in the environment and the SoS goals. Thus, the architecting heuristic should take into account parameters and objects related to the environment.

Openness: The openness of an architecting heuristic is its ability to add and remove component systems within the SoS during its functioning.

Computational cost: Computational cost is a metric representing the amount of computational resources the architecting heuristic needs to work.

Decentralization: It is the ability to propose architecture without the need of a central management to work. Managing is not equal to have a coercive power on component systems. For example, in acknowledged SoS, a central management exists but has no coercive power. Finally, a decentralized architecting heuristic enables to architecture collaborative and virtual SoS because these types of SoS have no central management.

Cooperation: It is the ability to design a SoS where component systems can use cooperation with each other. This ability is important because it drives the emergent behavior of the SoS [Gonçalves et al., 2014] and respects the synergism principle.

The following sections introduce two existing SoS architecting heuristics.

2.3 SoS Architecting Heuristic Based on a Collaborative Approach [Caffall and Michael, 2009]

In some types of SoS (like acknowledged and collaborative ones), component systems are free to participate (or not) to the SoS goals. As they are managerially independent, they have their own goals and it can happen that they are in part contradictory or incompatible with the SoS ones or with other component systems. In this case, game theory explains that a rational agent (here a component system) will evaluate the advantages and the constraints of the collaboration (i.e., the participation to the SoS goals) and if the choice of collaboration leads to few advantages or too much constraints, the component can decide to quit the SoS. Thus, authors propose in [Caffall and Michael, 2009] an architecting heuristic where component systems:

1. act collaboratively with each other to avoid counter-productive actions between each other;
2. are able to negotiate the gain and the constraints of collaboration with the SoS.

To reach (1), this architecting heuristic uses satisficing games theory (described in section 2.3.1) and to reach (2), mechanism design and Multi-Criteria Decision Analysis (MCDA) are used [Caffall and Michael, 2009]. The following sections detail these notions and their articulation around this architecting heuristic.

2.3.1 Satisficing Games

Satisficing games are used in this heuristic to enable collaboration between component systems. The notion of **satisficing** introduced by [Simon, 1956] comes from the combination of **satisfy** and **suffice**. Based on game theory, satisficing games are multi-agents games created to take decision with multiple agents dealing with rationality and enabling cooperation [Stirling, 2005]. As authors explain, in game theory, rationality with only one agent consists in choosing the preferred action for itself. These preferences are often ordered thanks to a utility, creating a utility function. But, in a multi-agent context, doing the best for an agent can be harmful to other agents because having concurrent objectives or concurrent access

to resources. Then, choosing the action that maximizes the utility function of an agent is not necessarily the best solution for the group. In other words, agents influence each other because the action of an agent can (or not) helps another agent to fulfill its goals.
To overcome this shortcoming, authors propose a new notion of rationality where preference (of an action) is conditioned by preferences of others. It enables to take into account both the preferences of an agent (its own rationality in terms of classic game theory) and the preferences of other agents. The following sections introduce the formalism of satisficing game theory and show its limitations concerning our evaluation criteria.

2.3.2 Interdependence Function and Satisficing Solution Set

Each agent computes its preferences taking into account the preferences of other agents. The preferences of an agent are constructed with two 'personas' or 'roles' that evaluates its own options (or actions): the first persona, called **selectability**, is based on the effectiveness of an option (i.e., how the option is close to the agent goal). The second called **rejectability** is based on resources consumption or cost, called inefficiency of an option. These two contradictory roles are mathematically represented by classical utilities having the same mathematical structure as a probability mass function [Hill et al., 2005].
Thanks to these functions, interdependencies between agents are represented through an **interdependence function** called **joint selectability and rejectability** mass function. This function is computed in the same manner as a **joint probability** function. The only difference is that conditioning has not the same semantic than the classical probability conditioning. In satisficing semantic, conditioning represents **option dependence** (in term of rejectability and selectability) for an agent with regard to other agents' options. The details of the interdependence function construction are explained hereafter.

For a $n-$multi-agent system, the interdependence function is mathematically a joint mass function and expressed as:

$$p_{int} = p_{S_1 \ldots S_n, R_1 \ldots R_n}(u_i, ..., u_n, v_i, ..., v_n)$$

with $\forall i \in [\![1, n]\!]$:

- $u_i \in U_i$, an available option (or action) of agent a_i;
- $v_i \in U_i$, an available option (or action) of agent a_i;
- U_i the set of all the options of a_i.

Interdependencies between agents are graphically represented with a **praxeic network**. In a praxeic network, each node represents how the agents' personas will influence others agents personas. Figure 2.1 shows a simple example of a praxeic network for a SoS composed of 3 agents. In this network, links show interdependence between agents' personas S_i (selectability of agent i) and R_i (rejectability of agent i). The figure 2.1 has to be read like this:

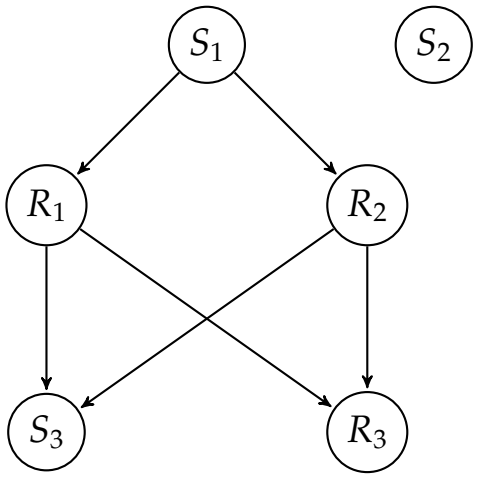


Figure 2.1 – Praxeic network for a SoS with three agents a_1, a_2, a_3

- Selectability S_1 of agent a_1 influences rejectability R_1 of agent a_1 and rejectability R_2 of agent a_2;
- Rejectability R_2 of agent a_2 influences selectability S_3 and rejectability R_3 of agent a_3;
- Rejectability R_1 of agent a_1 influences selectability S_3 and rejectability R_3 of agent a_3.
- Selectability S_2 has no influence on a_1 and a_3.

Then, the interdependence function can be expressed as following:

$$p_{S_1S_2S_3R_1R_2R_3}(u_1, u_2, u_3, v_1, v_2, v_3) = \\ p_{S_3|R_1R_2}(u_3, v_1, v_2) \cdot p_{R_3|R_1R_2}(v_3, v_1, v_2) \cdot p_{R_1|S_1}(v_1, u_1) \cdot p_{R_2|S_1}(v_2, u_1) \cdot p_{S_1}(u_1) \cdot p_{S_2}(u_2)$$

Where:

- u_1, u_2, u_3 are respectively the options chosen by a_1, a_2, a_3;
- v_1, v_2, v_3 are respectively the options rejected by a_1, a_2, a_3;
- $p_{S_3|R_1R_2}(u_3, v_1, v_2)$, for example, is the mass function giving the selectablity value of option u_3 chosen by a_3 conditioned by the choice of a_1 to reject option v_1 and the choice of a_2 to reject option v_2.

In the general case, the selectability and rejectability functions are marginal functions (in terms of probability function) of the interdependence function:

$$p_{S_1...S_n}(u_1, ..., u_n) = \sum_{v_1 \in U_1} ... \sum_{v_n \in U_n} p_{S_1...S_nR_1...R_n}(u_1, ..., u_n, v_1, ..., v_n)$$

and

$$p_{R_1...R_n}(v_1, ..., v_n) = \sum_{u_1 \in U_1} ... \sum_{u_n \in U_n} p_{S_1...S_nR_1...R_n}(u_1, ..., u_n, v_1, ..., v_n)$$

Finally, the **jointly satisficing solutions at caution level q** is defined as:

$$S_q = \{u \in U : p_S(u) \geq q p_R(u)\}$$

where:

- $q \in \mathbb{N}+$ represents the caution level;
- $U = \{u_1, ..., u_n, v_1, ...v_n\} \in U_1 \times ... \times U_n \times V_1 \times ... \times V_n$ is the vector of the n chosen options in terms of selectability and rejectability.

The set S_q contains all option vectors u such as the value of the joint selectability p_S is at least as great as q times the joint rejectability p_R. The use of a caution level q enables to modulate the set of satisficing solutions. Indeed, a caution level equal to q gives a set of satisficing solutions that are q times better in terms of **selectability** compared to the **rejectability**. Then, the higher the caution level is, the more satisficing the solutions are. This approach is used to model UAV formation flight with obstacle avoidance [Stirling and Frost, 2005] and aircraft collision avoidance [Hill et al., 2005].

2.3.3 Mechanism Design as a Basis for SoS Collaborative Formation

As explained before, in some types of SoS (like acknowledge or virtual ones), component systems do not have the obligation to collaborate with the SoS. This is a classic statement of game theory: each component system (seen as a player) is considered as rational when it chooses action with the highest gain (also called utility) for itself. But at the same time, it is also known that the optimization (maximization) of component system utility does not lead to the optimization of the global system (here the SoS) [Caffall and Michael, 2009]. But, it exists a game theory concept enabling the construction of a game where the highest collective benefit may be reached and where **at the same time** each player chooses an action that maximizes its own utility. This concept is called **mechanism design**.
More precisely, mechanism design is a concept of game theory created to solve a particular kind of Bayesian game (a game under private information between players). Sometimes, in a context of bargaining between players, a player called the "principal" would like to adapt his behavior according to private information owned by other players called "subordinates" that have the ability to lie about it. The "principal" has the whole bargaining ability and the idea is to find rules (more formally called **mechanism**) that lead other players to reveal their private information without forcing them. In this kind of game, the goal of the principal is generally to maximize the common interest. In this asymmetric game, the principal (which has its own objectives) defines the rules of the game in a way that agents will act as it wants. We propose the purchase of a used car as the informal example. To be satisfied, the buyer would like to have the good at the lower price as possible depending of the real state of the car (good, used, very used...) which is a **private** information of the salesman. Indeed, the buyer cannot have this information by asking the salesman, as he wants to get the higher price for his good and has the ability to distort the reality. Nevertheless, the best solution for the group is that the sell occurs. As the buyer can decide of some rules of the game, he will influence the salesman to reveal his private information. He can, for example, ask to subscribe to an insurance for the car. The response of the salesman will give

some information about the real state of the good. This kind of games can be solved with **mechanism design**.
Mechanism design is well formalized in some papers like [Jackson, 2003] and [Myerson, 1983] and the main important result is the **revelation principle** stating that *"To every Bayesian Nash equilibrium there is a corresponding Bayesian game with the same equilibrium outcome but in which players truthfully report type."* In other words, it is always possible for the principal of a game to find a **mechanism** (also called the **social function**) where players trustfully give their private information and at the same time, respect their own incentives.

To sum up, the mechanism design in Collaborative SoS Formation ([Caffall and Michael, 2009]) enables the architecting heuristic to design a **social function** (by finding rules as presented above) where SoS goals (through the role of principal) and where component systems (the subordinates) will participate for the good of the SoS and their own objectives (represented by their utilities) at the same time even if they own private information and *[...] may be autonomous and of self-interest* [Caffall and Michael, 2009].

This social function is based on the social utility function (in terms of satisficing games) as described in [Stirling and Frost, 2005]. As we previously presented, the social utility gives the satisficing solutions for a group of agents (here component systems) by taking into account interdependencies between them (i.e., component systems act collaboratively with each other). Moreover, each component system owns a private utility (also called "expected utility") function that is represented by the individual selectability and rejectability marginal functions presented above. To take into account the goals of the SoS, this one uses mechanism design to re-design the social function. Finally, as component systems can choose to leave the SoS, they are able to negotiate their **expected utility** (utility function of a component system) with other component systems and the SoS. This mechanism enables the SoS to keep component systems if they are, for example, really valuable. In [Archibald et al., 2006a], authors argue that it can happen if the satisficing set is empty. In this case, *there is a number of negotiation protocols that could be implemented, with one of the simpler being a round-robin procedure of decrementing the negotiation indices of the participants*. In Collaborative SoS Formation, the negotiation protocol is the Analytic Hierarchy Process (AHP) presented in the next section.

2.3.4 Multi-Criteria Decision Analysis and AHP Solving

MCDA (Multi-Criteria Decision Analysis) is a method enabling to take a decision between some alternatives by rating them with criteria to fulfill a given goal. To find solutions (i.e., to find the best alternative or to classify them), authors propose in [Baldwin and Sauser, 2009] to use AHP. Developed by Saaty in [Saaty, 1980] and as shown in figure 2.2, a hierarchy is created to evaluate choices by a set of criteria. These criteria are chosen by the decision-maker according to what he wants to evaluate. As an example, for choosing a car (figure 2.2), criteria can be the **Type**, the **Consumption**, the **Appearance** and the **Engine**. Each alternative is pair-compared with others according to a particular criterion. The pair-comparison

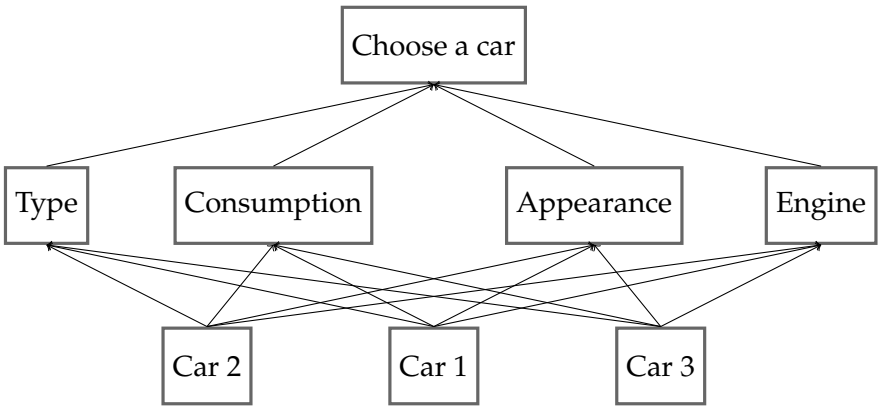


Figure 2.2 – AHP structure

is scored with an intensity of importance defined by the modeler of the AHP. As an example, Car 2 and Car 1 are compared with respect to **Consumption**: If Car 1 is two times better, then score of Car 1 is, for example, 2 and score of Car 2 is 1. After this process, the alternatives are classified according to a score computed as the average of all the criteria score attached to them. In Collaborative SoS Formation, the defined criteria are the Autonomy, Belonging, Connectivity, Diversity and Emergence. Unfortunately, authors do not explain in details how these negotiations between SoS and component systems are made.
Finally, Figure 2.3 sums up the global process of the method with three component systems: component systems X,Y and Z began the process (state $S0$ on the figure 2.3) by giving their Local Utility to the SoS to compute the Social Utility Function thanks to satisficing games. Then, the SoS will negotiate its preferences (thanks to its own goals and mechanism design) with X,Y and Z. Then, X, Y and Z will negotiate and change their expected utility (thanks to MCDA process). Once the negotiation is over (state $S1$ on the figure 2.3), X, Y and Z give their new Local Utility to the SoS to compute a new Social Utility Function and so on.

2.3.5 Discussion

This Collaborative SoS formation heuristic uses a combination of different AI technologies. We will evaluate this heuristic through the evaluation criteria defined in section 2.2.
Concerning computationality, the method uses satisficing games which is extremely formal and can be easily instantiated on a computer. But the paper lacks of details concerning the implementation of the whole heuristic. The articulation of mechanism design and MCDA (through AHP) is not well explained.
Concerning genericity, this method uses a model based on set theory model presented in chapter 1. This one is generic to model all kind of problems but has a lack in term of environment model. Moreover, the resolution of a satisficing game (i.e. find the satisficing set in a given environment) is based on Bayesian network resolution technique [Stirling and Frost, 2005] dealing only with **acyclic** graph. Then, if a dependence cycle exists, no solution can be found.
Concerning dynamicity, the method enables a re-computation of a new solution if changes appear in the environment. But the computation is heavy because it is based on satisfic-

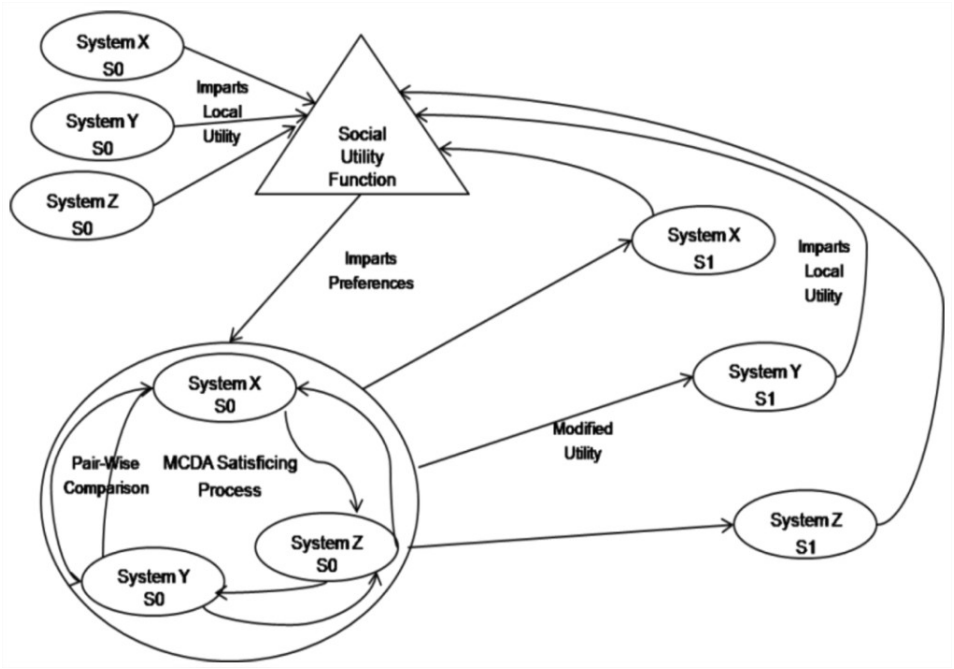


Figure 2.3 – Social utility function negotiation

ing games. Moreover, modelers must know a priori all direct interdependencies between agents, which is not always the case.

Concerning openness, there is no visible limitations about adding or removing component systems during functioning of the SoS. But once again, the use of satisficing leads to a certain complexity because of the use of the praxeic network: adding or removing an agent leads to a change, so a re-computation of the interdependence function that may be costly in time in a real time environment. Then, the openness of the method is costly.

Concerning decentralization, the use of mechanism design implies the existence of the "Principal" role. It implies the existence of a central management that disables to architect virtual and collaborative SoS.

Concerning computational cost, the method uses satisficing games that are costly because of the computation of the social and expected utility functions. As authors explain in [Archibald et al., 2006b]: *"Computational complexity arises because of the calculation of the marginal of the interdependence function. This complexity can be mitigated [...] using, for example, Pearl's Belief Propagation Algorithm [...]. Even so, it is well known that even these approaches are NP hard, and the computational burden for a tightly interconnected, high dimensional multi-agent system may become intractable."* Thus, the computational cost of this method may be high.

Concerning cooperation between component systems, the use of mechanism design enables to use the game theory formalism. But in order to use it, a global goal has to be well defined before the instantiation of the system.

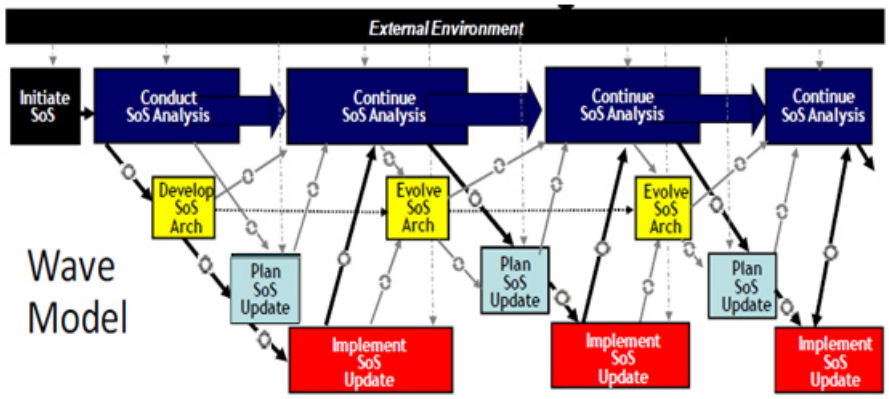


Figure 2.4 – Wave model

2.4 SoS Architecting Heuristic Based on Agent-Based Wave Model

SoS architecting heuristic based on an agent-based wave model developed in [Acheson et al., 2012] and [Edward Pape II, 2016] aims at proposing a generic method to find a satisfying architecture of a SoS constrained by an external environment (SoS total cost, development time, threats and so on). To reach this objective, the method is composed of several steps that are repeated through **waves** until a satisfying architecture is found. The different steps of this heuristic are explained in the following sections.

2.4.1 Step 0: SoS Formal Description through Interviews

The inputs of the method are the **global needs** of the stakeholders of the SoS and the stakeholders of the component systems that are collected and summed up thanks to interviews documents. The main objectives of these documents are :

- ▷ to establish a common lexicography between SoS stakeholders (i.e., the owners of the SoS) and SoS engineers;
- ▷ to extract the global goals (or objectives) of the SoS;
- ▷ to express these goals in terms of desired capabilities;
- ▷ to compose these desired capabilities with capabilities of component systems;
- ▷ to identify the component systems;
- ▷ to formally describe the SoS and component systems with the agent-based wave model (see section 1.4.3);
- ▷ to define Key Performance Attributes (KPAs) to evaluate SoS architecture.

The term **capability** is used in the agent-based wave model to describe a kind of activity made by a component system. For instance, considering an UAV (Unmanned Aerial Vehicle) as a component system, a capability may be the detection of threats on the ground.

Figure 2.5 – SoS architecture chromosome example

2.4.2 Step 1: Initiate SoS

The formal description made in step 0 with the agent-based wave model enables to define the desired capabilities of the SoS. As a reminder, a SoS is represented by:

$$SoS = \{C, W, P\}$$

With:

- C, a set of desired capabilities c_i;
- W, a set of weight concerning the set of desired capabilities C;
- P, a set of desired performances on capabilities in the set C.

Once the formal description is finished (step 0), a first SoS architecture is proposed by choosing a set of component systems that fulfill the desired capabilities. Indeed, each component system is described with a capability and performance on it. Then, this architecture is modeled through a chromosome (one example is presented in figure 2.5) composed of two parts. The first part represents the component systems that will participate to the SoS (1 for participating, else 0) and the second one represents the existing connections between component systems (1 for linked, 0 for not). This chromosome is used as an input of the step 2 presented below.

2.4.3 Step 2: SoS Architecture Evaluation through KPAs and FIS

A fitness function based on KPAs (defined in step 0) is used to evaluate the architecture. Generally, the KPAs are the following: Performance, Robustness, Funding (or Affordability) and Flexibility. The formulation used to compute these attributes depends on the stakeholders. Generally, the overall SoS performance on an attribute is a linear combination of component systems' performance on this attribute. Then, all of these KPAs are used as inputs of a Fuzzy Inference System (FIS) that is a fuzzy logic tool used to compute a **fuzzy variable output** with a set of **fuzzy variable inputs** and a set of **fuzzy rules**. A fuzzy rule is an association of fuzzy output values with a set of fuzzy input values. Table 2.1 shows an example of fuzzy rules based on three fuzzy variable inputs and one fuzzy variable output. As an example of rule, when the SoS has a **high** performance gap (between the desired performances and the current performances given by the component systems), the third rule (corresponding to the third line of the table 2.1) gives a **little increase** to the funding enabling the SoS to have more funds to negotiate with component systems. The main advantage of a FIS is to compute variables that are not expressed in the same unity. As Funding and Performance

Inputs			Outputs
Performance Gap	Weight	Funding Gap	Funding
none	none	none	decrease little
low	none	none	do nothing
high	none	none	increase little
none	low	none	do nothing
none	high	none	increase little

Table 2.1 – An example of set of fuzzy rules

may be expressed respectively in currency and up-time duration, they are not directly comparable. Then, the FIS enables to go through this kind of problem by translating (or fuzzing) attributes from their original value to fuzzy value. A FIS used to fuzzy the KPAs is described in [Dagli and Acheson, 2012]. Basically, the values are fuzzyfied and defuzzyfied with an algorithm called Enhanced-Karnik-Mendel algorithm [Dagli and Acheson, 2012].

2.4.4 Step 3: Propositions of Alternative SoS Architectures with Genetic Algorithm

For the genetic algorithm part, the SoS architecture modeled by a chromosome (presented in figure 2.5) is used as an input. The chromosome has two main parts: the first one represents the component systems involved in the SoS, the second one represents the set of links between the component systems. Then, the genetic algorithm computes new SoS chromosomes that are close from the first one. Finally, the fitness function (presented in the section 2.4.3) rates the new chromosomes and are presented as alternative architectures that can be used as the new SoS architecture. Then, an architecture is chosen (for example, the best rated by the fitness function) and a last step of negotiation between SoS and component systems is executed. The next section explains this negotiation step.

2.4.5 Step 4: Component System Model and Negotiation

When the architecture is chosen, SoS sends negotiation requests to component systems involved in the new architecture. As they are managerially independent, component systems can choose to participate to the SoS or not. A component system decides to negotiate with the SoS with the help of a Fuzzy Inference Engine (FIE) [Paredis et al., 2013] that has the following inputs:

- Willingness to cooperate, which is related to the degree of selfishness of the component system;
- Ability (in terms of resources) to cooperate depending on component system's resources that will allow it to be part of the SoS.

These variables represent the propensity for a component system to collaborate with the SoS. Because it is domain-dependent, FIE has to be defined for all component systems. This

negotiation enables to simulate the managerial independence of component systems. At the end of this step, stakeholders analyze the new architectures and choose the preferred one based on their expertise. Finally, the whole process may be repeated again to find better SoS architectures.

2.4.6 Discussion

This generic SoS architecting heuristic, divided on several steps, is based on interviews with stakeholders, fuzzy logic and a genetic algorithm. This method uses the concept of SoS as an autonomous entity that negotiates with component systems: it is not a decentralized heuristic. All communications are going through the SoS and component systems cannot directly communicate between them. Moreover, this SoS decides to send collaboration request to component systems, which cannot choose to contact the SoS or other component systems. This method uses a model that is generic enough to model all kind of problems.
Concerning dynamicity, the SoS agent makes its goals evolving according to the changing environment. For example, the process can update the SoS if the global funds for the SoS are changed (through, for example, the KPA concerning the affordability).
Concerning openness, the SoS is able to negotiate with other component systems to join the SoS. But component systems cannot quit the SoS by themselves.
Concerning computationality, the fuzzy assessor needs a set of fuzzy inference rules. In the paper, an example for 4 variable inputs that can take 4 values leads to a set of (4^4) 256 rules that have to be defined by the SoS designers. Moreover, the fuzzyfication of all the variables (funding, performance, robustness and so on) are made and chosen in an informal manner by the designers and stakeholders. Then, there is no insurance that the fuzzy methodology used will product the best collaborative architecture. Moreover, if they are not well defined, they may introduce bias : *"Some KPAs of the SoS remain ambiguous even after extensive discussions among the stakeholders"* [Edward Pape II, 2016].
Concerning computational cost, no study is available on the different papers found for this method. Concerning cooperation, this method enables collaboration between SoS and component systems but does not deal with direct collaboration between components.

2.5 Simulation of Systems

For [Banks et al., 2005], *"a simulation is the imitation of the operation of a real-world process or system over time"*. A simulation of a system can have several goals:

- ▷ to predict and/or understand a real phenomenon that, at first glance, is difficult to predict and/or understand;
- ▷ to evaluate the effects of potential changes in a system structure or inputs;
- ▷ to test hypothesis on the functioning of a phenomena or a system;
- ▷ to serve as a pedagogical device for training and learning;

- to deal with about internal interactions of complex system (factory, water fabrication plant, service organization, etc.) [Banks et al., 2005].

The simulation of a system requires the creation of a simulation model: it takes the form of assumptions concerning the functioning of the system. These assumptions are expressed in terms of mathematical, logical and symbolic relationships between the entities of interests of the system [Banks et al., 2005]. The development of such a simulation model is not an easy task because of several problematics. As an example, creating a simulation model is a simplification of the real system in order to be able to run numerically an experiment. The choice of the right level of simplification may be difficult: a too simple simulation model can lead to an irrelevant simulation and at the contrary, a too complex simulation may lead to results that are difficult to understand. Another problematic is the choice of the type of simulation model: it can be static or dynamic, deterministic or stochastic and discrete or continuous. A static simulation model, also called a Monte Carlo simulation, represents a system at a particular point in time. On the other hand, a dynamic simulation model represents a system as it changes over time [Banks et al., 2005]. Deterministic means that the simulation model does not contain random variables in it. Discrete means that variables may change only at discrete values of times. On the other hand, in a continuous simulation model, variables change continuously. The choice between a continuous or discrete simulation depends on the type of studied phenomena. To drive the choice and the creation of a simulation model, the objectives of the simulation have to be defined:

- What are the characteristics or properties of the real system to be studied?
- What is the right level of abstraction (or simplification) needed?
- What are the relevant inputs or outputs?

Finally, this variety of simulation choices may lead to a problematic of coherence between simulations: for a given problem, using different simulation models may lead to different simulations, and consequently conclusions.

To help simulation designers, the research on simulation of systems focuses on the development of tools, methodology, standards and so on. This area of research involves several application domains (transport, defense, logistics and so on). In classical System Engineering (SE), Modeling and Simulation (M&S) methodologies give a scientific framework to guide designers to create simulation models. But as explained in the introduction of this document, the oncoming of SoS Engineering (SoSE) leads to "re-open" classical SE research areas because systems under studies have changed: the characteristics of a SoS and its component systems (heterogeneity, openness, independence of elements and so on) explain the limits for applying M&S to SoS simulation. This point is developed in the following section.

2.6 Challenges and Issues of "SoS Simulation"

As classical simulation, **SoS simulation** concentrates on reproducing SoS as close as possible to the reality [Zeigler et al., 2000]. But, as highlighted in the previous section, one main

issue is to find models that lead to interesting results close to the reality. For classical system simulations, M&S proposes methodologies to realize these simulation models, but they do not fit for SoS simulations because of the huge complexity of SoS. The complexity of its component systems and its dynamics conduct to non-linearity effects between them. Reproducing this non-linearity is important because it can be the source of desired emergence in the SoS. But, simplifications may entail the loss of this non-linearity [Henshaw et al., 2013]. That is why getting the correct **level of abstraction** for a component system is important: in one hand, a too simple model and simulation can be too simple and unrealistic and in the other hand, a too complex model and simulation may lead to a huge computational cost or simply becomes incomprehensible. Additional issues due to the SoS characteristics have been highlighted in [Henshaw et al., 2013].

Being composed of heterogeneous and complex systems implies that heterogeneous simulation models (of different types) work together. Moreover, modeling and simulating these components require **expertise** on specific domains and, for simulation tools, the need to model or to use existing models that can be heterogeneous in terms of types (static or dynamic, discrete or continuous and so on). Then the ability to simulate **heterogeneous** models is an important criterion.

Moreover, SoS is often composed of legacy systems having their own models and/or simulation tools that can be incompatible because of their different types. For example, plunging a time discrete simulation model with a continuous simulation model is generally difficult (but still possible). It leads to a choice of a simulation paradigm that enables **interoperability**.

Furthermore, as several SoSs integrate the **"human in the loop"**, it is important to be able to model human behavior, which is not an easy task. Finally, as a huge number of component systems and interactions can be involved, simulation has to be efficient to avoid computational cost. Thus, the last criterion is the **simulation performance**.

2.7 Challenges and Issues of SAHS (SoS Architecting Heuristic Simulation)

SAHS and classical SoS Simulation presented before have different goals. The main idea of SAHS is to study at a high level of abstraction how management of the SoS and **social behavior** of the component systems can be used to propose good SoS architecting heuristics. As explained in chapter 2, SoS architecting heuristics focus on finding the right behavior and management of component systems to propose a **dynamic architecture**, in order to cope with changes of the SoS and its environment. SAHS must take into account the evolution of the SoS architecture to cope with the dynamics of the environment. Moreover as we explained in chapter 1, the dynamic architecture leads to the ability of a SoS to produce a large-scale **emergent behavior** and a social behavior the ability to **self-organize**. These criteria are also called meta-requirements in SoS literature [Ceccarelli et al., 2015]. Furthermore, a SoS architecting heuristic is more valuable if it is generic enough to be applied to a large variety of domains. This genericity leads to focus on component system model where

	SoS Simulation	SAHS
Component system level of abstraction	-	+
Interoperability	Technical	Pragmatic
Heterogeneous simulation models	+	-
Human models	+	-
Dynamic architecture	-	++
Dynamic Component social behavior	-	++
Simulation performance	++	+

Table 2.2 – Criteria for SoS Simulation and SAHS discrimination

the **level of abstraction** is less important than in classic SoS simulations.

Nevertheless, there are some common points to both simulation approaches: interoperability is an important issue concerning classical and heuristic simulations but not studied at the same level. In [Mittal et al., 2014], authors argue that in classical SoS simulations, efforts are put on **technical interoperability** (i.e., focused on data exchange between heterogeneous simulations). For example, the interoperability of an aircraft model developed in JAVA and a control tower in C++ consists in making them able to exchange data. In SAHS, interoperability is studied at a **pragmatic** level (i.e., how the information in the data are used). As a huge number of component systems and interactions can be involved in SAHS, the simulation performance is also an important criterion.
This analysis shows that the evaluation criteria are not the same to evaluate classical SoS simulation and SAHS. Thus, we choose the evaluation criteria that seem the most important for SAHS: the ability to deals with **pragmatic interoperability**, **heterogeneous models**, a **dynamic architecture**, a **dynamic component social behavior** and **simulation performances**. Then, the following sections focus on finding simulation paradigm that fulfills these criteria. Table 2.2 presents the whole criteria for both SoS simulation and SAHS. The legend is the following: (++) means that the evaluation criterion is mandatory, (+) means that is important but not mandatory, (-) means that the criteria can be used but is not important (- -) means that the criterion is not important.

2.8 Evaluation of Existing Paradigms for SoS Simulation

In literature, the two main paradigms for SoS simulation are Agent-Based Simulation and Discrete Event System Specification (DEVS). Both are not focused on the same needs concerning SoS simulations. The next sections explain in more details these differences and analyze their adequation for SAHS through the prism of the criteria addressed and summed up in table 2.2.

2.8.1 DEVS and DEVS Variants

Introduced by [Zeigler et al., 2000], DEVS is a formalism used to model and simulate system called DEDS (Discrete Event Dynamic System). A DEDS is defined in [Banks et al., 2005] as a *"system where state space is discrete and where state can only change as a result of asynchronously occurring instantaneous events over time"*. So basically, DEVS models systems in which the state variable changes only when events occur during time. This separation and the attached formalism (presented below) of DEVS enable to bring interesting mathematical proofs about correctness of simulation algorithms. The aim is to have a formal verified simulator. This is the main reason of the great interests of DEVS in simulation area. In [Mittal and Luis Risco Matin, 2013], authors present DEVS and its extensions (also called variants) for simulating SoS and argue that they suit for SoS simulations. The following paragraphs present the basis of DEVS and how its variants can be used to suit SoS simulations. In DEVS, the simplest system is called an atomic-DEVS.

An atomic-DEVS is defined by the 7-tuple $M = \{X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta\}$ where:

- X is the set of input events;
- Y is the set of output events;
- S is the set of sequential states;
- $\delta_{int} : S \to S$ is the internal transition function which defines how a state of the system changes internally;
- $\delta_{ext} : Q \times X$ is the external transition function which defines how an input changes the state of a system;
- $Q = \{(s, e) : s \in S, e \in [0, ta(s)]\}$ is the total state set, and e is the elapsed time since the last transition;
- $\lambda : S \to Y$ is the output function used to generate output events;
- $ta : S \to \mathcal{T}^{\infty}$ is the time advance function which is used to determine the lifespan of a state.

At the beginning of the simulation, M is in state $s_0 \in S$. Then, there is only two ways for M to change its current state:

1. if an event occurs in one of its inputs in X;
2. if the elapsed time t_e reaches the lifespan of s_0 (given by $t_a(s_0)$)

Then, δ_{ext} and δ_{int} are respectively used in case (1) and (2) to change the state of M. Finally, λ is used to generate output events due to the new state of M.

The second main model enabling interactions between atomic-DEVS is the coupled model. It is defined as an 8-tuple: $N = \{X, Y, D, \{M_i\}, C_{xx}, C_{xy}, C_{yy}, Signal\}$ where:

- X is the input set;
- Y is the output set;
- D is the name set of sub-components;
- $\{M_i \| i \in D\}$ is the set of sub-components, each M_i can be an atomic-DEVS or a coupled one;
- $C_{yx} \subset Y_i \times \bigcup_{i \in D} X_i$ is the set of internal coupling. It contains couples of output and input where outputs of sub-components are connected to inputs of sub-components;
- $C_{xx} \subset X \times \bigcup_{i \in D} X_i$ is the set of external input coupling. It contains all the inputs of sub-components minus the ones contained in C_{yx} (i.e., the ones that "go in" M);
- $C_{yy} \subset Y_i \rightarrow Y^{\phi}$ is the external output coupling function. It contains all the outputs of sub-components minus the ones contained in C_{yx} (i.e., the ones that "go out" from M);
- $Select : 2^D \rightarrow D$ is the tie-breaking function which defines how to select the event from the set of simultaneous events.

This coupled model N enables to **put together** atomic-DEVS or other coupled models. Finally, the notion of atomic and coupled DEVS enable to simulate entities that are close to the notion of component system of a SoS. An atomic-DEVS can model a component system and a coupled-DEVS can model interactions between component systems with the set of internal coupling C_{yx}. But the first limitation of classical DEVS concerns the dynamicity of the SoS architecture: it is not possible to change the connections (representing the interactions) at runtime. To fulfill this need (i.e., to simulate architecture dynamicity), an extension called ρDEVS is presented in [Hu et al., 2005] which introduces a new set in the atomic-DEVS enabling the change of the connections at runtime. Moreover, authors in [Mittal and Luis Risco Matin, 2013] propose an entire net-centric and client-server oriented platform based on DEVS formalism to simulate distributed models described with a common language (DEVSML).

2.8.2 Agent-Based Simulation

Agent-Based Simulation (ABS) uses to simulate ABM (Agent-Based Model). It is a paradigm of simulation that is really developed ([Bonabeau, 2002], [Behdani, 2012], [Mour et al., 2013]). It does not own a concrete formalism and is a flexible simulation paradigm: it is used to simulate systems according to a particular point of view that is the individual social behaviors and interactions of autonomous entities (also called agents). More precisely, it enables to implement specific agent behaviors and interaction rules between each other. This behavior is often based on the Perceive-Decide-Act cycle. [Bonabeau, 2002] argues that ABS (also called Agent-Based Modeling) is *powerful simulation modeling technique* that is useful when:

- *"the interactions between agents are complex, nonlinear, discontinuous or discrete [...]*;

- *space is crucial [...]*;
- *the population is heterogeneous [...]*;
- *agents exhibit complex behavior [...]"*.

Furthermore, ABS is used to exhibit emergent phenomena resulting from interactions between entities. [Behdani, 2012] argues that ABS is useful when the system to simulate:

- is *"individual-oriented; focus is on modeling the entities and interactions between them"*;
- is composed of "heterogeneous entities";
- is composed of micro-level entities that are active and interact with each other and evolves in an environment;
- is a system where the dynamic behavior comes from 'agents' decisions and interactions.

In [Mour et al., 2013], authors propose a comparison of ABS and Discrete Event Simulation and conclude that ABS is more suited to represent *"individual entities that drive the discrete events and allow for possible emergent behavior"*.

2.9 Discussion

Table 2.3 sums up the different evaluations of DEVS and ABS for SAHS. The legend is the following: the evaluation criterion for the heuristic is totally filled (++), partially filled (+), partially absent (-), totally absent (- -). Concerning our evaluations criteria, because DEVS models exchange only data of the same type (events), it enables pragmatic interoperability and ρDEVS enables dynamic architecture. Moreover, DEVS enables to produce heterogeneous models by simulating, for example, atomic-DEVS with different set of states S. But, authors argue that DEVS and its variants are not expressive enough at this time to model social behavior of component systems: in [Mittal and Luis Risco Matin, 2013], they argue that as the representation of cognitive capacities and negotiations protocols (such as collaboration, cooperation and so on) are future work.

As ABM paradigm enables to model SoS (see section 1.4.2) and as ABS concerns the simulation of ABM, ABS is well suited to simulate SoS. In ABM, there is no precise rule for defining the behavior or the rule of interactions of agents [Bonabeau, 2002]. It gives a huge flexibility to ABS to implement heterogeneous agents' models and dynamicity of the system architecture. Moreover, there is a flexibility concerning communication between agents enabling pragmatic interoperability. Finally, concerning simulation performance, [Bonabeau, 2002] argues that *"Simulating the behavior of all of the units can be extremely computation intensive and therefore time consuming"*.

We conclude that ABS paradigm is more suited to study SAHS. The main advantage is the possibility to study how the social behavior of component systems will influence the architecture. DEVS is an important and formal paradigm that brings advantages to study SoS, but it fails concerning implementing social behavior (such as cooperation) of component

	DEVS and variants	ABS
Pragmatic Interoperability	+	+
Heterogeneous Models	+	+
Dynamic Architecture	+	+
Dynamic Component Behavior	- -	+
Simulation Performance	++	+

Table 2.3 – DEVS & variants vs ABS for SAHS

systems and this is an important lack for SAHS because, for example, cooperation between component systems are the basis of SoS emergent behavior [Gonçalves et al., 2014]. This limitation does not enable to use it for SAHS. It is better to simplify the model of the component system to focus on social component behaviors. DEVS will not be used to simulate the works of this thesis. That is why the following sections will focus on existing tools for ABS and if they are usable for SAHS.

2.10 Existing Tools for SAHS

To be able to analyze SoS architecture, we propose a small review about existing tools concerning SAHS. Concerning evaluation of these tools, we think that the main need is to have the ability to implement a new generic SoS model. This need is important because we conclude in chapter 1 that existing generic models have some limitations we want to go through by proposing a new generic model. As there is a few works about SAHS, there are a few tools for testing existing SoS architecting heuristics. Main contributions are in [Edward Pape II, 2016] and [Caffall and Michael, 2009]. [Caffall and Michael, 2009] has already been presented through its heuristic in chapter 2. Authors test their heuristic on an Auto Battle Management Aid SoS. Unfortunately, there are not enough details to present properly tools they used. [Edward Pape II, 2016] chooses to implement their heuristic based on Agent-Based Wave model presented in chapter 2. The author tests it on a variety of problems like Search And Rescue SoS and Operation Other Than War (OOTW) scenarios and decides to develop its own tools to analyze, visualize and run his models.

Few tools exist and they have strong limitation: there is no concrete method to implement a new generic SoS model as we proposed to develop in the conclusion of the chapter 1. Furthermore, these tools are not in the public domain so are not reusable. For these reasons, the development of a dedicated tool for SAHS will be proposed. This tool is detailed in the chapter 8 of this document.

2.11 Conclusion

The two SoS architecting heuristics presented in this chapter have been presented and evaluated through chosen criteria. Table 2.4 sums the whole criteria for both models. The legend is the following: the evaluation criterion for the heuristic is totally filled (++), partially filled

	Collaborative Formation	Based Waved Model
Computationality	-	+
Genericity	+	+
Dynamic	-	-
Openness	- -	-
Computational cost	-	+
Decentralization	- -	- -
Cooperation	-	+

Table 2.4 – Evaluation of existing SoS architecting heuristics

(+), partially absent (-), totally absent (- -). We can notice that both fail concerning the total decentralization of the process disabling to architecting collaborative and virtual SoS. Moreover, both are based on a heavy process that leads to difficulties concerning the dynamicity and the openness. Finally, cooperation between component systems is not totally represented in both of them that are the basis for the SoS emergent behavior. For these reasons, we propose as a second contribution of this thesis a fully decentralized, open SoS architecting heuristic based on cooperation between component systems.

This chapter proposes a state of the art concerning SoS simulation and SAHS. By comparing both, evaluation criteria for SAHS are highlighted and used to evaluate two simulation paradigms that are DEVS and ABS. The result of this evaluation is that ABS is more suited to study SAHS especially because the easiness to study social behavior of component systems of a SoS. Then, existing tools are presented and limits are highlighted: they do not enable to implements easily a new generic SoS model. Finally, this thesis will propose to use ABS through a self-made tool called SoS Architecting HEurIStIc based on Agent (SApHESIA) to test a new SoS architecting heuristic based on cooperation.

3 The Adaptive Multi-Agent System (AMAS) Approach

After addressing SoS paradigm, this chapter focus on a MAS paradigm called the AMAS approach that will be the basis of our proposition of SoS architecting heuristic. Indeed, as explained in chapter 1, the MAS paradigm is a natural way to model SoS. Then, we think that AMAS approach is close enough to generic SoS paradigm to be able to propose a new heuristic formation based on **cooperation**. Thus, after presenting general definitions about MAS, this chapter introduces the AMAS approach to justify that it can be used to propose a new SoS architecting heuristic.

3.1 Multi-Agent Systems

Multi-Agent Systems (MAS) are generally defined as systems composed of interconnected autonomous parts called agents. MAS are mainly used to propose solutions to complex problems and to simulate biological phenomena or behaviors such as social insects. The following sections propose basic definitions about MAS.

3.1.1 MAS Definition

A MAS is made of parts that are called **agents** situated in an **environment** and interacting with each other. Ferber in [Ferber, 1995] defines a multi-agent system as a system composed of the following elements:

- ▷ textit"An environment E, which is a space with a metric" (he does not define the concept of metric);
- ▷ *"A set of objects O that are located"* (i.e., they can have a position in E at a given time);
- ▷ *"A set of agents A that are particular objects* ($A \subseteq O$ *that represents the active entities of the system"*;
- ▷ *"A set of relations R that link objects"*;
- ▷ *"A set of operations Op enabling to agents of A to perceive, produce, consume, transform and manipulate objects of O"*;

- *"Operators being responsible to represent the application of these operations and the reaction of the world to a modification"*.

3.1.2 Agent Definition

An agent is an autonomous piece of software which tries to fulfill its own goal by using its own reasoning and actions [Ferber, 1995]. It can communicate with other agents to help them or ask for services.
In order to complete the previous definition, [Ferber, 1995] defines an agent as a software entity that:

- evolves in an open computer system;
- can communicate with other agents;
- has its own goals;
- possesses its own resources;
- has only a partial representation of the other agents;
- has capabilities (services) that it can offer to other agents;
- has a behavior that tends to satisfy its own goals, taking into account its resources, the representations and the communications it receives.

To explain the functioning of an agent, the representation through the *Perception*, *Decision*, *Action* cycle is often used:

- the *Perception* phase, where the agent updates its representation of what it locally perceives of its outside (its environment and the agents around it);
- the *Decision* phase, where the agent uses its behavior to bind the available actions that will fulfill the most its goals considering its new representations (or perceptions);
- the *Action* phase, where the agent actually performs the chosen action, that implies changes in its environment and the agents around it.

This representation has led to define two types of agents: the reactive agent and the cognitive agent. On one hand, a reactive agent chooses its action depending only on the perceived events of the environment. Its behavior is represented through a set of $if < perception > then < action >$ with no goals explanation nor plan mechanism. In the $Perceive - Decide - Act$ cycle representation, this kind of agents is represented with an empty *Decision* phase: perceptions are directly bound to actions. The well-known example of collective reactive agents is the ant colony. Each ant reacts to a perceived event such as the presence of pheromone or food without reasoning.
On the other hand, a cognitive agent has a non-empty *Decision* set: it has a base of knowledge concerning information about other agents (such as their available resources, services

or even their emotional states [Rincon et al., 2016]), an expertise concerning the realization of its task, the interactions management with others agents and its environment. We also speak about "intentional" agents: they own goals they try to achieve.

3.1.3 Environment

[Weyns et al., 2005] argues that in MAS, the environment of a MAS is an important notion but does not have a formal definition. Nevertheless, [Di Marzo Serugendo et al., 2011] proposes a general definition through set theory by defining the world $W = ES \cup S$, S being the MAS and ES the environment is plunged into. ES can be described as being all what is outside the MAS ($ES = W/S$). Another definition is the one from [Russell and Norvig, 2002]. They define the environment as the problem where the MAS evolves and where the MAS is the solution of that problem. They propose to characterize the environment through four criteria:

- **Deterministic/Stochastic**: an environment is deterministic if its next state depends only on its current state and the action we apply on it (i.e., there is no stochastic variable that influences its state);
- **Fully observable/Partially observable**: an environment is fully observable if one agent is able to perceive the complete environment's state;
- **Static/Dynamic**: an environment is static if its state does not change over time by itself;
- **Discrete/Continuous**: an environment is discrete if its number of states is finite.

This characterization helps the MAS designer to choose the right agent design. As an example, in a partially observable environment, an agent is designed with limited perceptions. The composition of the environment depends on what the designers would like to focus. In the ant colony example, the environment can be in a first time represented by pheromones, obstacles, terrain, the nest and food. And in a second time, if a focus on the ability of ants to defend themselves against other insects needs to be studied, then other insects can be added to the environment.

These previous characterizations can be defined as the **physical** environment of the MAS. But the notion of environment exists also for an agent and can be defined as the union of the environment plus the other agents of the MAS. Finally it also exists the notion of **social** environment of an agent that is defined in [Di Marzo Serugendo et al., 2011] as *"the set of known agents"*. This social environment is important because it enables an agent to have a representation of skills or available resources of other agents that it needs to achieve its own objectives.

3.1.4 Organization

[Morin, 1977] defines an organization as *"an agency of relations between components or individuals that produces a unity (or system) owning qualities that are unknown from the components or the individuals. The organization binds together elements, events or individuals in an interrelation*

manner that become the parts of a whole. It enables solidarity and relative solidity and then enables a time duration of the system in despite of random perturbations". In a MAS, there are a lot of interrelations between agents through information exchanges, actions synchronization and so on. Moreover, the term organization defines the process of a structure elaboration and the result of this process. This duality shows that there is always a dynamic aspect attached to an organization: the organization is necessarily dynamic and always in a reorganization process of the entities and the links that bind these entities.

3.2 The Adaptive Multi-Agent System (AMAS) Approach

In computer science, user's needs and usages are never been so sophisticated, diversified and changing over time. Some examples such as smart cities [Paz et al., 2016], Web of Thing (IoT) [Mrissa et al., 2015] and the use of Big data show that this assertion will become truer in the future. Then, to fulfill these complex and heterogeneous user's needs, system designers have to build systems using heterogeneous and strongly interconnected parts. For example, a system of air-ballistic defense leads to different people (engineers, ballistic experts, military and so on) to work together with heterogeneous systems. System designers need to cope with a strong dynamic environment where unpredictable and heterogeneous events may occur: in the air-ballistic example, it can be cyber and physical attacks, critical system failure and so on. Another constraint of modern systems is the need to be up and running 100% of the time: this constraint makes more and more difficult to wait a human intervention for repairing the system. These two facts lead to design systems having the ability to repair, to change, to adapt itself or to **self-adapt** continuously [Di Marzo Serugendo et al., 2011]. Moreover, user's needs can also be under-specified; in that case designers have to build systems where the desired functionality is not known a priori.

AMAS approach has been developed to build these types of **self-adaptive** and **under-specified** systems. To enable self-adaptation, AMAS approach uses the concept of **self-organization**. To cope with **under-specification**, AMAS approach also uses the concept of **self-organization** enabling the **emergence of the adequate functionality**. These general notions are presented in the following sections.

3.2.1 Self-Adaptation

The justification of using the property of self-adaptation to cope with a dynamic environment comes from the following assertion derived from **the law of requisite variety Ashby** [Ashby, 1956]: controlling a system with a level of variety needs a system with at least more variety. The variety is the number of states a system can be. Indeed, the role of the controller is to lead and maintain the system to a desired state. Then, each state of the system has to be related to a state of the controller to manage it. But systems evolve in more and more complex and dynamic environments where a huge variety of events can occur. The construction of this external controller becomes more and more complex. Then,

a solution can be to build a system that is able **to cope itself** with unexpected situations with no need for external control: it is called a self-adaptive system.

The AMAS approach has been developed to design **self-adaptive** MAS [Capera et al., 2003], [Lemouzy, 2011], [Di Marzo Serugendo et al., 2011]. Moreover, AMAS approach is also used to design systems when the problem is under-specified and/or where no algorithmic solution is known a priori. It uses the principle of emergence of the adequate functionality of the system; this is the topic of the next section.

3.2.2 Under-Specification

In the AMAS approach, the adequate functionality of a system is defined as the functionality that solves the problem a system has been made for, regarding to the satisfaction of an exterior observer. To produce it, the AMAS approach uses the concept of **emergence** of the adequate functionality. It is used to help the system to cope with an under-specified problem. We propose to develop these notions of emergence and adequate functionality in the following sections.

a) Emergence...

Many definitions of emergence exist but no real consensus exists. Emergence in computer science [Di Marzo Serugendo et al., 2011], inspired by biological study, is a phenomenon observed in a system that may be the apparition of a pattern or a structure, a behavior, a property that is not present in the parts of the system. The emergent phenomenon is derived from the simple interactions of the parts and appears without any central control. Emergence is often presented as " The whole is more than the sum of the parts" [Bonabeau, 2002], [Di Marzo Serugendo et al., 2011]. This concept exists in the nature like, for example, the emergence of flocking behavior in group of birds or fishes. For [De Wolf and Holvoet, 2005], an emergent phenomenon, is produced when coherent macro level properties can be observed as a result of micro level interactions of the parts. Finally, an emergent phenomenon is characterized as [Georgé, 2004] :

- an **ostensible** phenomenon, meaning that it is self-evident for the observer at the macro level;
- a **radically new** phenomenon, meaning that it is neither observable nor explicable at the micro level;
- a **coherent** and **correlated** phenomenon, meaning that it has its own identity but is strongly related to the parts that produce it;
- a phenomenon that produces a **particular dynamic**, meaning that it is not predefined, it is self-created and self-maintained.

b) ...of the Functional Adequacy

The justification of using the concept of emergence in AMAS comes from the under-specification of the problem to solve. It leads to design system that is under-specified. The under-specification of a system means that the complexity of the problem to cope with does not enable to define the desired functionality of the system. To tackle this kind of problems, the AMAS approach enables to build systems that produce the adequate functionality (i.e., the function that solves the problem, regarding to the satisfaction of an exterior observer) by using **emergence** when classical **top-down** approaches cannot be used. A top-down approach is used by dividing the problem in simpler sub-problems to solve and then re-assembling the sub-solutions to solve the global problem. This kind of classical approach is not adapted for complex problems: dividing a complex problem into sub-parts leads to lose the complexity of the problem because the complexity may come from the strong interrelations of the sub-parts. For example, a problem can be complex to solve because of retro-acting loops produced by parts. If parts are solved independently, solutions do not take into account retro-acting loops effects. Reassembling these solutions does not solve the overall complex problem. This kind of approach cannot be used when the desired functionality of the system is not known a priori.

At the contrary, the AMAS approach is a **bottom-up** approach: the design is concentrated on the parts of the system that use **self-organization** for the **emergence** of the adequate functionality and for the **self-adaptation** to a dynamic environment. The next sections present how the notion of self-organization enables the self-adaptation and the emergence of the adequate functionality.

3.2.3 Self-Adaptation through Self-Organization

To enable self-adaptation, the AMAS approach uses the notion of self-organization. Authors in [Di Marzo Serugendo et al., 2011] defines self-organization as the process of a system that leads to the apparition of a global structure from interactions of its parts. In others words, it is the ability to change its internal structure or organization without explicit control or constraints from the outside of the system. Self-organization brings to the system properties like resilience and robustness. Resilience is the ability to continue to operate when a failure occurs. Robustness is the ability to stay efficient when an unexpected event in the environment occurs. As said previously, an organization enables the MAS to have a relative solidity to environment pressures (section 3.1.4). As self-organizing systems are resilient, they are able to cope with environment dynamics by changing their organization and continue to function. This is a desired property for systems that need to cope with a dynamic environment: self-organizing systems are able to change their organization to **adapt themselves**.

3.2.4 Functional Adequacy through Self-Organization

To enable emergence, the AMAS approach uses self-organization by only focusing on the local behavior of the agents until the apparition of the organization that produces the over-

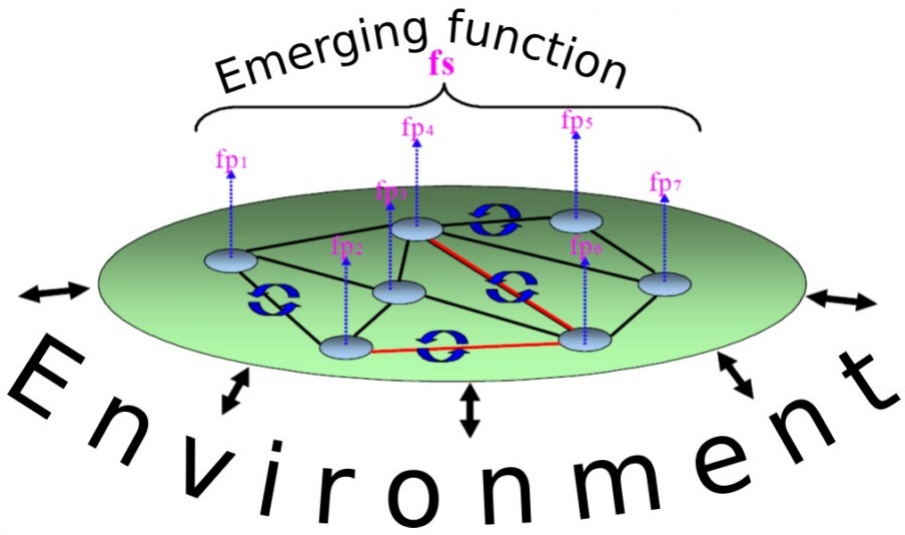


Figure 3.1 – Emergence of the function by self-organization [Picard, 2004]

all function (that seems satisficing for an exterior observer) of the system. The aim is to produce an organization that is an adequate response to the dynamics of the environment. In other words, the overall function, which is not implemented in any agents of the system (because this overall function is under-specified), will emerge from the self-organization of the agents. Figure 3.1 sums up this idea: an AMAS is a multi-agent system S that realizes a function f_s and where agents are interacting parts. Each agent a_i of S realizes a partial function $f_{p_i} \neq f_s$. The function of the system f_s results from the combination of the f_{p_i} ($f_s = f_{p_1} \circ f_{p_2} \circ ... \circ f_{p_n}$) and the current organization of the system. As generally $f_{p_1} \circ f_{p_2} \neq f_{p_2} \circ f_{p_1}$, by transforming the organization, the combination of the partial functions is changed and therefore the global function f_s is changed. In other words, by changing the agents interactions, the system self-organizes and as a result changes its function f_s to self-adapt to the changes of the environment and to find itself its adequate functionality. Then, an AMAS has to reach what is called the **functional adequacy**: to find the right organization to produce the right functionality f_s in a given environment. To drive the self-organization of the agents, the AMAS approach uses the **cooperation as a social behavior**.

3.2.5 Self-Organization through Cooperation

Figure 3.2 presents the philosophical view of cooperation: considering a group where dependencies exist between individuals, it is more valuable to work cooperatively than individually. The notion of cooperation is used in the MAS research area. It enables, for example, to allow group of agents to exploit the *collective intelligence* of other groups instead of only exploiting individual skills of each other [Bessadi et al., 2016]. In AMAS approach, [Picard, 2004] defines cooperation as the behavior at equal distance between selfishness and altruism. Thus, a cooperative agent tends to reach its own goals and at the same time helps its neighborhood to reach their own. Then, the cooperation is defined as a social behavior

that each agent does not depart from.

To justify that the cooperation between agents can lead to the functional adequacy, the theorem of functional adequacy is presented in [Glize, 2001] and [Camps, 1998]. In an informal manner, this theorem states that for each system S designed to achieve a particular task f_s, it exists a system having all its parts in a **cooperative state** achieving an equivalent task. This theorem is important because it justifies the use of cooperation as a social behavior to make systems that tend to the functional adequacy. Furthermore, doing a monolithic system that solves a particular complex problem is barely impossible, mainly because of the validation and the under-specification of this one. Thus, the main idea is to build systems able to self-organize to tend to reach the right functionality and then to self-adapt to the dynamics of the environment.

In AMAS approach, the cooperation is implemented through two mechanisms: the resolutions of Non-Cooperative-Situations (NCS) or the calculation of the criticality.

a) Cooperation through NCS Detections

If an agent detects that it is not cooperative with its neighborhood, it has to solve this situation called a Non-Cooperative-Situation (NCS). To classify NCS, the AMAS approach defines three meta-rules to instantiate depending on the problem to solve and based on the $Perceive - Decide - Act$ cycle. An agent is in a "cooperative state" if these three meta-rules are verified:

- c_{per}: a perceived signal has to be understood and not ambiguous; here, signal is a general term to point out something received or perceived by the agent (like a message, a video feed and so on);
- c_{dec}: a perceived information (i.e., an understand signal) has to be useful to produce a new decision of action in the environment;
- c_{act}: the consequences of its action have to be useful to other agents.

When at least one of these meta-rules is not verified ($\neg c_{per} \vee \neg c_{dec} \vee \neg c_{act}$) by an agent, this one is faced to a **non-cooperative** situation. Seven generic non-cooperative situations have been highlighted:

- incomprehension (¬cper): the agent cannot extract the semantic contents of a received signal;
- ambiguity (¬cper): the agent extracts several interpretations from a same signal;
- incompetence (¬cdec): the agent cannot benefit from the current knowledge state during the decision;
- unproductiveness (¬cdec): the agent cannot propose an action to do during the decision;

- ▷ concurrency (¬cact): the agent perceives another agent which is acting to reach the same world state;

- ▷ conflict (¬cact): the agent believes that the transformation it is going to operate on the world is incompatible with the activity of another agent;

- ▷ uselessness (¬cact): the agent believes that its action cannot change the world state or it believes that the results for its action are not interesting for the other.

When an agent has to cope with a, it has to apply counter-actions also called cooperative mechanisms to go back on a cooperative state:

1. Tuning: the agent realizes limited changes in its internal state in order to modify its partial function f_{p_i}. For example, changing the values of parameters that influence its partial function;

2. Reorganization: if tuning mechanisms failed, the agent tries to change its interactions with its neighborhood. It can change its neighborhood and then the organization of the system;

3. Evolution: if tuning and reorganization mechanisms failed, the agent uses an evolution mechanism enabling to create or delete agents. An agent creation occurs when any agent is able to fulfill a partial function f_p that seems necessary for one of existing agents. The suppression of an agent occurs when the functionality of an agent is not useful anymore.

b) Cooperation through Criticality

Another way for an agent to be cooperative is to use the notion of criticality. Indeed, another vision of being cooperative for an agent may that to **help** in priority the other agents (including itself) of its neighborhood that are more critical than it. To compute this notion of level of cooperation, each agent computes and uses a metric called "criticality". The notion of criticality is then introduced to enable the agent to take locally the best decision for itself and its neighborhood. Basically, the criticality represents the distance between the agent's current state and its goal. If its current state is far from its goal, then the criticality is high. If it is close, then the criticality is low. Then, when an agent has to decide, it locally compares its criticality with the criticalities of its neighbor agents. As it is cooperative, it has to choose the action that helps its most critical neighbor (itself included) even if this action increases its own criticality except if it becomes the most critical one. This metric is already used in developed AMAS ([Kaddoum, 2011], [Lemouzy, 2011]) but as its definition is problem-dependent, it is not reusable for other problems. One contribution of this thesis is to propose a formal and generic definition of the criticality, which will be explained in chapter 5.

Figure 3.2 – The 'Two mules' cooperation metaphor

3.2.6 Adequation of AMAS Approach for SoS Architecting

The AMAS approach is based on Agent-Based Model (ABM) paradigm and as seen in chapter 1, ABM is well-suited for modeling SoS. As a reminder, the agent paradigm can be naturally used to model a component system of a SoS because of the similarities between the two paradigms (see chapter 1 for more details). Furthermore, chapter 2 defines the following evaluation criteria for SoS architecting heuristic: computationality, genericity, dynamicity, openness, computational cost, decentralization and cooperation.
The AMAS approach is easily **computational** and is **generic** enough because it has been implemented by a consequent number of authors and in a huge variety of areas such as information access customization [Lemouzy, 2011], control systems[Boes, 2014], ambient systems [Guivarch, 2014] and mechanisms design [Capera, 2005].
It is a **dynamic** heuristic because by self-organizing, it enables to self-adapt to unpredicted changes in the environment.
It enables to add or remove agent (through the cooperation mechanisms) during functioning so it is an **open** approach.
As an AMAS agent has a limited perception of its environment: it enables to limit the **computational cost** of the approach.
The AMAS approach is fully **decentralized**: there is no central management and the overall functionality emerges only from the self-organization of the agents.
The approach is **cooperative** because each agent uses the cooperation as a social behavior and cannot depart from this behavior.
Finally, the characteristics of the AMAS approach fulfill the evaluation criteria for SoS architecting heuristic. Then, a SoS architecting heuristic based on the AMAS approach is proposed in chapter 5 of this document.

State of the Art: Conclusion

This first part presents a state of the art concerning a particular kind of complex systems called System of Systems (SoS) through existing definitions, models and simulations paradigms. In a first time, it presents the notion of System Of System through a wide overview of existing definitions, characterizations and classifications. They introduce the idea that SoS is a relatively recent research area where efforts have to be made concerning a wide variety of fields like taxonomy, modeling, simulation, architecting heuristics and so on. They also enable to extract the main characteristics of a SoS with regards to classical systems and a working definition about SoS used in the rest of this document. Then, an overview of generic SoS models is presented. Mainly based on set theory and Agent-Based Model (ABM), they are evaluated through defined criteria issued from SoS definitions and characteristics such as the ability to model component systems heterogeneity, operational and managerial independences and so on. This overview brings forwards limitations concerning the ability to model a dynamic environment and a more expressive notion of interactions.

After defining SoS architecting, an overview of generic SoS architecting heuristic is presented. Then, they are evaluated through criteria derived from the core principles for efficient SoS Architecting by [Azani, 2008] and from general thinking about SoS architecture such as the ability to be computational and to architect fully decentralized SoS like collaborative SoS, to cope with a dynamic environment and so on. This overview brings forwards limitations concerning the ability to cope with a dynamic environment and to architect fully decentralized SoS such as collaborative SoS. Then, a presentation and a differentiation between classical systems, SoS and SoS Architecting simulations through criteria we propose are made. They enable to present and evaluate simulation paradigms like DEVS and ABS and to choose ABS for doing SAHS because it enables to simulate dynamic component behaviors. An overview of existing tools for SAHS is then presented as well as their limitations (such as their problem-dependence).

Finally we proposed to introduce the reader to the Adaptive Multi-Agent System (AMAS) approach that enables to design self-adaptive and under-specified Multi-Agent Systems (MAS) through self-organization of the agents. This self-organization also drives the emergence of the adequate function of the system. This self-organization is driven by the notion of cooperation between agents. This cooperation is a social attitude that can be

modeled by Non-Cooperatives-Situations (NCS) or level of criticality. Finally, for these all the limitations presented above, we propose in the following part to go further these limitations:

- ▷ a new generic SoS model called SApHESIA model that is agent-based and where the notion of environment is more developed;
- ▷ a new SoS architecting heuristic based on cooperation between parts that enables to architect decentralized SoS such as collaborative SoS;
- ▷ a generic tool enabling to experiment generic SoS through SApHESIA model.

Then, these contributions are validated through experimentations available in the third part of this document.

Part II

Contribution to SoS Modeling and Architecting

4 SApHESIA Model

SoS literature does not propose a huge quantity of generic SoS models. Moreover, existing ones are not expressive enough, do not enable to model a dynamic environment and interactions between component systems. In this chapter, we propose a new SoS model called SoS Architecting HEuriStIc based on Agents (SApHESIA) model enabling to model more expressive problems than existing SoS models ([Acheson et al., 2012] and [Baldwin and Sauser, 2009]), to compute them and to use cooperation as a heuristic of SoS architecting. This chapter presents in details the different parts of the SoS generic model we propose. Throughout the chapter, this model is exemplified through the Missouri Toy Problem ([Edward Pape II, 2016]) to evaluate the adequacy of the model with SoS paradigm. The Missouri Toy Problem has been chosen for its representativeness of a SoS problem and also because of its simplicity of explanation.

4.1 Presentation of the Missouri Toy Problem

The Missouri Toy Problem is a scenario initially presented by [DeLaurentis et al., 2012]. The aim of this SoS is to *relay commands and ISR data from ground station to a Carrier Battle Group via Unmanned Aerial Vehicles (UAVs) and Satellites* and to check if a communication link is always up between the ground station and the carrier. The ground station needs the UAVs and

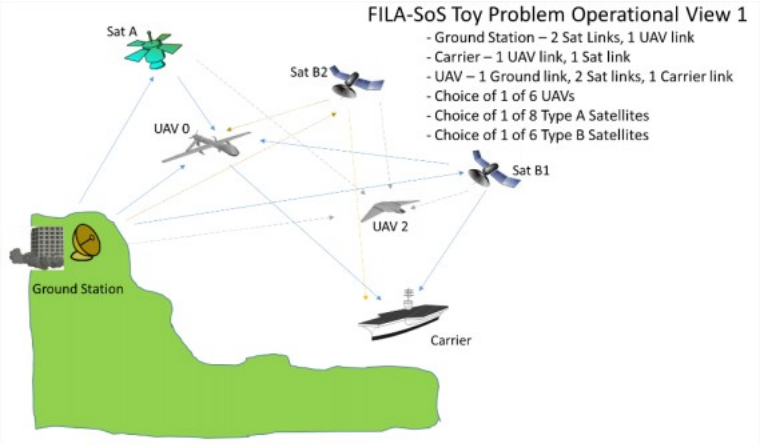


Figure 4.1 – The Missouri Toy Problem

satellites because it cannot send signals directly to the carrier battle group. Then, this problem has been extended by [Edward Pape II, 2016] *"for purposes of having a few more systems to choose from"*. More precisely, the following types of component systems can be interfaced with each other:

- the ground station *Ground*; its functionality is to *send a signal* to satellites and UAV;
- the UAV UAV; its functionality is to *relay a signal* to UAV and other satellites;
- the satellite of type A Sat_A; its functionality is to *relay a signal* to UAV and other satellites of type A;
- the satellite of type B Sat_B; its functionality is to *relay a signal* to UAV and other satellites of type B;
- the carrier battle group *Carrier*; its functionality is to *receive a signal* from other component systems except the ground station;

Every of them can communicate with each other except:

- *Ground* that cannot interface with *Carrier*;
- Sat_A that cannot interface with Sat_B and vice versa.

Events like cyber-attacks, weather issues, jamming of communication links are unexpected events occuring in the environment. Furthermore, the availability of component systems can perturb interactions between component systems.

The solution proposed in [Edward Pape II, 2016] consists in using an approach based on a genetic algorithm close to the agent-based wave model presented in chapter 2. As a reminder, to compute and evaluate new chromosomes, the global function of this method uses Key Performance Attributes (KPAs) that are chosen by stakeholders of the SoS depending on what they want to evaluate such as performance, robustness and affordability.

4.2 General Structure of the SoS Architecting HEuriStIc based on Agents (SApHESIA) Model

As existing generic SoS models are not expressive enough, do not enable to model a dynamic environment and interactions between component systems. We propose the SApHESIA model composed of the main following components: the **SoS** and the **environment**. The SoS is composed of **component systems** and **goals**. The environment (containing the objects and constraints that are not parts of the SoS) is composed of **entities** and **rules**. These components are illustrated in figure 4.2 and detailed in the following sections.

4.3 SoS and Component System Models

This section presents the SoS model and the component system model.

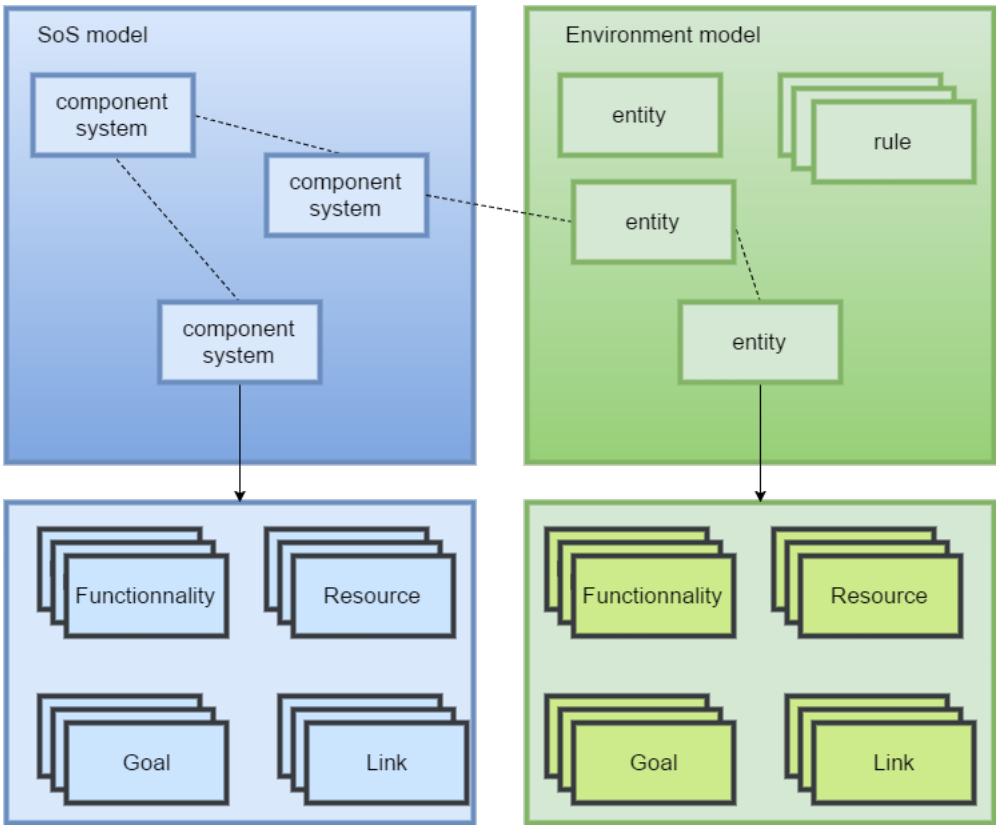


Figure 4.2 – Components overview of SApHESIA model

4.3.1 Component System Model

A component system is the smallest part of a SoS, it represents the second S of SoS. Formally, a component system S is defined as:

$$S = \{T, R, Acq, L, F, G, Cost\}$$

where:

- T is the type of S;
- $R = \{R_1, ..., R_n\}$ is a set of **resources**;
- $Acq = \{Acq_1, ..., Acq_m\}$ is a set of **acquaintances** with other component systems;
- $L = \{L_1, ..., L_p\}$ is a set of **links** with other component systems;
- $F = \{F_1, ..., F_q\}$ is a set of **functionalities**;
- $G = \{G_1, ..., G_r\}$ is a set of **goals**;
- $Cost \in \mathbb{R}$ is the **cost** of the component system.

a) Type

The type enables to model the different types of the component systems in the SoS. For example, a component system u of type UAV is represented by $u = \{UAV, R_u, Acq_u, L_u, F_u, G_u, Cost_u\}$.

b) Resource

A **resource** R_i is a structure $R_i = \{type : String, quantity : Float\}$ which represents passive elements in the SoS (i.e. which has no effector on the SoS itself).
For example, if a component system a is of type Sat_A in the Missouri Toy Problem, a resource can be $R = \{Signal, 1\}$. It means that a owns one $Signal$ resource.

c) Acquaintance

An acquaintance is an oriented association between two component systems. It enables to model the knowledge of a component system by another component system but does not enable the exchanges of resources between these two component systems. An acquaintance is associated with a component system and is modeled by a singleton representing the acquainted component system.
For instance, if an UAV (modeled through a component system $u = \{UAV, R_u, Acq_u, L_u, F_u, G_u, Cost_u\}$ of type UAV) knows a satellite b of type Sat_B, this acquaintance $aq \in Acq_u$ is represented by $aq = \{b\}$.

d) Link

A link is an oriented association between two component systems. A link enables to model of exchange channels between two component systems such as communication channels or channels of resource exchange. A link owns also a strength of association enabling to represent that two component systems can be more or less linked together. To create a link with another component system, a component system must have this one in its acquaintances. Then, the acquaintance of a component system is a precondition for link creation. Then, when a component system decides to destruct a link with another one, it may keep it in its acquaintances, enabling to recreate this link at another time.
For a component system $S = \{T, R, Acq, L, F, G, Cost\}$, a link $l \in L$ representing the association from S to S' is defined as:

$$l = (S', soa)$$

where $soa \in]0, 1]$ is the strength of this association. For instance, in our example if a satellite (modeled through a component system $a = \{Sat_A, R_a, Acq_a, L_a, F_a, G_a, Cost_a\}$ of type Sat_A) has a bad communication channel with a UAV (modeled as a component system u of type UAV), the link $l \in L_a$ is represented by $l = (u, 0.3)$.

e) Functionality

A functionality is an effector on the SoS itself. The functionality can affect :

- ▷ the resources of component systems;
- ▷ the links of component systems.

A functionality $\mathcal{F}$ is defined as a triplet :

$$\mathcal{F} : \{f, t, p\}$$

where:

- ▷ t is the execution time of $\mathcal{F}$;
- ▷ $p \in [0, 1]$ is the performance of $\mathcal{F}$. It represents the probability of $\mathcal{F}$ to success.
- ▷ f is the function of $\mathcal{F}$. It is defined as:

$$f : Conditions \rightarrow Effects$$

Conditions and $Effects$ are sets that respectively represent the conditions for the function to be executed and the effects applied on the SoS or the environment once the functionality is executed. *Conditions* may represent:

- ▷ a certain quantity of resources;
- ▷ the existence of a link between two component systems;
- ▷ the existence of a component system.

Effects may represent the change of:

- ▷ a certain quantity of resources;
- ▷ the existence of a link between two component systems.

To present the different kinds of *Conditions* and $Effects$, we consider a functionality $F_j = \{f_j, t_j, p_j\} \in F$, with F the set of functionalities of a component system $S = \{T, R, Acq, L, F, G, Cost\}$ and $f_j = \{Conditions \rightarrow Effects\}$. The next paragraphs present the different kinds of conditions and different kinds of effects.

A resource condition $Co \in Conditions$ concerns the comparison of a certain kind of resource owned by a certain component system and is represented by :

$$Co = \{Sys.R(Re) \ \mathtt{op} \ Qu\}$$

Where:

- $Sys \in \{Self, T'\}$ is the type of component system involved in the condition Co. If the resource condition concerns the component system S that executes the functionality, then $Sys = Self$. It can be also a component system of type T' linked with S (then, $\exists l = (S_l, soa) \in L | S_l.T = T'$);
- Re is a type of resource;
- $Sys.R(Re)$ is the quantity of resource Re in R (the set of resources of the component system of type Sys);
- $Qu \in \mathbb{R}$ is the involved quantity of resource of type Re owned by Sys;
- $\mathtt{op} \in \{=, <, >, \leqslant, \geqslant\}$ is the comparison operator.

To make the formulas easier to read, when the condition has to apply on the component system which executes the functionality ($Sys = Self$), the condition will be written as follow $Co = \{Re \ \mathtt{op} \ Qu\}$.

For example, $Co = \{UAV.R(Signal) \geqslant 1\}$ is true if S has a link with a component system of type UAV that owns at least one $Signal$ resource. $Co = \{Signal \geqslant 1\}$ is true if S owns at least one $Signal$ resource.

A link condition represents the existence of a link between the component system S and another component system of type T'. It is defined by:

$$Co = \{\exists l = (S_l, soa) \in L | S_l.T = T'\}$$

$Co = \{\exists l = (S_l, soa) \in L | S_l.T = Sat_B\}$ is true if the component system S has a link with a component system of type Sat_B.

An existence condition represents the knowledge of the existence of a component system of type T' by S. It is defined by:

$$Co = \{\exists S_{co} \in Acq | S_{co}.T = T'\}$$

$Co = \{\exists S_{co} \in Acq | S_{co}.T = Carrier\}$ is true if the component system S knows one component system of type $Carrier$.

A resource effect $Ef \in Effects$ represents the generation or the consumption of a given quantity of resource. It is represented by:

$$Ef = \{Sys.R(Re) \circ\!\!\mathrm{p}\ Qu\}$$

where:

- $Sys \in \{Self, T'\}$ is the type of component system involved in the effect Ef. It can be the component system that executes the functionality ($Self$). It can also be another component system of type T' **that is linked to S** (then, $\exists l = (S_l, soa) \in L | S_l.T = T'$);
- Re is a type of resource;
- $Sys.R(Re)$ is the quantity of resource Re in R that is the set of resources of S_l or S (depending on Sys);
- $Qu \in \mathbb{R}$ is the involved quantity of resource of type Re owned by S_l or S (depending on Sys);
- $\circ\!\!\mathrm{p} \in \{=, +=, -=\}$ the operator applied on $Sys.R(Re)$.

$Ef = \{Signal\ +=\ -3\}$ means that the component system S will consume three $Signal$ resources.

A link effect corresponds to the creation of a link from the component system S to another component system of type T'. It is represented by:

$$Ef = \{T', soa\}$$

Then, $Ef = \{UAV, 0.7\}$ means that the component system S will create a link towards a component system of type UAV with a $soa = 0.7$.

As an example, the component system $Ground$ from the Missouri Toy Problem owns functionality $\mathcal{F}_{Send}$ as:

$$\mathcal{F}_{Send} : \{\{Signal \geqslant 1\} \rightarrow \{\{Signal\ +=\ -1\}, \{Sat_A.R(Signal)\ +=\ 1\}\}, 0.01, 0.99\}$$

$\{Signal \geqslant 1\}$ is a condition resource. To clarify the syntax, we remind that the component system involved is not written when a condition is applied on itself. This is the same rule for effects. This condition means that the component system needs at least 1 resource of type $Signal$ to execute f. $\{Sat_A.R(Signal)\ +=\ 1\}$ is an effect resource. Finally, $\mathcal{F}$ models that the ground station is able to send a signal to a system of type $SatA$. It represents its ability to produce one signal (this signal is also consumed by the second effect resource) provided that it has one signal to send. The duration of the process is 0.01 unit of time and the probability to success is 99 %.

f) Goal

A goal is the special state a component system tries to reach with a given priority. This state can be:

- to own of a certain quantity of a type of resource;
- the existence of a link between this considered component system and another one.

A goal can be defined in two different ways.
The first one is to express that a component system tries to own a certain quantity resource:

$$G_R = \{Re \ \mathtt{Op} \ Qu, Pr\}$$

where:

- Re is a type of resource;
- $Qu \in \mathbb{R}$ is the quantity of resource of type Re the component system wants to own;
- $\mathtt{Op} \in \{=, \neq, <, >, \leqslant, \geqslant\}$ is the comparison operator in order to compare Qu to the current quantity of Re;
- $Pr \in \mathbb{N}^+$ is the priority of the goal enabling to model the relative importance of a goal. The higher the priority is, the more important the goal is.

As an example, a *Carrier* component system whose goal is to receive signals (that are represented with the *Signal* resource) with a priority of 1 is defined as :

$$G_{Signal} = \{Signal > 0, 1\}$$

The second one is used to express that a component system tries to add a link with another component system of type T'.

$$G_L = \{T'\}$$

where :

- T' is the type of a component system;
- $Pr \in \mathbb{N}^+$ is the priority of the goal.

As an example, a component system of type S trying to add a link with a component system of type *Carrier* with a priority of 3 is defined as:

$$G_L = \{Carrier, 3\}$$

4.4 SoS Model

A SoS is represented by a set of sets:

$$SoS = \{\mathcal{S}, \mathcal{G}\}$$

where:

- $\mathcal{S}$ is a set of component systems ;
- $\mathcal{G}$ is a set of goals; these goals are a subset of the goals of the component systems in $\mathcal{S}$.

As an example, the Missouri Toy Problem is instantiated with the following component systems:

- 1 ground station *Ground* (noted g);
- 5 UAV (noted $u_i, i \in [\![1,5]\!]$);
- 8 Sat_A (noted $a_i, i \in [\![1,8]\!]$);
- 6 Sat_B (noted $b_i, i \in [\![1,6]\!]$);
- 1 carrier battle group (noted c).

$$SoS = \{\{g, u_1, u_2, u_3, u_4, u_5, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, b_1, b_2, b_3, b_4, b_5, b_6, c\}, \mathcal{G}\}$$

The global goal of the SoS is to be able to *relay as much as possible the signals from the ground station to the carrier battle group*. Then, the global goals of the SoS can be defined as: $\mathcal{G} = \{G_{Carrier} = \{Signal > 0, 1\}, G_{Ground} = \{Signal == 0, 1\}\}$

$G_{Carrier}$ means that c wants the most $Signal$ resources as possible with a priority of 1. G_{Ground} means that g want zero $Signal$ resource a priority of 1. $G_{Carrier}$ and G_{Ground} are goals that will be respectively defined in the carrier c and the ground g in the following sections.

4.5 Environment Model

This section contains the model of the environment in which the considered SoS evolves. The environment contains two mains objects:

- **Rules** that represent the frame in which the SoS evolves. They enable to model external constraints that affect the SoS;
- **Entities** that represent active independent objects that are able to affect the environment or the SoS itself but which are not parts of the SoS. It is close from the notion of component system in a SoS.

Formally, an environment is defined as a set of **entities** and **rules**:

$$\mathcal{E} = \{E, Rules\}$$

where:

- E is a set of entities;
- $Rules$ is a set of rules.

4.5.1 Entity Model

An entity is an active independent object that is not a part of the SoS. It is defined as a set of sets :

$$E_i = \{T, R, Acq, L, F, G\}$$

where:

- T is the type of E;
- $R = \{R_1, ..., R_n\}$ is a set of **resources**;
- $Acq = \{Acq_1, ..., Acq_m\}$ is a set of **acquaintance**;
- $L = \{L_1, ..., L_p\}$ is a set of **links**;
- $F = \{F_1, ..., F_q\}$ is a set of **functionalities**;
- $G = \{G_1, ..., G_r\}$ is a set of **goals**;
- n, m, p, q, r are natural numbers.

The notions of type, resource, acquaintance, link, functionality and goal are the same as the one defined in a component system. Nevertheless, an entity can be linked to another entity or a component system. This is not the case with component systems : they can only be linked to other component systems.

As an example, in the Missouri Toy Problem, an entity may be a spy satellite able to cyber-attack the SoS by destructing existing signals of UAV component systems: $F_{DestructSignal} = \{\{UAV.Re(Signal) > 0\} \rightarrow \{\{UAV.Re(Signal) = 0\}, 0.3, 0.99\}$. It can be also other battle elements such as ships or airplanes that are not in the SoS because they are owned by another army.

4.5.2 Rule Model

A rule is a set of conditions and effects that enables to model how the environment reacts, evolves and interact with the SoS:

$$Rule = \{Conditions \rightarrow Effects\}$$

A rule needs to fulfill its *Conditions* to apply its $Effects$. A rule affects all the entities in the environment or all component systems in the SoS that fulfill the *Conditions*.

As an example, in the Missouri Toy Problem, the resource *Signal* of the ground g is generated from the following rule:

$$Rule_{Signal} = \{g.R(Signal) < 10 \rightarrow g.R(Signal) \mathrel{+}= g.R(R_{GenerationRate})\}$$

$Rule_{Signal}$ represents a rule applied on g (which is ground station) on its resource *Signal*. When its resource *Signal* is lower than 10 then $R_{GenerationRate}$ is added to $g.R(Signal)$. $R_{GenerationRate}$ is a resource of g representing the generation rate of signal to send.

4.5.3 SApHESIA Model with SoS Characteristics

To evaluate the compatibility of SApHESIA model with SoS paradigm, Table 4.1 shows the main components of SApHESIA with regards to the evaluation criteria we defined in chapter 1 in section 1.4.1 to evaluate generic SoS model.

We remind that, in our working definition, component systems tend to be heterogeneous in terms of capabilities (i.e., actions). Then, **heterogeneity** of component systems are given in SApHESIA thanks to the concept of functionality, resource, goal and cost: Component systems with different functionalities represent component systems that have different possibilities of action. **Operational independence** is also given with functionalities: they enable the component system to act. **Managerial independence** is modeled through the concept of goals: each component system may act by itself for itself. **Geographic distribution** is modeled through resource. As an example, an UAV can have R_X, R_Y and R_Z resources that represent a position in a space. Component systems can **interact** with each other through functionalities, acquaintances and links.
SoS is modeled through its own goal G (enabling to model Directed SoS) and its set of component systems $\mathcal{S}$. **Metric of evaluation** can be defined through the concepts of resources, goals and cost. As an example, in the Missouri Toy Problem, an interesting metric concerning the efficiency of the SoS can be the ratio between the number of generated signals by the ground g and the number of signals received by the carrier c. This two numbers can be given by the resources sets of g and c.
The **environment model** is dynamic because composed of independent and active entities as well as rules. Moreover, rules enable to express a non trivial behavior of the environment: creation of threats such as spy satellite, failure of links through the concept of rule. Finally, the model has a higher **global expressiveness** than models presented in chapter 1 because of a more expressive model of environment and actions (through preconditions and effects of functionalities).

	Component system							SoS		Env	
	T	R	Acq	F	L	G	$Cost$	$\mathcal{G}$	$\mathcal{S}$	$Rules$	$Entities$
Component system evaluation criteria											
Heterogeneity	x	x		x		x	x				
Operational independence		x		x							
Managerial independence						x					
Geographic distribution		x									
Interactions			x	x	x						
SoS evaluation criteria											
SoS model								x	x		
Metric definitions		x				x	x		x		
Dynamic environment model										x	x
Global expressiveness	x	x	x	x	x	x				x	x

Table 4.1 – Adequation of SApHESIA elements to SoS paradigm through evaluation criteria

4.6 Instantiation of the SApHESIA Model to the Missouri Toy Problem

This section presents the entire model of the Missouri Toy Problem. First, we remind the SoS composition in terms of component systems and its goals. Then, we remind the different component systems involved in the model by presenting the 5 different types of component systems: $Ground$, UAV, Sat_A, Sat_B and $Carrier$. The different components of the environment such as rules are then defined.

4.6.1 SoS Model Definition

The SoS is composed of 1 ground station, 5 UAVs, 8 satellites of type A, 6 satellites of type B and 1 carrier. The main goals of the SoS are that the ground sends all its signals and that all these signals are received by the carrier:

$$SoS = \{\{g, u_1, u_2, u_3, u_4, u_5, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, b_1, b_2, b_3, b_4, b_5, b_6, c\}, \{G_{Ground}, G_{Carrier}\}\}$$

With $\mathcal{G} = \{G_{Carrier} = \{Signal > 0, 1\}, G_{Ground} = \{Signal == 0, 1\}\}$.

4.6.2 Component Systems Definitions

All the component systems as well as their functionalities and goals are defined hereafter. The different numerical values for resources and cost are given as example:

$$Ground = \{T_{Ground}, R_{Ground}, Acq_{Ground}, L_{Ground}, F_{Ground}, G_{Ground}, Cost_{Ground}\}$$

With:

- $T_{Ground} = Ground$

- $R_{Ground} = \{\{Signal, 20\}, \{R_{LinkResource}, 5\}, \{R_{Turn}, 0\}, \{R_{FinalTurn}, 500\}, \{R_{GenerationRate}, 0.3\}\}$
- $Acq_{Ground} = \emptyset$
- $L_{Ground} = \emptyset$
- $F_{Ground} = \{F_{LinkUAV}, F_{LinkSatA}, F_{LinkSatB}, F_{Send}, F_{Unlink}\}$
- $G_{Ground} = \{G_{Link}, G_{LinkResource}, G_{Send}\}$
- $Cost_{Ground} = 100$

R_{Turn} is a resource used in the rule $Rule_{Turn}$ (given in the section describing the environment) to keep the current simulation cycle. $R_{FinalTurn}$ and $R_{GenerationRate}$ are resources used in the rule $Rule_{GenerateSignal}$ (given in the section describing the environment). $R_FinalTurn$ represents the cycle number when $Signal$ is no more generated and $R_{GenerationRate}$ the generation rate of $Signal$. $G_{Send} = \{Signal == 0, 1\}$ concerns the sending of signals. It enables to model the objective for $Ground$ to send $Signal$ to other component systems. $G_{Link} = \{Link > 0, 1\}$ concerns the creation of links with other component systems. It enables to model the objective for $Ground$ to link with other component systems.
We assume that each component system has limited number of available connections. To model this constraint, the goal $G_{LinkResource} = \{R_{LinkResource} > 0, 1\}$ concerning the preservation of the $R_{LinkResource}$ is defined to limit the number of link. Then, $R_{LinkResource}$ is incremented in the different functionalities of link creation (here $F_{LinkUAV}$, $F_{LinkSatA}$ and $F_{LinkSatB}$). As all of these functionalities concerning link creation are similar, we propose this common definition :

$$F_{LinkSyS} = \{\{\{SyS \in Acq\}\} \rightarrow \{\{Link, 1\}, \{SyS.R(R_{LinkResource}) \mathrel{+}= -1\}, t_{Link}, p_{Link}\}$$

With $Sys \in \{UAV, Sat_A, Sat_B, Carrier\}$.

F_{Unlink} is the functionality used for removing a link:

$$F_{Unlink} = \{\{\{SyS \in L\}\} \rightarrow \{\{Link, -1\}, \{SyS.R(R_{LinkResource}) \mathrel{+}= 1\}, t_{Unlink}, p_{Unlink}\}$$

With $Sys \in \{UAV, Sat_A, Sat_B, Carrier\}$.

F_{Send} is the functionality used for sending signal:

$$F_{Send} = \{\{Signal \geqslant 1\} \rightarrow \{\{Signal \mathrel{+}= -1\}, \{SyS.R(Signal) \mathrel{+}= 1\}\}, t_{Send}, p_{Send}\}$$

With $Sys \in \{UAV, Sat_A, Sat_B, Carrier\}$.

$$UAV = \{T_{UAV}, R_{UAV}, Acq_{UAV}, L_{UAV}, F_{UAV}, G_{UAV}, Cost_{UAV}\}$$

With:

- $T_{UAV} = UAV$
- $R_{UAV} = \{\{Signal, 0\}, \{R_{LinkResource}, 5\}\}$
- $Acq_{UAV} = \emptyset$
- $L_{UAV} = \emptyset$
- $F_{UAV} = \{F_{LinkUAV}, F_{LinkSatA}, F_{LinkSatB}, F_{LinkCarrier}, F_{Send}, F_{Unlink}\}$
- $G_{UAV} = \{G_{Link}, G_{LinkResource}, G_{Send}\}$
- $Cost_{UAV} = 100$

$$Sat_A = \{T_{Sat_A}, R_{Sat_A}, Acq_{Sat_A}, L_{Sat_A}, F_{Sat_A}, G_{Sat_A}, Cost_{Sat_A}\}$$

With:

- $T_{Sat_A} = Sat_A$
- $R_{Sat_A} = \{\{Signal, 0\}, \{R_{LinkResource}, 5\}\}$
- $Acq_{Sat_A} = \emptyset$
- $L_{Sat_A} = \emptyset$
- $F_{Sat_A} = \{F_{LinkUAV}, F_{LinkSatA}, F_{LinkCarrier}, F_{Send}, F_{Unlink}\}$
- $G_{Sat_A} = \{G_{Link}, G_{LinkResource}, G_{Send}\}$
- $Cost_{Sat_A} = 100$

$$Sat_B = \{T_{Sat_B}, R_{Sat_B}, Acq_{Sat_B}, L_{Sat_B}, F_{Sat_B}, G_{Sat_B}, Cost_{Sat_B}\}$$

With:

- $T_{Sat_B} = Sat_B$
- $R_{Sat_B} = \{\{Signal, 0\}, \{R_{LinkResource}, 5\}\}$
- $Acq_{Sat_B} = \emptyset$
- $L_{Sat_B} = \emptyset$
- $F_{Sat_B} = \{F_{LinkUAV}, F_{LinkSatB}, F_{LinkCarrier}, F_{Send}, F_{Unlink}\}$

- $G_{Sat_B} = \{G_{Link}, G_{LinkResource}, G_{Send}\}$
- $Cost_{Sat_B} = 100$

$$Carrier = \{T_{Carrier}, R_{Carrier}, Acq_{Carrier}, L_{Carrier}, F_{Carrier}, G_{Carrier}, Cost_{Carrier}\}$$

With:

- $T_{Carrier} = Carrier$
- $R_{Carrier} = \{\{Signal, 0\}, \{\{R_{LinkResource}, 5\}\}$
- $Acq_{Carrier} = \emptyset$
- $L_{Carrier} = \emptyset$
- $F_{Carrier} = \emptyset$
- $G_{Carrier} = \{\{Signal > 0, 1\}\}$
- $Cost_{Carrier} = 100$

we model the send of a signal by the following function :

$$F_{SendToSyS} = \{\{Signal \leqslant 1\} \rightarrow \{\{Signal \mathrel{+}= -1\}, \{SyS.R(Signal) \mathrel{+}= 1\}, t_{Send}, p_{Send}\}$$

With $Sys \in \{UAV, Sat_A, Sat_B, Carrier\}$.

As a reminder, Sat_A and Sat_B component systems are not able to link together, and the *Carrier* component system does not link with other component systems. Table 4.2 sums up the different functionalities available for every type of component system.

Table 4.2 – Available link functionalities according to the type of component system

	$Ground$	UAV	Sat_A	Sat_B	$Carrier$
$F_{LinkUAV}$	X	X	X	X	
$F_{LinkSatA}$	X	X	X		
$F_{LinkSatB}$	X	X		X	
$F_{LinkCarrier}$		X	X	X	

4.6.3 Environment Definition

The environment of the Missouri Toy Problem contains two rules and no entities. The first one, $Rule_{Turn}$ increments the resource R_{Turn} of ground g which contains the number of cycles of the simulation. This information will be used in the next presented rule. The second one, $Rule_{GenerateSignal}$ is used to generate signal in ground g. Here, we use R_{Turn} to stop the generation of signals at cycle $R_{FinalTurn}$. In this way, we can limit the number of generated signals during a simulation.

$$Rule_{Turn} = \emptyset \rightarrow \{Ground.R(R_{Turn}) \mathrel{+}= 1\}$$

$$Rule_{GenerateSignal} = \{Ground.R(R_{Turn}) < Ground.R(R_{FinalTurn})\} \rightarrow \{Ground.R(Signal) \mathrel{+}= Ground.R(R_{GenerationRate})\}$$

4.7 Conclusion

We present a new generic SoS model and instantiate it to the Missouri Toy Problem. First, we verify that SApHESIA model is adequate in regards of evaluation criteria we defined in chapter 1 (table 4.1). From a SoS literature review, we highlight a lack on existing models concerning the ability to model the dynamics of the environment, the interactions between component systems and non-trivial problems. To fulfill these lacks, we extended the notion of environment enabling to model more detailed and dynamic problems through the concept of entities and rules. Then, we improve the model of interactions between component systems through oriented links enabling interactions (by exchanges of resources) and the strength enabling to model their intensity. Moreover, our model is more expressive than existing one because of the creation of the condition and effects of functionalities that enable to model, for example, resource flows or object creations and a more detailed environment description. Furthermore, the model enables the creation of metrics through resources, goals and the notion of cost. Metrics will be used in our experimentation to evaluate the architecture solution given by our formation heuristic proposed in chapter 7. Finally, to check that our model is able to model SoS, we propose to model a well-known SoS problem which is the Missouri Toy Problem. To solve SoS problem, we will propose in the next chapter a generic algorithm for using cooperation between component systems in a SoS.

5 Cooperative SoS Architecting Heuristic Based on Criticality

This chapter presents a cooperative SoS Architecting Heuristic based on criticality. To provide this decentralized and generic SoS Architecting Heuristic, we propose a decentralized decision algorithm based on the Adaptive Multi-Agent System (AMAS) approach that uses cooperation as a social behavior between agents [Capera et al., 2003]. This approach enables to develop complex systems where the global function emerges from the self-organization of parts of the system. To drive self-organization, agents use the concept of cooperation (see chapter 3) between each other and their environment. This chapter will present a way to compute this cooperation through the concept of criticality as well as an algorithm of cooperation for component system in a SoS.

5.1 Through a totally Decentralized SoS Behavior

Centralized heuristics are useful when architecting directed and acknowledged SoS: the SoS entity is used to negotiate the services that the component systems can provide to the SoS. But the use of the SoS entity disables the architecting of collaborative and virtual SoS because they do not own a centralized entity (see chapter 1 for more details). Then, centralized SoS architecting heuristics such as Agent Based Wave and Collaborative Model (presented in chapter 2) cannot be used for all types of SoS. Moreover, we show that they cannot cope with a dynamic environment and fails to respect the open interface principle (i.e., add/remove component systems during functioning). Then, we will propose an decentralized architecting heuristic where the SoS will have the ability to cope with a dynamic environment, respecting the open principle interface and enabling to architect collaborative and virtual SoS without the help of the SoS entity.

5.2 The Criticality: Metric of Cooperation

The criticality is a concept used in the AMAS approach; an agent helps the more critical agents of its neighborhood (including itself). A first definition of the criticality is given by [Lemouzy, 2011]: *The criticality of an AMAS agent is a metric representing the distance between its current state and the final state it tries to reach.*

The term *distance* has been chosen because it is generic enough to enable the designer to instantiate it depending on his problem. Indeed, the designer can choose a temporal, spatial or even logic distance. Nevertheless, this one is too generic: when AMAS designers create their own metric of criticality for their systems, these ones are problem-dependents and then are not reusable for other AMAS they develop. To fill these gaps, this section presents works on formalization of the criticality. Our first work on criticality exclusively focuses on the AMAS approach (but will be reused in our SoS architecting heuristic). Then, a formal definition of the criticality is presented and its instantiation through a simple example of resource transportation system is addressed.

5.2.1 Formal Definition of Criticality

To formalize the notion of criticality, we propose to formally define the two following sentences: (1) *the current state of an agent* and (2) *the final state it tries to reach*.

As previously said, an AMAS agent pursues a $Perception - Decision - Action$ cycle. During its perception cycle, the agent updates its perception on itself (i.e.; its internal state) and its environment. We propose to define the current state S_a of an agent a (1) as *the result of its perception phase*. The perceptions of an agent are obtained thanks to a set of sensors given by the AMAS designer. We propose to define the output of each of these sensors as a function $p_i(t)$ ($i \in \mathbb{N}$). As an example, a sensor can be a temperature sensor where the output function will be the numerical value of the temperature given by the sensor. We propose to formally define the current state $S_a(t)$ at time t of an agent as the set of its perceptions (containing the set of perceptions of its internal state) at time t :

$$S_a(t) = P(t)$$

where $P(t) = \{p_1(t), ..., p_n(t)\}$ is the entire set of perceptions of the agent a at time t.

The final state an agent tries to reach (2) relies on the notion of objective or goal. Then, to be coherent with (1) we propose to define *the final state it tries to reach* as a particular state S_{goal} the agent a tries to reach concerning a subset of its perceptions:

$$S_{goal} = \{p_{1_{goal}}, ..., p_{m_{goal}}\}$$

where $\{p_{1_{goal}}, ..., p_{m_{goal}}\}$ concerns a subset of P (i.e., S_{goal} does not concern all the perceptions of the agent).

From these definitions, we presented in [Bouziat et al., 2014] a first generic and computable metric of criticality of an agent i defined as:

$$C_i(t) = F(p_1(t), ..., p_m(t), S_{goal})$$

Where:

- $S_{goal} = \{p_{1_{goal}}, ..., p_{m_{goal}}\}$ is the final state the agent i tries to reach.
- $p_1(t), ..., p_m(t)$ is a subset of the current perceptions of the agent i;

▷ F is a mathematical function chosen by the designer of the AMAS.

Then, criticality of an agent is defined as a function taking a subset of perceptions and its goals concerning these perceptions as inputs. Then, the work of the designer is to define F as well as to choose rigorously the perceptions $p_1(t), ..., p_m(t)$ and the associated goals to compute the criticality.

But, this metric is not useful if it is used alone. Indeed, the criticality is an instant metric that only gives the level of difficulty of the agent at a time t. But according to the AMAS approach, to be cooperative, an agent must choose the action that helps the most critical agent of its neighborhood (including itself) without making another agent more critical than the most critical. In term of criticality, this sentence can be translated as follow: to be cooperative means that the agent must choose the action that reduces the criticality of the agent with the highest criticality without increasing another criticality above this one. To do that, an agent has to know the result of each of its action in term of criticality. That is why, we propose the notion of *anticipated criticality* to address this purpose.

5.2.2 One Step Further: The Anticipated Criticality

To be compliant with the AMAS theory, an agent action cannot lead intentionally to a future Non-Cooperative Situation (NCS) with its neighborhood. Before performing an action, the agent will ask to agents around it the effects (regarding to their criticality) of an action a. That is why the agents compute the anticipated criticality. Formally, the anticipated criticality for an action a represents the next criticality an agent will have knowing the effect of its next action a. It enables the agent to predict the effect of the action a on its own criticality. Formally, the anticipated criticality of an agent i for an action a is defined as:

$$CA_i(t, a) = C_i(t) + Eff(a)$$

with $C_i(t)$ the criticality of a at time t and $Eff(a)$ a function giving the effect in term of criticality for the action a. Finally, the anticipated criticality can be defined for a sequence of actions $A = \{a_1, a_2, ..., a_n\}$ as :

$$CA_i(t, A) = C_i(t) + \Sigma_{i \in [\![1,n]\!]} Eff(a_i)$$

5.2.3 Example of Criticality Instantiation

To give a concrete example of the concept of criticality, we present here a resource transportation system that is a simplified version of CoCaRo described in the chapter 8. We only introduce here the main elements of the system to give an understandable example of the use of the criticality. The overall objective of the system is to transport resources dispersed through the environment towards a main area called a *nest*. This system is composed of a finite group of robots (that are agents) that can do the following actions :

▷ to move randomly on the environment;

▷ to deposit a resource on the environment;

- to take a resource from the environment;
- to go to a particular point on the environment.

Moreover, each robot has an initial amount of energy (equals to Max_{N_e}) consumed at each movement implying that each robot has a finite life time. However, each time a robot drops a resource in the nest, it refills its energy level. This refill can be seen as a reward in the form of energy allowing the robot to remain longer active. The defined perceptions of each robot are the following:

- to detect a box around it;
- to detect other agents around it;
- to know its level of energy.

To make the robots surviving as long as possible, it seems natural to choose the following objective for the robots: *"Bring back as much as possible resources to the nest"*. To fulfill this objective, the robot owns a useful perception that is its own level of energy. Indeed, if its level of energy is high, then it can move in the environment and find a lot of resources. On the contrary, if the level of energy is low then it has strong chances to run out of energy and to become not useful anymore. Then, the local objective of a robot which is to bring back as much as resources as possible can be translated to *"Maximize its level of energy"*. The criticality C_{r_i} of a robot agent r_i is defined as:

$$C_i(t) = Max_{Ne} - Ne_{r_i}(t)$$

With:

- $Ne_{r_i}(t)$ the perception of energy level of the robot r_i at time t;
- Max_{Ne}, the possible maximum level of energy (i.e., the constant related to the goal of maximizing its energy level).

Concerning the anticipated criticality, the effects of the different actions of r_i is calculated as follow.

$$Eff_i(a) = \begin{cases} conso & \text{if } a = move \\ conso \times dist(b_j, r_i) & \text{if } a = go(b_j) \\ -reward & \text{if } a = deposit(b_j) \\ 0 & \text{if } a = pickup(b_j) \end{cases}$$

Where:

- *conso* the energy level units consumed during one time unit;
- *move* is the action to move randomly on the environment;
- $dist(b_j, r_i) \in \mathbb{R}$ the distance between resource b_j and robot i;

- $go(b_j)$ is the action of moving to resource b_j;
- $deposit(b_j)$ is the action of dropping the resource b_j in the nest.

By noting the sequence of actions of bringing back a resource b_j to the nest $bback(b_j) = \{go(b_j), pickup(b_j), go(nest), deposit(b_j)\}$ The anticipated criticality of $bback(b_j)$ is calculated as follow:

$$CA_i(t, bback(b_j)) = C_i(t) + Eff_i(go(b_j)) + Eff_i(pickup(b_j)) + Eff_i(go(nest)) + Eff_i(deposit(b_j))$$

Finally, figure 5.1 sums up how the criticality and the anticipated criticality can be used during the *Decision* step of the agent. In this example, robot r_1 can choose between two resources (b and b') to be rewarded. The anticipated criticality enables to know that bringing back b is better for it because

$$CA_1(t, bback(b)) < CA_1(t, bback(b'))$$

Indeed, the total path to pick up b and bring it back to the nest is less long than b' ($d_1 + d_b < d'_1 + db'$).

Figure 5.1 shows also how the criticality and the anticipated criticality enable cooperative decision: if robot r_1 perceives a robot r_2, the idea is to communicate criticality and anticipated criticality for the detected resources. If the two robots are in conflict on the same resource, then the two robots have to choose the sequence of actions that will minimize in priority the highest criticality without making another criticality higher than the highest one. In this way, the behaviors of robots are totally cooperatives because *it enables the most critical agent to fulfill its goals in priority*. In this example, the anticipated criticalities for bringing back b and b' are given by the numerical application of $CA_i(t, bback(b_j))$ (with $reward = 100$ for this example):

- $CA_1(t, bback(b)) = C_1 + d_1 + d_b - reward = 20$
- $CA_1(t, bback(b')) = C_1 + d'_1 + d_{b'} - reward = 50$
- $CA_2(t, bback(b)) = C_2 + d_2 + d_{b'} - reward = 10$
- $CA_2(t, bback(b')) = C_2 + d'_2 + d_{b'} - reward = 70$

Then, even if r_1 is the most critical and b is better for it, it will choose b' because if r_2 brings back b', criticality of r_2 will be higher than the one of r_1. For more details, the complete cooperation algorithms and simulations are presented in the chapter 10.

5.3 Instantiation of Criticality for SApHESIA

In this section, our objective is to re-use the concept of criticality in SApHESIA to propose a generic cooperative decision algorithm for architecting SoS. In this aim, we propose a generic formulation of the concept of criticality for a component system based on

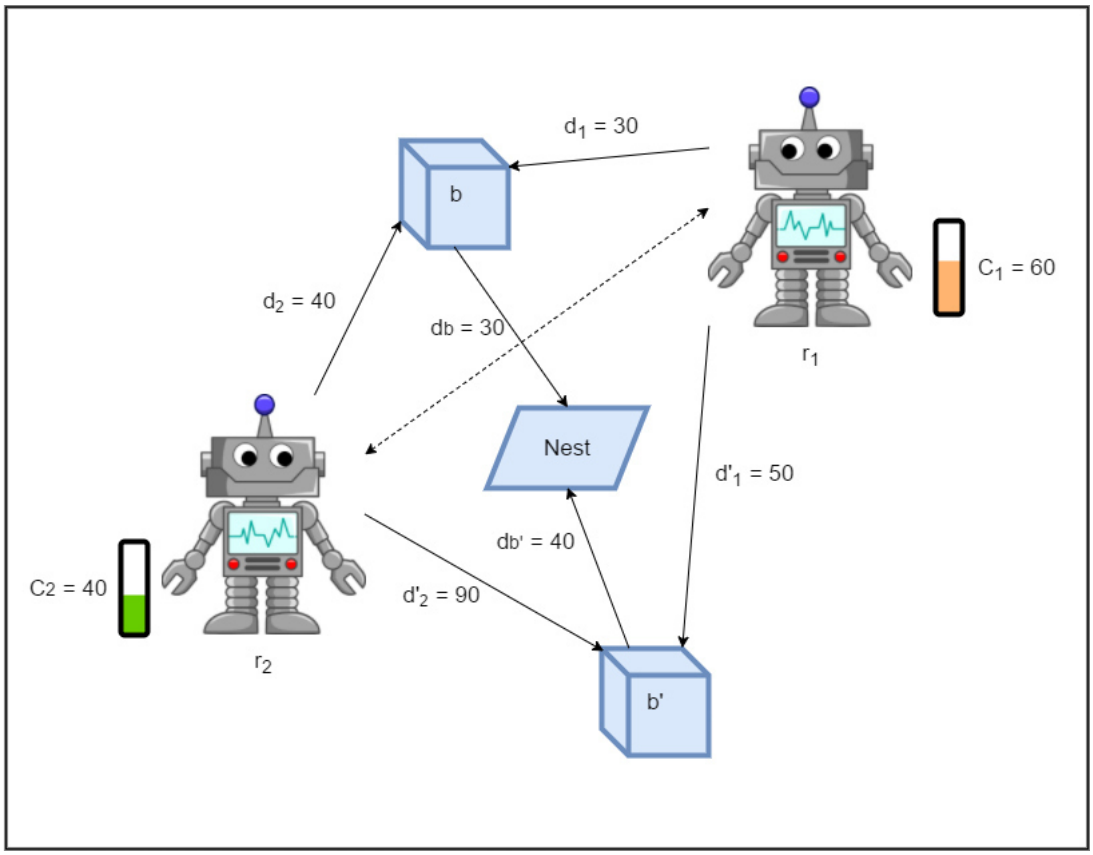


Figure 5.1 – Agent cooperative decision through criticality

the SApHESIA model. Then, we propose a decision algorithm (that we have proposed in [Bouziat et al., 2016] and extended in [Bouziat et al., 2017]) based on this criticality metric to enable cooperation between component systems.

We propose to integrate the concept of criticality used in the AMAS approach within the SApHESIA model. To do that, we define the criticality according to the resources and goals of a component system. To represent the **current state** of a component system, we use the perceptions it has on its current resources. To represent **the state it tries to reach**, we propose to use its goals. In this way, the criticality for a component system is totally compatible with the criticality definition we gave in section 5.2.1. We remind that the component system S is defined as a set of sets such as (see chapter 6 for more details): $S = \{T, R, Acq, L, F, G, Cost\}$ and a goal $g \in G$ is defined as a particular level of resources it tries to reach:

$$g = \{Re \ \mathtt{Op} \ Qu, Pr\}$$

where:

- Re is a type of resource;
- $Qu \in \mathbb{R}$ is the quantity of resource of type Re the component system wants to own;
- $\mathtt{Op} \in \{=, \neq, <, >, \leqslant, \geqslant\}$ is the comparison operator in order to compare Qu to the current quantity of Re;

- $Pr \in \mathbb{N}^+$ is the priority of the goal enabling to model the relative importance of a goal. The higher the priority is, the more important the goal is.

Then, we define a criticality for each goal g contained in the set of goals G for the component system S. This criticality has to represent the distance of fulfillment of this goal g. For example, if the goal g concerning the resource Re is to reach a given quantity Qu ($g.\texttt{Op} \in \{=\}$), then the criticality of g has to be high if Re is 'far' from the objective and low otherwise. In the same way, if the goal g concerns a resource Re that has to be greater than a given threshold quantity Qu ($g.\texttt{Op} \in \{>, =>\}$), then the criticality of g has to be high if the resource is smaller than the threshold and low if the resource is greater than the threshold.

To represent these different cases, we propose to use the following functions for the calculation of the criticality of the goal g at time t $C_g(t)$:

$$C_g(t) = \begin{cases} (1) : 1 - \dfrac{1}{e^{\alpha \times \Delta_g(t)}} \\ (2) : \dfrac{1}{e^{\alpha \times \Delta_g(t)}} \\ (3) : 1 - \dfrac{atan((\alpha \times \Delta_g(t)) + \dfrac{\Pi}{2}}{\Pi} \\ (4) : \dfrac{atan((\alpha \times \Delta_g(t)) + \dfrac{\Pi}{2}}{\Pi} \end{cases}$$

(1): if $g.Op \in \{=\}$: the goal g is to reach a given quantity Qu of resource Re;
(2): if $g.Op \in \{\neq\}$: the goal g is to be as far as possible from a given quantity Qu for the resource Re;
(3): if $g.Op \in \{<, =<\}$: the goal g is to be inferior or equal to a given quantity Qu for the resource Re;
(4): if $g.Op \in \{>, =>\}$ the goal g is to be superior or equal to a given quantity Qu for the resource Re .
With :

- $\Delta_g(t) = g.Qu - S.R(g.Re)(t)$ is the difference between the given quantity Qu and the current amount of resource Re of S;

- $S.R(g.Re)(t)$ is the current amount of resource Re of component system S at time t;

- α is a coefficient value that influences the shape of the curve.

These sigmoid functions have been chosen because their shapes fit with the meaning of each goal. Examples of these functions are given in the annex 2. Moreover, one issue of [Lemouzy, 2011] was that the criticality of its different systems was not calculated in the same norm: a first AMAS could have its criticality in an interval $I_1 = [\beta, \gamma]$ and another one in $I_2 = [\beta', \gamma']$. Then [Lemouzy, 2011] proposes, without developing it, to do a translation of criticality values of I_1 from I_2. To simplify and get out of this problematic, we choose functions that give numerical values in the interval $[0, 1]$.

To calculate the criticality $C_S(t)$ of a component system S from the criticalities of its goals G (given by each $C_g(t)$, with $g \in G$) , we calculate the weighted average of all its goals criticalities:

$$C_S(t) = \frac{\sum_{g \in G}(C_g(t) \times g.Pr)}{\sum_{g \in G}(g.Pr)}$$

With $g.Pr$, the priority of goal g.

Finally, this formula of the criticality for component systems is adequate with the first definition of the criticality (given in section 5.2): we compare the current state of a component system (represented by its resources) with the state it wants to reach (represented by its goals). This calculation is central in the cooperative decision algorithm presented in the next section.

5.4 Cooperative Decision Algorithm for Component Systems (of a SoS)

We present here a decision algorithm for component systems based on criticality and anticipated criticality comparisons between component systems. Here, a component system is modeled as an autonomous entity that pursues a $Perceive - Decide - Act$ cycle. This decision algorithm is incorporated in the *Decision* phase of the component system and enables it to choose the most cooperative action (here a functionality).

Algorithme 5.1 : Cooperative component system S_i decision.

```
1  CoopTable As Dictionnary < Functionnality, Dictionnary < ComponentSystem, List <
   Float >>> ;
2  forall f ∈ F_i do
3  |  forall S_j ∈ L_i ∪ Acq_i do
4  |  |  CoopTable(f)(S_j) = CoopTable(f)(S_j) ∪ askAnticipatedCrit(S_j, f) ;
5  |  |  forall S_k ∈ (L_i ∪ Acq_i) do
6  |  |  |  CoopTable(f)(S_j) = CoopTable(f)(S_j) ∪ askAnticipatedCrit(S_j, S_k, f) ;
7  |  |  end
8  |  |  Sort CoopTable(f)(S_j);
9  |  end
10 end
11 return minmaxFunc(CoopTable) *Choose f that minimize the max of criticality*;
```

Basically, each component system $S_i = \{T_i, R_i, Acq_i, L_i, F_i, G_i, Cost_i\}$ does two important actions: construct the cooperative table and choose the most cooperative functionality. These two actions are presented in the two next paragraphs.

Construction of the cooperative table: The aim of this procedure is to construct a table where lines contains the anticipated criticalities for all the neighborhood of S_i (given by $L_i \cup Acq_i$) depending on the potential execution of each functionality f of S_i. For example, the first line of the table 5.1 contains all the anticipated criticalities of the neighborhoods of S_i if this one apply its functionality F_1 on S_1. The second line contains also all the anticipated criticalities its neighborhoods but if it apply F_1 on S_2 and so on.

Then, each S_i asks for the anticipated criticality of its neighborhood (in Acq_i and L_i) for each functionality $f \in F_i$ of its available functionalities. This request is made by the function $askAnticipatedCrit(S_j, f)$ (line 4 of algorithm 5.1). This function takes a component system S_j and a functionality f and returns the value of the anticipated criticality of S_j if S_i apply f on S_j. Then, this anticipated criticality is saved in a new line (identified by a couple (S_j, f)) of the cooperative table ($CoopTable$ in the algorithm 5.1). Then, the anticipated criticalities of other component systems (including S_i) are added to this line through the function $askAnticipatedCrit(S_j, S_k, f)$ that returns the anticipated criticality of S_k if S_i apply f on S_j (line 6 of algorithm 5.1). Then, the lines contain the anticipated criticalities of its entire neighborhood (including S_i) if S_i apply f on S_j. Finally, all the lines are sorted from the highest criticalities to the lower one (line 8 of algorithm 5.1).

This table is implemented through a dictionary. The term dictionary has to be understood as an abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection. Then, this table is implemented through a dictionary where the keys are the functionalities and the values are also dictionaries where the keys are component systems and the values are lists of floats $Dictionnary < Functionnality, Dictionnary < ComponentSystem, List < Float >>>$ in algorithm 5.1).

Choose the most cooperative functionality: In the AMAS approach, when an agent has several choices concerning its next action, it will choose the one that minimizes the maximum of the anticipated criticalities. We propose to use the same approach here. Then, once the cooperative table is built, S_i chooses the functionality that is the most cooperative in terms of criticality: the one that minimizes the maximum of the anticipated criticalities thanks to the $minmaxFunc(CoopTable)$ function that return the function f to apply. The details of $minmaxFunc(CoopTable)$ is given by the algorithm 5.2 and is explain more in details here through the table 5.1:

Table 5.1 presents the anticipated criticality of the neighborhood of a component system S_i when applying the available functionalities F_1 on the component system S_1. Knowing the lists of all expected criticalities for all of these functionalities, S_i will compare the first float of each line (given by $CoopTable(f, s)(currentLevel)$) to choose the functionalities that lead to the **minimum of the maximum** of the criticality. Indeed, the first float of each line (i.e., the first column of the cooperative table) is the maximum of each line because the table is already sorted. In the algorithm 5.2, this operation is made from line 8 to 22. If several lines give the same minimum, then the variable $currentLevel$ is incremented and the second float of each line also given by $CoopTable(f, s)(currentLevel)$ (i.e., the second column of the cooperative table) is compared and so on.

		Anticipated criticalities of S_i neighborhood			
F_1 applied on	S_1	0.85	0.47	0.11	0.05
	S_2	0.95	0.28	0.09	0.08
	S_3	0.71	0.41	0.31	0.29
F_2 applied on	S_1	0.99	0.80	0.74	0.65
	S_2	0.71	0.41	0.35	0.08
	S_3	0.77	0.75	0.33	0.15
F_3 applied on	S_1	0.71	0.47	0.11	0.05
	S_2	0.71	0.41	0.31	0.22
	S_3	0.95	0.58	0.47	0.31

Table 5.1 – Example of sorted anticipated criticalities table for a component system S_i

Detailed example with the table 5.1: In table 5.1, the functionalities that enable to choose the minimum of the maximum of criticalities are F_1 on S_3 , F_2 on S_2 and F_3 on S_2 (because the minimums of the maximums of criticalities are 0.71). Here, three functionalities lead to the same minimum of maximum (0.71), then the second elements of the lists (corresponding to the second column of the cooperation table because the lines are sorted) is compared. In this example, this second comparison does not eliminate any functionality (because the three have the second criticality equals to 0.41). But the third comparison eliminates F_2 on S_2 (because F_2 on S_2 gives a third criticality of 0.35 and the two others 0.31). This algorithm is repeated until only one functionality is remaining. If several functionalities are remaining, a random choice is done because they are equivalent. Finally in this example, the last is F_3 on S_2 and will be the choice returned by $minmaxFunc(CoopTable)$. This behavior leads to minimize the maximum of neighborhood criticality. Indeed, each component system tends to "help" his neighborhood by choosing the functionality that, in the worst case, causes the minimum raise of criticality.

Let's take a SoS $SoS = \{S_1, ..., S_n\}$ where:

- $\forall i \in n, S_i = \{T_i, R_i, Acq_i, L_i, F_i, G_i, Cost_i\}$
- $\forall i \in n, C_{S_i}(t)$ is the criticality of S_i at time t
- $currentLevel$ is the column index of the $CoopTable$ that is being compared;
- $minF, minS$ the functionality and the component system giving the minimum of criticality.

5.5 Complexity Analysis of the Decision Algorithm

We propose to analysis the complexity of the proposed algorithms 5.1 and 5.2 to validate the effectiveness in time execution of our approach.

Algorithme 5.2 : minmax function applied on *CoopTable*

```
1  /*Find f and S_i containing the min of the max of expected criticalities*/
2  (Functionality, ComponentSystem) minmaxFunc(CoopTable As
   Dictionnary<Functionnality, Dictionnary<ComponentSystem, List<Float»>):
3  currentLevel ⟵ 0 ;
4  minF, minS ⟵ ∅ ;
5  while currentLevel < Length(CoopTable)) do
6  |  minCrit ⟵ 1 ;
7  |  minF ⟵ ∅ ;
8  |  for f ∈ CoopTable do
9  |  |  minS ⟵ ∅ ;
10 |  |  for S ∈ CoopTable(f) do
11 |  |  |  current ⟵ CoopTable(f, s)(currentLevel) ;
12 |  |  |  if current < minCrit then
13 |  |  |  |  current ⟵ minCrit ;
14 |  |  |  |  CoopTable(f).remove(minS) ;
15 |  |  |  |  minF ⟵ f ;
16 |  |  |  |  minS ⟵ s ;
17 |  |  |  end
18 |  |  |  else if current > minCrit then
19 |  |  |  |  CoopTable(f).remove(s) ;
20 |  |  |  end
21 |  |  end
22 |  end
23 |  currentLevel + +
24 end
25 return (minF, minS) ;
```

5.5.1 Complexity analysis of the algorithm 5.1

Let S_i a component system such as $S_i = \{T_i, R_i, Acq_i, L_i, F_i, G_i, Cost_i\}$. The size of a set A is noted $|A|$. The complexity of this algorithm depends of the three for loops in line 2, 3 and 5 where the number of calls are respectively $|F_i|$, $|L_i \cup Acq_i| \leqslant |L_i| + |Acq_i|$ and $|L_i \cup Acq_i|$. During the simulation, S_i does not have new functionalities, then $|F_i|$ is static. As previously said, a component system can have functionality to create link with other component systems, then $|L_i|$ can vary during the simulation. Finally, the acquaintances are updated through the perception phase of the component system (described in section 6.3.1) but this set is bounded. Let B_{Acq} the static upper bound of $|Acq|$. Then, we propose to analyze the complexity of this algorithm with the variation of $|L_i|$. Let c_i the execution time of line i and $C_{5.1}(|L_i|)$ the complexity in time of 5.1 in function of $|L_i|$:

$C_{5.1}(|L_i|) = c_2 \times |F_i| + |F_i| \times (c_3 \times (B_{Acq} + |L_i|) + (B_{Acq} + |L_i|) \times (c_4 + c_5 \times (B_{Acq} + |L_i|) + (B_{Acq} + |L_i|) \times c_6 + c_8)) + c_{11}$

With:

- $c_2 = c_4 = c_5 = c_6 = \mathcal{O}(1)$ because are simple operations;
- c_8 is a sorting of a line of the *CoopTable* and as previously said, a line contains the anticipated criticalities for all the neighborhood (given by $L_i \cup Acq_i$). Finally our sorting function is implemented in JAVA that specifies that the sorting operation is implemented as a modified mergesort, an algorithm offering a guaranteed $n \times log(n)$ performance. Then $c_8 = \mathcal{O}((|L_i| + B_{Acq}) \times log(|L_i| + B_{Acq})) = \mathcal{O}(|L_i| \times log(|L_i|))$;
- line 11 is a call to the algorithm 5.2, then c_{11} is equal to the complexity of the algorithm 5.2 presented hereafter.

By developing and replacing the different c_i, the highest term is given by the multiplication of c_8 with $(|L_i| + B_{Acq})$. Then, excepting the complexity of c_{11} that is computed hereafter, the highest term gives a complexity of $\mathcal{O}(|L_i|^2 \times log(|L_i|))$ for algorithm 5.1.

5.5.2 Complexity analysis of the algorithm 5.2

Let S a component system such as $S_i = \{T_i, R_i, Acq_i, L_i, F_i, G_i, Cost_i\}$. The complexity of this algorithm depends of the while loop of line 5 and two for loops in line 8 and 10 where the number of calls are respectively $|L_i| + |Acq_i|$, $|F_i|$ and $|L_i| + |Acq_i|$.

Then , the complexity in time of the algorithm 5.2 is given by:
$C_{5.2}(|L_i|) = c_3 + c_4 + c_5 \times (|L_i| + B_{Acq}) + (|L_i| + B_{Acq}) \times (c_6 + c_7 + |F_i| \times c_8 + |F_i| \times (c_9 + |F_i| \times c_{10} + |L_i| \times (c_{11} + C_{12} + c_{13} + c_{14} + c_{15} + c_{16} + c_{18} + c_{19})) + c_{21}$

- c_{14} and c_{19} are guaranteed with a performance of $\mathcal{O}(1)$ because these lines concern removing operations on a Dictionary;
- the remaining lines have a complexity of $\mathcal{O}(1)$ because they are simple operations.

Finally, the development of $C_{5.2}(|L_i|)$ gives as a highest term of degree 2, then $C_{5.2}(|L_i|) = \mathcal{O}(|L_i|^2)$. Then, in algorithm 5.1, $c_{11} = \mathcal{O}(|L_i|^2)$. Then, by replacing this complexity in $C_{5.1}(|L_i|)$, we conclude that $C_{5.1}(|L_i|) = \mathcal{O}(|L_i|^2 \times log(|L_i|))$.

Then, our the complexity analysis of our approach shows a quasi-quadratic complexity with the number of links for each component systems. Then, to guarantee a constant time execution at the component system level, we propose to limit the number of links given by $|L_i|$ for a component system S_i. To do that, we propose to add a resource $R_{LinkResource} \in R_i$ that :

- decreases when S_i creates a link with another component system;
- increases when S_i unlinks with a component system;
- is used as a condition of functionality of link creation in the following way:

$$F_{LinkSyS} = \{\{SyS \in Acq\} \rightarrow \{Link, 1\}, \{SyS.R(R_{LinkResource}) \mathrel{+}= -1\}, t_{Link}, p_{Link}\}$$

In this way, if the component system is defined with $R_{LinkResource} = 4$, the maximum number of links of the component system is 4. Then, the time execution of its cooperative algorithm is bounded, guarantying a linear complexity of our approach with the number of component systems.

5.6 Conclusion

We propose a formal definition and a calculation of the criticality concept used in the AMAS approach. Then, after proposing an example of a transportation system, we propose a cooperative decision algorithm for component systems based on criticality comparisons. First, this heuristic is fully decentralized because decisions are only made by the component systems. Then it enables to get ride off a centralized management for finding solutions and model collaborative and virtual SoS. Using cooperation, this heuristic ensures self-organization because of the theorem of functional adequacy (see chapter 4). By self-organizing, this heuristic enables to architect SoS that self-adapt and where the adequate functionality emerges from the parts. Then, our heuristic is open (component systems can be added/removed during functioning from the SoS). It is able to cope with a dynamic environment and is robust to openness. Furthermore, based on our generic model of SoS, this heuristic can be used on any kind of SoS problems (military, economics, science and so on). Concerning computationality, the model as well as our algorithm can be can be easily instantiated because it is based on a documented generic model and on precise and explained algorithms. Then, the limited perception of a component system (through the upper bound of the set of acquaintances) and the limitation of the number of links (thanks to $R_{LinkResource}$) guarantee a linear complexity with the number of component systems. Finally, our heuristic fulfills our evaluations criteria concerning the evaluation of SoS architecting heuristics (see section 2.2). Chapters 7, 8, 9 and 10 propose experimentations to validate these points.

6 SApHESIA tools for SoS Architecting

The objective of this chapter is to present the tools that enable to run SoS architecting experiments based on the cooperative decision algorithm for component systems presented in the previous chapter. To make SoS architecting tests, we first searched existing platforms that are able to model and run scenarios for SoS architecting. We conclude that they do not enable to implement easily a new SoS language as SApHESIA Modeling Language (SML). We decided to develop SApHESIA tools, a set of tools enabling to run experiment of SoS architecting. This chapter presents in details the functioning of SApHESIA and its main functionalities. This platform has been used for implementing the whole of our experiments on SoS Architecting.

6.1 General Presentation

The SApHESIA tools consists of 2 components: **SoS and Environment generator** and the **core architecting**. The main objective of the generator is to transforms the SML files to binaries that are usable for the core architecting. Figure 6.1 shows that the generator is composed of the **SML parser and compiler** and the **GUI generator**. The main objective of the core architecting is to run and show the results of the experiment. Figure 6.1 shows that this one is composed of two main components: the **SApHESIA architecting engine** and the **SApHESIA viewer**. The next sections describe more in details these two components.

6.2 SoS and Environment Generator

The main objective of the **generator** (the blue part of figure 6.1) is to generate the binaries for the tests. The inputs of this component are the description of the SoS and its environment. This description is given to the **SML Parser & Compiler** through the **GUI generator** that enables the user to manually create and/or modify a SoS and/or its environment. Figure 6.2 shows the main view of the SApHESIA tools enabling the modification of component systems and figure 6.4 shows the main view of a functionality creation. Moreover, it enables to add or remove component systems and entities during the experiment. All the different properties of a component system can be changed (Resources, Links, Goals and so on). In

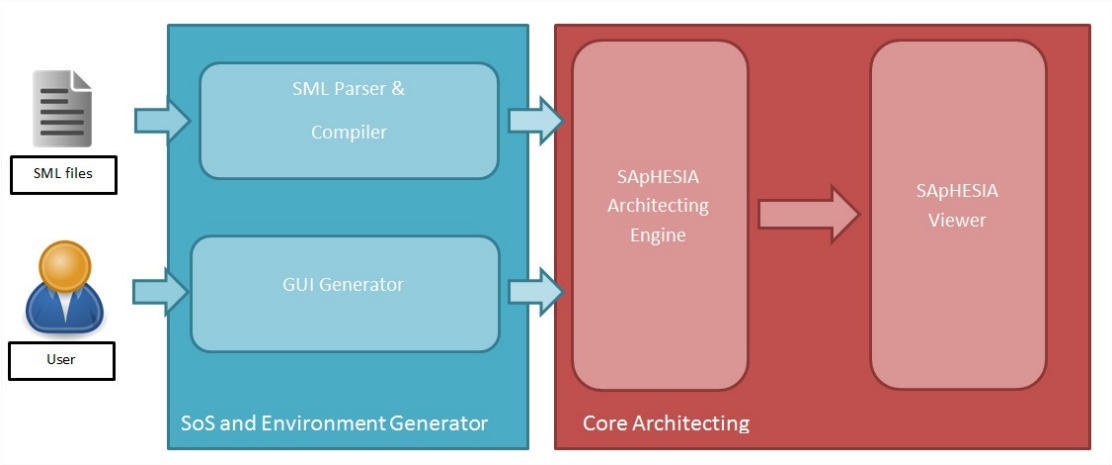


Figure 6.1 – Main components of SApHESIA tools

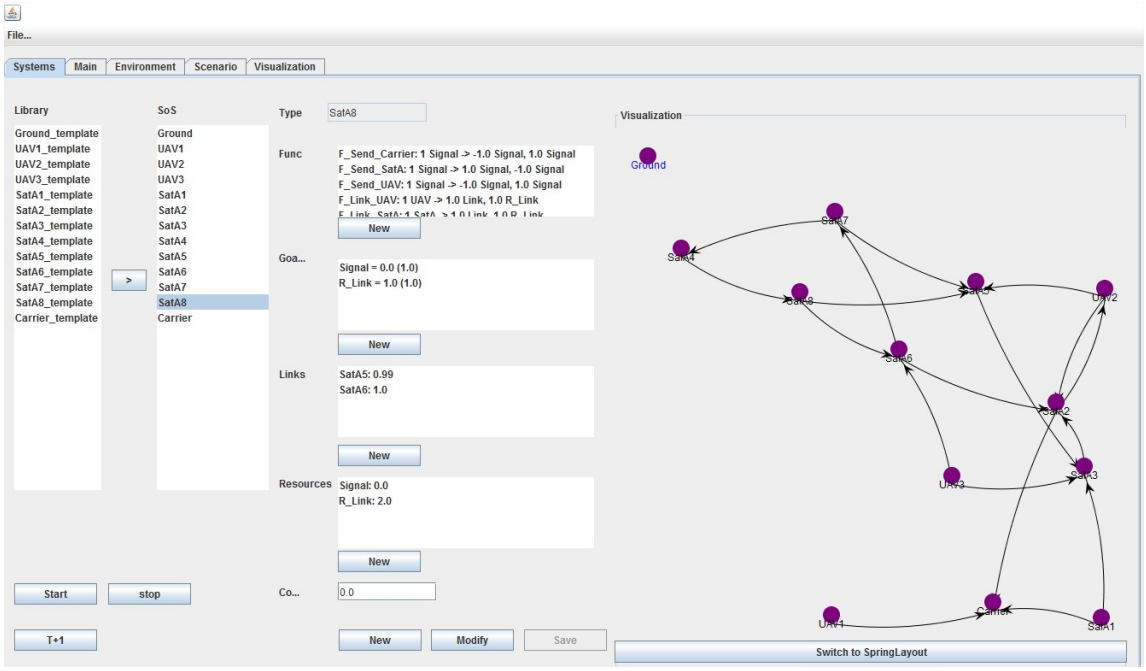


Figure 6.2 – Main view of SApHESIA tools

this way, the designer can see in at runtime what the effects of his changes on the SoS are. Nevertheless, this GUI generator can be fastidious for creating from scratch a huge SoS, that is why the designer has the possibility to use the **SML Parser & Compiler** through SML files containing a predefined SoS and environment. The result of the **SML Parser & Compiler** is a fully loaded model ready to test. Once the model is loaded, the experiment is ready to run by the **core architecting** (the red part of figure 6.1).

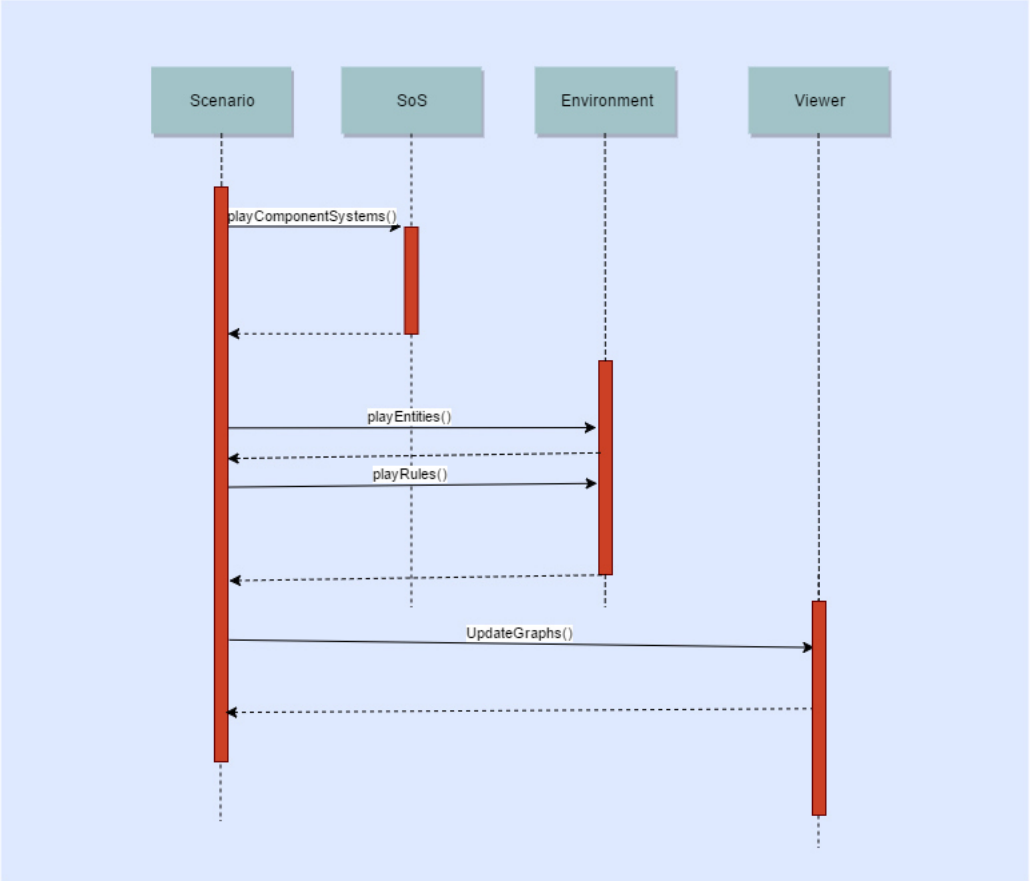


Figure 6.3 – High level sequence diagram of SApHESIA Engine

6.3 SApHESIA Core Architecting

The main functionality of the core architecting of SApHESIA tools is to run step by step the test defined by the SoS and environment generator. This one is composed of the SApHESIA Engine and Viewer which respectively runs a loaded SApHESIA model and enables to visualize results of the tests.

6.3.1 SApHESIA Engine

SApHESIA Engine is the core of our tools. This component runs the experiment by using the components (SoS and environment) loaded with the SML Parser & Compiler. The sequence diagram 6.3 (presenting one experiment cycle) shows that the engine is composed of several modules detailed hereafter: the scenario, the SoS and the Environment modules.

a) Scenario Module

The scenario module is the main module of the SApHESIA engine. It enables to run the SoS and the environment modules. Figure 6.3 shows the main sequence diagram of the SApH-

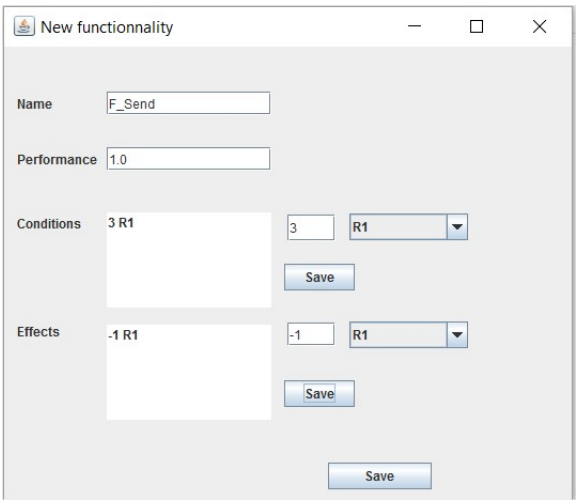


Figure 6.4 – Example of a functionality creation

ESIA engine: at each step of the experiment, the SoS module is called to run the component systems of its SoS. Then, the environment module is called to run the entities and the rules. Finally, viewer module is called to update information the user wants to display (level of resource of a component system and so on).

b) SoS Module

The SoS module contains the component system submodules. At the beginning of the experiment, there is one component system module creation by component system in the SoS. The component system module simulates the behavior of a component system in the SoS. A component system module uses a $Perception - Decision - Action$ cycle. The next paragraphs detail theses three phases. Figure 6.5 details the behavior of a component system during experiment.

Perception: The perception phase of a component system $S = \{T, R, Acq, L, F, G, Cost\}$ enables to update its acquaintances set Acq. More precisely, S adds new component systems in its acquaintances and remove some others. In this way, this phase enables to simulate a component system with changing perceptions. Initially, Acq is empty for all the component systems. At each turn of the experiment, S detects new component systems or entities. The maximum number of new component systems detected can be set at the beginning of the experiment in the SML file with the tag $< AddPerception >$. For example, for a maximum number of 6 component systems detected, the following text has to be added to the component system declaration:$< AddPerception > 6 < /AddPerception >$. Once a component system or an entity is detected, this one is put in the Acq set of the component system. In the same manner, a number of component systems in Acq will be removed. This number is set at the beginning of the experiment in the SML file with the tag $< removePerception >$.

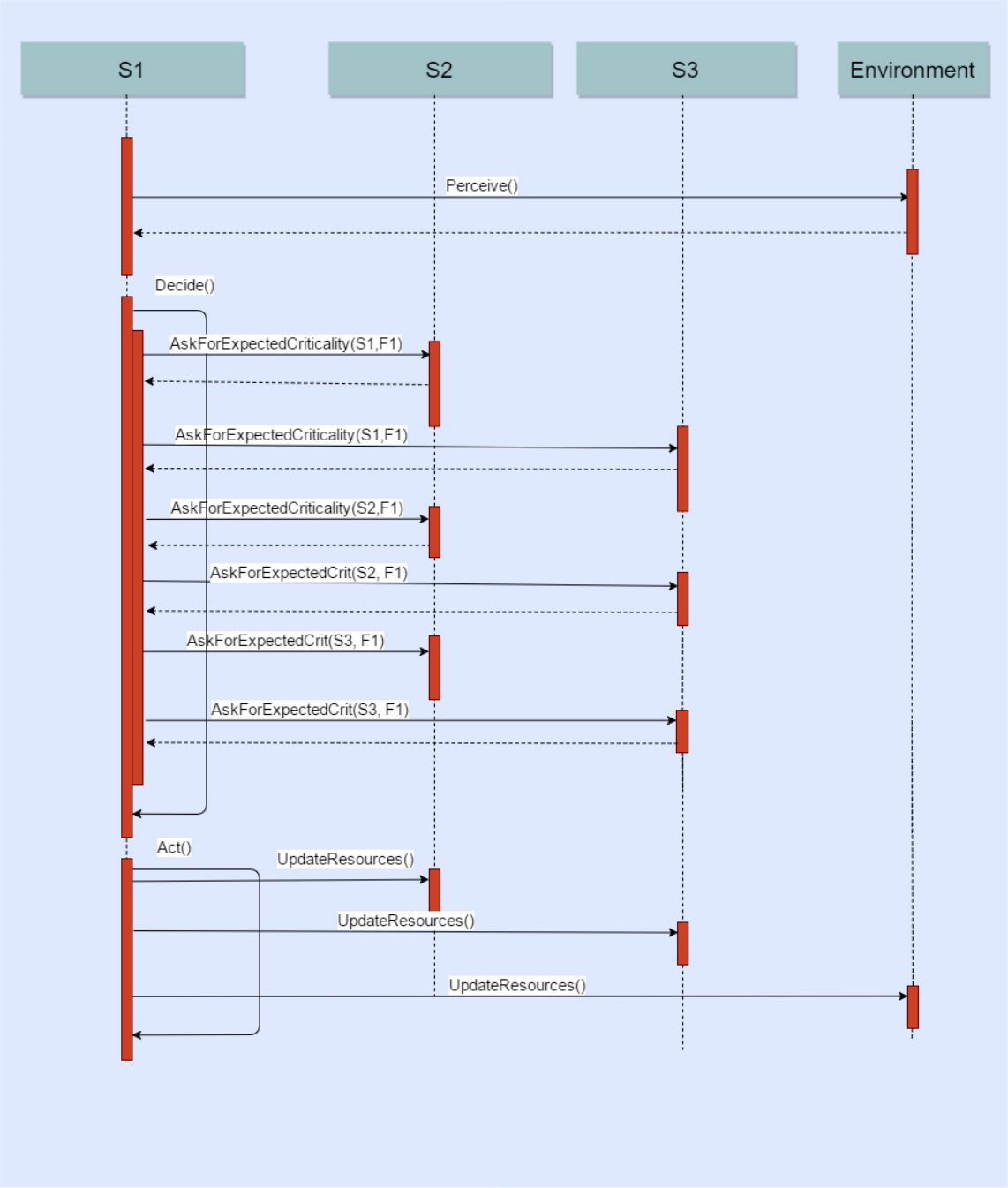


Figure 6.5 – *Perceive − Decide − Act* cycle for three component systems S1,S2 and S3

Decision: The decision phase of a component system consists in the decision algorithm based on cooperation presented in the chapter 5. As Acq has changed during the perception phase and because the anticipated criticality of all the component systems in L and Acq have possibly changed, the algorithm constructs a new cooperation table. Then, it chooses the functionality that helps the most critical component system (including itself) by minimizing the maximum of the anticipated criticality of its neighborhood (see chapter 5 for more details).

Action: Once the component system has taken its decision, the functionality is executed. As an example, let's take a functionality $F = \{f, t, p\}$ with $f = Conditions \rightarrow Effects$. At this action phase, the *Conditions* are applied. At the action phase t cycles after this one, a random number $r \in [0, 1]$ is compared with the probability p. If $r <= p$ (F succeeds), the $effects$ are applied. If $r > p$ (F fails), the effects are not applied and the conditions are canceled. As an example, if S applies this functionality,

$$F_{SendToSyS} = \{\{Signal \leqslant 1\} \rightarrow \{Signal \mathrel{+}= -1\}, \{SyS.R(Signal) \mathrel{+}= 1\}, 2, 0.99\}$$

Then, at this cycle, one $Signal \in R$ resource is consumed. Then, 2 cycles after this one and if $F_{SendToSyS}$ succeeds, the effects are applied: the component system *Sys* get one *Signal* resource. if $F_{SendToSyS}$ fails, the *Signal* resource is not consumed for S.

c) Environment Module

The environment module executes the module representing the entities and the one representing the rules. In the same manner as the modules of the component systems, a module is generated for each entity in the environment. The entity module works also on a $Perceive - Decide - Act$ cycle. Finally, the rule module that contains all the rules in the environment is also executed.

6.3.2 SApHESIA Viewer

When a experiment is running, the SApHESIA Viewer automatically shows in another window the data that the user chooses to display. The data can be resources, criticality and so on. In this way, the user can visualize the state of component systems. For example, in the Missouri Toy Problem, user can easily display how many *Signal* has been sent and how many *Signal* has been received. Links between component systems and entities can also be visualized as showed in figure 6.2. Figure 6.7 shows an example of graph than can be displayed. This figure displays the number of *Signal* owned for each component system (the graph to the left) and the number of links created by each component system (the graph to the right). As shown in figure 6.3, the viewer is updated once the component systems of the SoS and the entities in the environment have been executed.

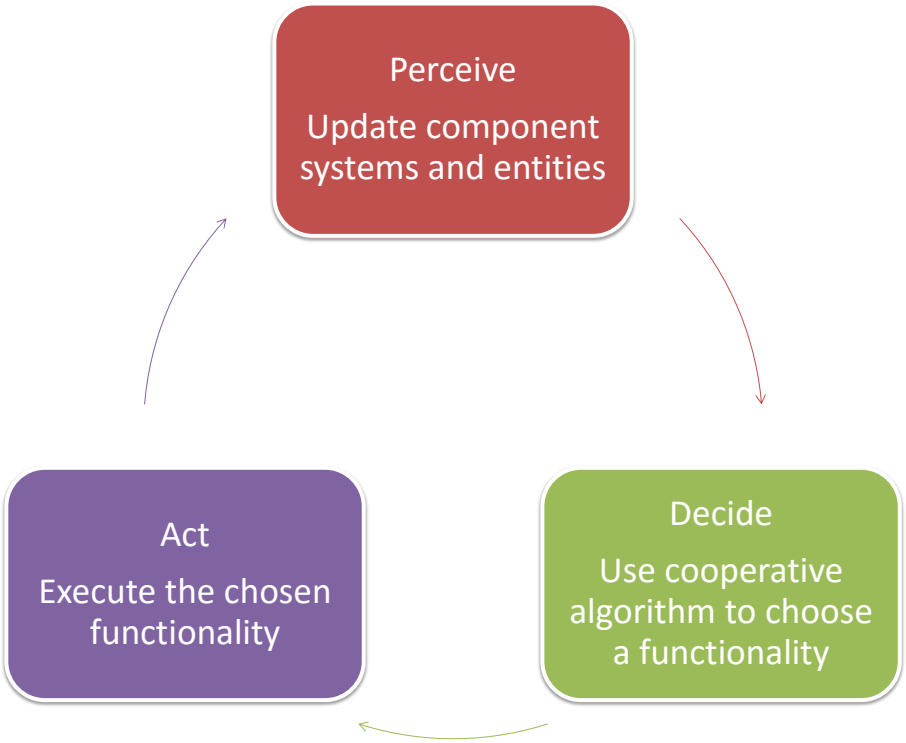


Figure 6.6 – SApHESIA component system behavior

6.3.3 Implementation of SApHESIA

SApHESIA has been implemented in JAVA language. JAVA has been used because it is a well maintained programming language that is based on the Object paradigm close to the agent paradigm. Indeed, the Object paradigm owns the concept of properties and methods that are useful to represent the $Perception - Decision - Action$ cycle of an agent. Figure 6.8 shows the main classes used to implement SApHESIA tools.

We propose concrete examples of each element of the SApHESIA model in the annex A.

6.4 SML: SApHESIA Modeling Language

SML is the XML based language used to save and load SoS experiments through the SML Parser & Compiler. It enables to declare each type of element (component system, entity, rule and so on) of the SApHESIA model. The following sections present in details the syntax to declare the different elements of the model such as component system, SoS and the environment. Each element of the model is presented through a hierarchical tree where each node represents a XML keyword usable to declare a property of a SApHESIA element.

Figure 6.7 – SApHESIA graph viewer

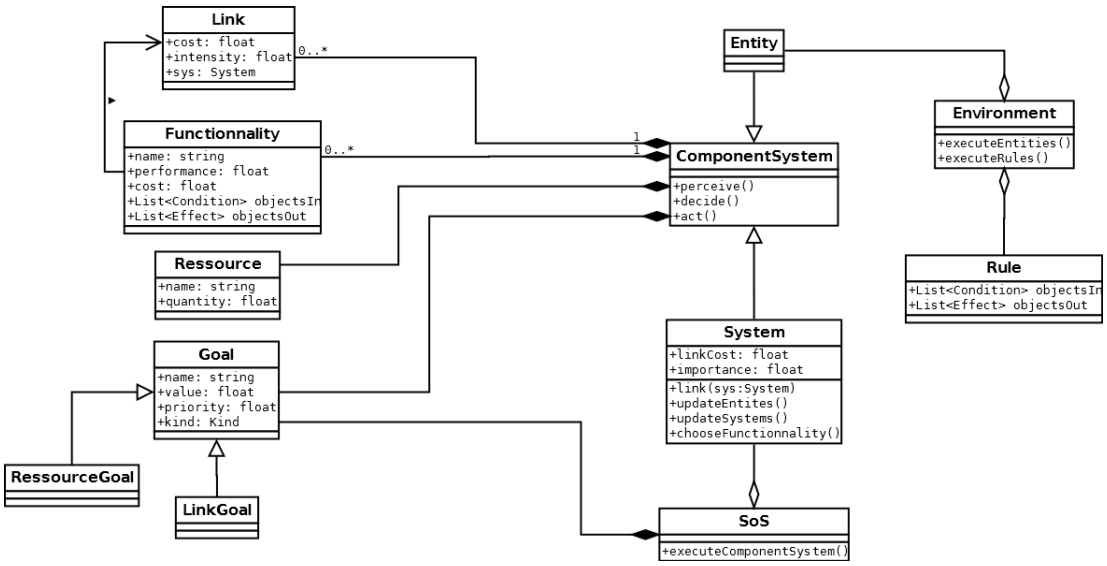
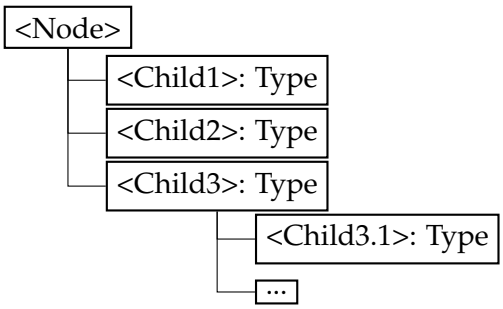


Figure 6.8 – SApHESIA main UML diagram



6.4.1 SoS and Component Systems

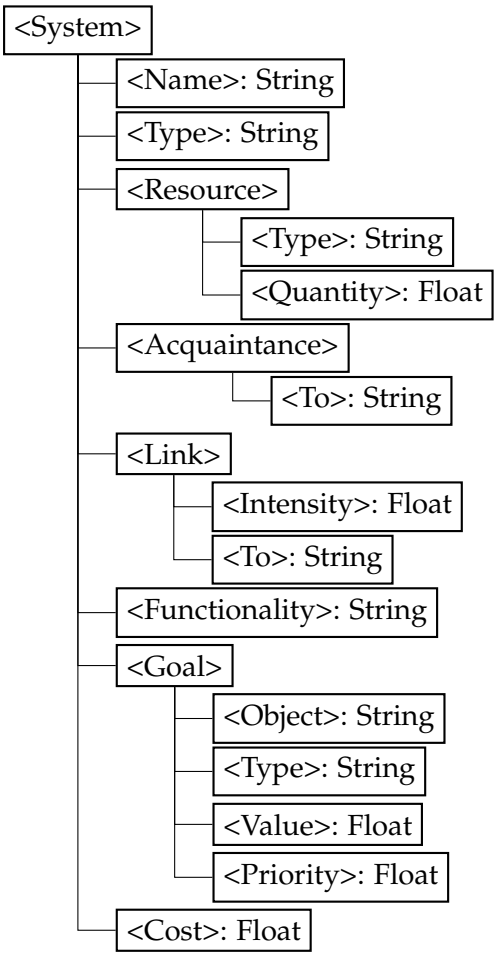
This section presents the syntax for SoS and component systems.

a) Component System

This section describes how to declare a component system in SML. In SApHESIA model, a component system S is declared as:

$$S = \{T, R, Acq, L, F, G, Cost\}$$

Then, the associated SML for a component system is the following:



An example of component system described in SML is given in annex A.

b) Functionality

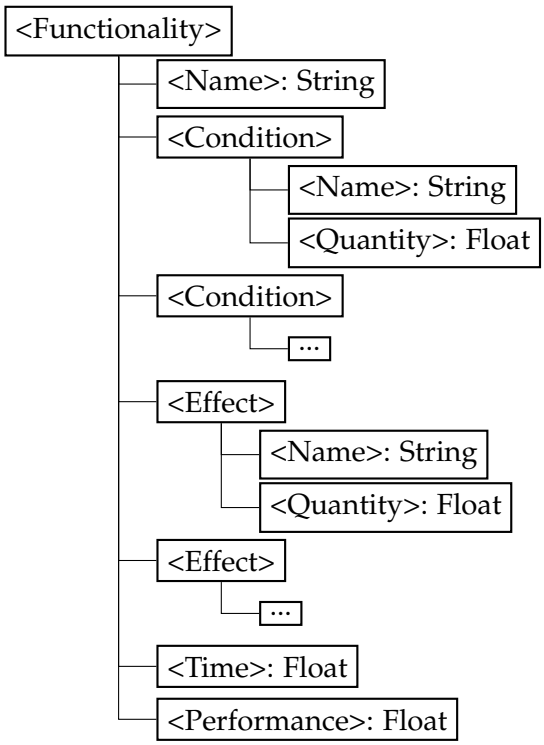
A functionality $\mathcal{F}$ in SApHESIA language is declared as follow:

$$\mathcal{F} : \{f, t, p\}$$

where:

- f is the function of $\mathcal{F}$ defined as: $f : Conditions \rightarrow Effects$;
- t is the execution time of $\mathcal{F}$;
- $p \in [0, 1]$ is the performance of $\mathcal{F}$. It represents the probability of $\mathcal{F}$ to success.

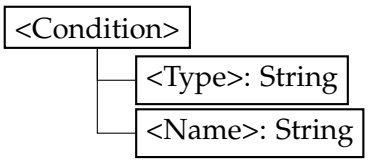
Then, the associated SML for functionality is the following:



An example of functionality described in SML is given in annex A.

c) Conditions and Effects

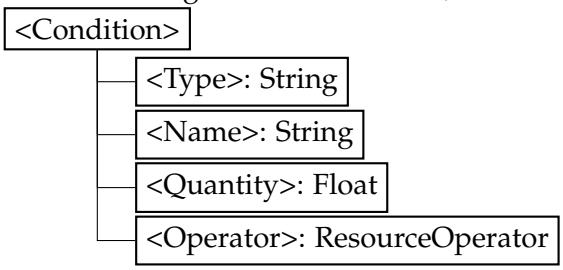
Conditions can be used with the following structures concerning each type of conditions (i.e., link, existence and resource conditions):



This first definition can be used for link and existence conditions. For example, to define a link condition to a *SatA* component system, the following syntax is used:

```
<Condition  >
        <Type> Link </Type>
        <Name> SatA </Name>
</Condition>
```

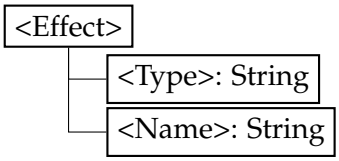
Concerning resource condition, the definition is the following :



With:

$$\texttt{ResourceOperator} \in \{\texttt{==}, \texttt{SUP}, \texttt{SUPEQ}, \texttt{INF}, \texttt{INFEQ}\}$$

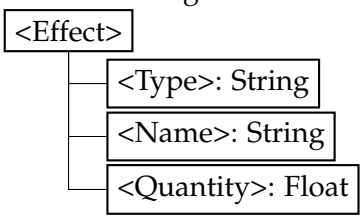
Conditions can be used with the following structures for the link and resource effects:



If a link effect is to create a link to an UAV component system, the following syntax is used:

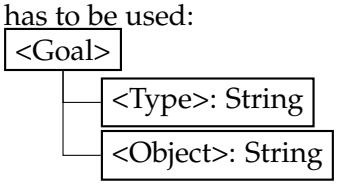
```
<Effect  >
        <Type> Link </Type>
        <Name> UAV </Name>
</Effect>
```

Concerning resource effect, the definition is the following:



d) Goals

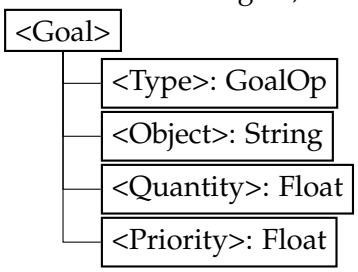
Goals can be of one of both kinds: *Link* or *Resource*. For a link goal, the following structure has to be used:



For example, if a goal is to create a link to the *Carrier* component system, it has to be declared as:

```
<Goal>
        <Type> Link </Type>
        <Object> Carrier </Object>
</Goal>
```

For a resource goal, the following structure has to be used:



With:

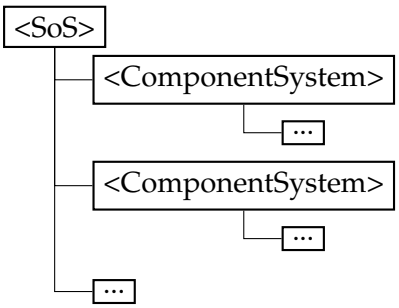
$$GoalOp \in \{EQ, NEQ, SUP, SUPEQ, INF, INFEQ\}$$

For example, if a goal is to get zero resource of type *Signal*, it has to be declared as:

```
<Goal>
        <Object> Signal </Object>
        <Type> EQ </Type>
        <Value> 0 </Value>
        <Priority> 1 </Priority>
</Goal>
```

e) SoS

A SoS is composed of component systems. To declare a SoS, the following structure has to be used:



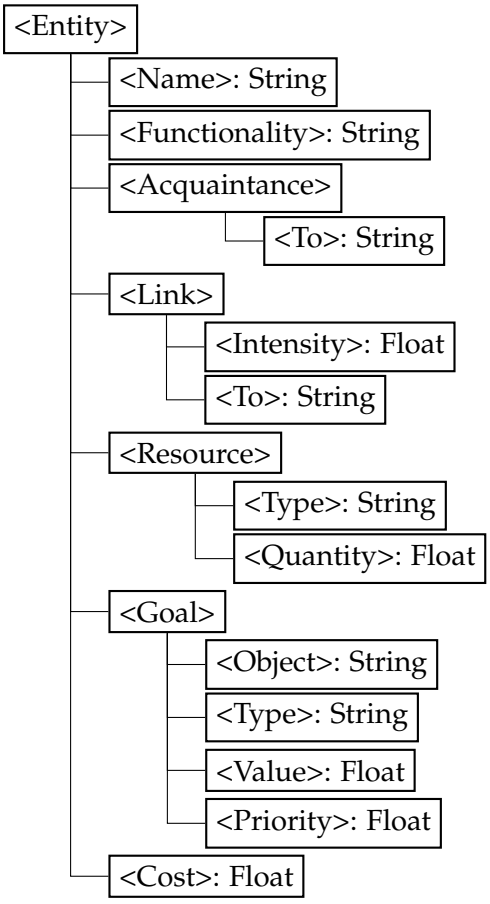
An example of SoS described in SML is given in annex A.

6.4.2 Environment

This section presents the syntax for the elements of the environment that are entity and rule.

a) Entity

This section describes in SApHESIA language how to declare an entity. The structure is close to component system one. The associated SML for entity is the following:



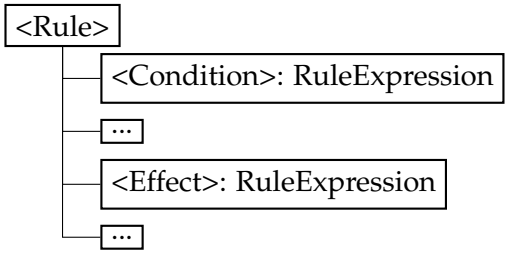
b) Rule

A rule is composed of conditions and when these are fulfilled, effects of these conditions are applied. Rule can be applied on resources and links of component systems and entities. A rule *RuleExpression* is defined as:

$$RuleExpression = S.R(Re)\ \mathtt{Op}\ S'.R(Re)$$

With:

- S and S' are component systems or entity types.
- $S.R(Re)$ the number of resource Re owned by S;
- $S'.R(Re)$ the number of resource Re owned by S';
- $\mathtt{Op} \in \{EQ, NEQ, SUP, SUPEQ, INF, INFEQ\}$.



As an example, hereafter a rule generating the resource *Signal* to the component system *Ground* when this one has its resource R_{On} greater than 0:

```
<Rule>
        <Condition>
                'Ground.R_On' SUP 0
        </Condition>

        <Effect>
                'Ground.Signal' EQ 'Ground.Signal' + 0.09
        </Effect>
</Rule>
```

Then, at each step of the experiment, *Ground* component system generates 0.09 *Signal* resource to itself.

6.5 Conclusion

As SoS architecting tools are domain-dependent and/or not available in the public domain, this chapter proposed a set of tools in order to model and run easily SoS experiment through

the SApHESIA model. The GUI generator and our language called SML enables to translate a SApHESIA model to a computable model usable by our tools. The GUI also enables to change dynamically the SoS composition by adding or removing component systems or entities. Thanks to the viewer, user can display useful data that enable to validate our SoS architecting heuristic based on cooperation through different scenarios. The following chapters present experimentations that use these tools to validate our heuristic.

Part III

Experimentations

7 Instantiation of our SApHESIA Model to the Missouri Toy Problem

This chapter presents the Missouri Toy Problem (introduced in chapter 6) that is one of the first SoS example studied in this field. The aim of this chapter is to show that our heuristic based on cooperation can resolve this problem in a totally decentralized way by finding new architectures without recalculating the entire solution in a dynamic environment (by triggering component systems failures). By "solution", we mean an architecture that tends to be the most **functionally adequate**, **efficient** and **robust**. To evaluate these three criteria, we define and use evaluation metrics described in the following sections. Then, the model is tested and discussed through specific scenarios containing disturbing events such as component systems failures.

7.1 Presentation of the Missouri Toy Problem

The Missouri Toy Problem is a SoS architecting scenario initially presented by [DeLaurentis et al., 2012]. The goal of this SoS is to *Relay commands and ISR data from ground station to a Carrier Battle Group via Unmanned Aerial Vehicles (UAVs) and Satellites*. The ground station needs intermediary component systems such as UAVs and satellites because it cannot send signals directly to the carrier battle group. This problem has been extended by [Edward Pape II, 2016] *"for purposes of having a few more systems to choose from"* (i.e., to propose a more complicated problem). The following types of component systems can be interfaced with each other:

- the ground station *Ground*, its functionality is to *send a signal* to satellites and UAVs;
- the UAV UAV, its functionality is to *relay a signal* to UAVs and other satellites;
- the satellite of type A Sat_A, its functionality is to *relay a signal* to UAVs and other satellites of type A (not to satellites of type B);
- the satellite of type B Sat_B, its functionality is to *relay a signal* to UAVs and other satellites of type B (not to satellites of type A);

- ▷ the carrier battle group *Carrier*, its functionality is to *receive a signal* from other component systems except the ground station;

Every of them can communicate with each other excepting:

- ▷ *Ground* that cannot exchange with *Carrier*;
- ▷ Sat_A that cannot exchange with Sat_B and vice versa.

Perturbations in the environment can be events such as cyber-attacks, weather issues, jamming of communication links and the availability of component systems.

7.2 Evaluation Criteria and Associated Metrics

This section presents and defines the evaluation criteria and the associated metrics we propose to evaluate them. Each of our criteria is formally defined in the following section.

7.2.1 Functional Adequacy

We want to propose a decentralized architecting heuristic that leads to a *SoS* that is functionally adequate (i.e., it achieves the expected overall function from the viewpoint of an external observer who knows the SoS purpose). To validate its adequacy, we propose a metric that will enable to evaluate the overall behavior of the *SoS* in relation with what it has made for (i.e., its functional adequacy): the *SoS* has been designed to communicate signal from the *Ground* station to the *Carrier* one (each one represented by a component system). We propose as a metric of functional adequacy the ratio TTS (Total Transmitted Signal) between the number of *Signal* resources that are generated by the *Ground* and the number of *Signal* that are effectively received by *Carrier*:

$$TTS = \frac{GS}{RS}$$

With:

- ▷ GS corresponding to the number of generated *Signal* by the *Ground* component system;
- ▷ RS corresponding to the number of received *Signal* by the *Carrier* component system.

7.2.2 Efficiency

As a reminder, [Edward Pape II, 2016] uses the concept of Key Performance Attributes (KPAs) to evaluate SoS architecture proposed by its SoS architecting heuristic based on Based-Wave model (presented in chapter 2). One of this KPA, for the Missouri Toy Problem, comes from Functional Dependency Network Analysis (FDNA). In few words, FDNA is a technique used in Logistics to evaluate the impact on the operability (also called performance) of a group of systems where dependencies between each other system exist. In FDNA, the group of systems is represented through an oriented graph where each node n_S

represents a system S and each oriented link represents a dependence between two systems. To represent operability, each S node owns a calculated value of *operability*. Moreover, each link owns a given and static value $v_{SS'}$ representing the *strength of dependence* of S for S'. All of these strengths of dependence are represented through a $M_n(\mathbb{R}^+)$ matrix where each coefficient a_{ij} represents the strength of dependence of system i to system j. Operability of each system is based on the average of the operability of S's dependencies plus the associated strengths of dependency. Then, giving the operability of root nodes (i.e., with no predecessors) the operability of each node can be sequentially computed, starting from the root nodes, in a breadth-first way: after the roots, nodes directly depending on the root are analyzed, and so on. A strong limitation of FDNA is the obligation to use acyclic graphs, because loops lead to infinite sequential computation. In [Edward Pape II, 2016], FDNA enables to define a KPA on the SoS performance where each node represents a component system. Then, the operabilities of component systems can be computed and, finally, operability computation of the *Carrier* component system gives an indication about the performance of the proposed SoS architecture (i.e., the more the *Carrier* operability is high, the more the SoS architecture is efficient).

We propose to re-use this approach with matrices to model the different efficiencies of component systems to send signals between each other and deduce the overall SoS efficiency. In SApHESIA model, the p_{Send} value represents the performance when the F_{Send} functionality is used by a component system. Then, to be efficient, the *SoS* has to use the most efficient component systems (i.e., the ones having the best p_{Send}). To evaluate this efficiency, we propose to use a metric that represents *global cost of the SoS* called *Cost*. This cost metric is inversely proportional to the efficiency of the *SoS*. Indeed, the lower the cost is, the more efficient the SoS is. To compute *Cost*, we propose to change the F_{Send} functionality by adding a new effect with a new resource called $R_{CostSend}$:

$$F_{Send} = \{\{Signal \geqslant 1\} \rightarrow \{\{Signal \mathrel{+}= -1\}, \{R_{CostSend} \mathrel{+}= 1\}, \{Signal \mathrel{+}= -1\}, \{SyS, Signal \mathrel{+}= 1\}\}, t_{Send}, p_{Send}\}$$

With $Sys \in \{UAV, Sat_A, Sat_B, Carrier\}$. The F_{Send} functionality represents the sending of a signal between two component systems. It consumes one *Signal* and generates one *Signal* to the component system of type *SyS*. In this way, $R_{CostSend}$ saves the number of times the component system is used to send a signal. Then, the following cost metric is proposed:

$$Cost = \Sigma_{S_i \in \mathcal{S}}(S_i.Cost + S_i.R(R_{CostSend}))$$

To compare the *Cost* of the current *SoS* architecture with the *Cost* of the most efficient architecture (called *MinCost*), we will compute the ratio between these two values. More precisely, *MinCost* is the minimum cost for transmitting all the signals between *Ground* and *Carrier* for a given scenario. Thus, *MinCost* is calculated for the scenario and compared with the current cost of the architecture. The resource $R_{CostSend}$ contains the total 'cost' a component system for sending its signals. This resource is incremented through the functionality F_{Send} of each component system. Formally, the ratio is defined as:

$$CostRatio = \frac{Cost}{MinCost}$$

Table 7.1 – Evaluation metrics for the Missouri Toy problem

Functional adequacy	Efficiency	Robustness
TS	$CostRatio$	TS evolution

7.2.3 Robustness

[IEEE, 2010] defines the robustness as "the degree to which a system or a component can function correctly in the presence of invalid inputs or stressful environmental conditions". To evaluate robustness of our SoS architecting heuristic, we define events during the scenario representing failures of component systems. A failure disables the component system to use its functionalities (i.e., it cannot receive, send signals or link with other component systems). To cope with these failures, the SoS has to self-adapt through the behavior of the component systems that will find and use other component systems that are able to send signals. Finally, we propose to evaluate the robustness through the evolution of TTS. Indeed, if the SoS is not robust, the failure of a component system will stop the adequate functioning of the SoS, then the transmission of signals to the *Carrier* will be interrupted. Thus, TTS will not evolve anymore. Finally table 7.1 summarizes the metrics used for each evaluation criteria.

7.3 Scenarios Description

This section describes the scenarios by giving the available component systems for the SoS, the initial values of the different component systems resources and the description of the scenario.

7.3.1 Available Component Systems and Initialization Values

The SApHESIA model used in these scenarios is exactly the same as proposed in chapter 6. The SoS is composed of 21 component systems of different types:

- 1 *Ground* g;
- 5 UAV $u_i, i \in [\![1,5]\!]$;
- 8 Sat_A $a_i, i \in [\![1,8]\!]$;
- 6 Sat_B $b_i, i \in [\![1,6]\!]$;
- 1 *Carrier* c.

The main goal of the SoS is to assure robust transmission from the *Ground* to the *Carrier*. Each type of component system is initialized with the parameters given in table 7.2. $R_{Signal} = 0$ means that any component system owns *Signal* to send at the beginning of the simulation. $R_{ResourceLink}$ is set to 1 for all the component system to avoid too much links creation between component systems when the functionality $F_{LinkSyS}$ is used:

Table 7.2 – Resources initialization of the Missouri Toy problem component systems

	g	u_1	u_3	a_1	a_2	Other UAV	Other Sat_A	Sat_B	c
R_{Signal}	0	0	0	0	0	0	0	0	0
$R_{ResourceLink}$	0	1	1	1	1	1	1	1	1
$R_{CostSend}$	1	1	1	1	1	2	2	2	X

Table 7.3 – Parameters of SApHESIA Engine

Reinforcement link	0.1
Destruction link	0.1

$$F_{LinkSyS} = \{\{\{SyS \in Acq\}\} \rightarrow \{\{Link, 1\}, \{SyS.R(R_{LinkResource}) \mathrel{+}= -1\}, t_{Link}, p_{Link}\}$$

The consumption of $R_{LinkResource}$ by this functionality avoids the component systems to create too many links between each other. The SApHESIA engine is initialized with the parameters contained in 7.3. Then, each link intensity is decreased of $Destructionlink = 0.1$ when the component system does not used it in terms of functionality. At the contrary if the link is used, the intensity of this one is increased of 0.1 (thanks to Reinforcement link). Concerning the total number of signal to transmit, the rule $Rule_{Signal}$ (defined in section 4.6.3) generates $Signal$ with a rate of $g.R(R_{GeneationRate}) = 0.09$ until cycle 2000. Then, the total number of signals is equal to 180 (0.09×2000).

7.3.2 Minimum Cost Calculation

The table 7.5 gives the different performances of the F_{Send} functionality for each component system. For example, the UAV u_1 has a performance of 0.5 when sending a signal to Satellite a_1. Empty cells correspond to a 0. The values have been chosen to make a path more efficient that all the other paths. Then, this table enables to calculate the most efficient path for the 180 signals to transmit, i.e., the one with the highest performances (because composed of component systems with the highest performance): $Ground \rightarrow u_3 \rightarrow a_3 \rightarrow Carrier$. Concerning $R_{CostSend}$, the value for each component system is given in table 7.2. Then, these value enable to compute $MinCost$:

$$MinCost = 180 \times \Sigma_{S_i \in Opti}(S_i.Cost + S_i.R(R_{CostSend}))$$

With: $Opti = \{Ground, u_3, a_3\}$.
Finally, the numerical application (thanks to table 7.4 and 7.5) gives $MinCost = 180 \times (Ground.Cost + u_3.Cost + a_3.Cost + Ground.R(R_{CostSend}) + u_3.R(R_{CostSend}) + a_3.R(R_{CostSend})) = 180 \times (1+1+1+1+1+1) = 1080$.

Table 7.4 – Cost of each component system

	G	u_1	u_2	u_3	u_4	u_5	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	b_1	b_2	b_3	b_4	b_5	b_6	C
$Cost$	1	1	2	1	2	2	1	2	1	2	2	2	2	2	2	2	2	2	2	2	2

Table 7.5 – Performance p_{Send} of F_{Send} functionality

	G	u_1	u_2	u_3	u_4	u_5	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	b_1	b_2	b_3	b_4	b_5	b_6	C
G		0.7	0.1	0.4	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
u_1			0.1	0.1	0.1	0.1	0.5	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
u_2		0.1		0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
u_3		0.1	0.1		0.1	0.1	0.1	0.1	0.1	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
u_4		0.1	0.1	0.1		0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
u_5		0.1	0.1	0.1	0.1		0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
a_1		0.1	0.1	0.1	0.1	0.1		0.1	0.1	0.1	0.1	0.1	0.1	0.1							1
a_2		0.1	0.1	0.1	0.1	0.1	0.1		0.1	0.1	0.1	0.1	0.1	0.1							0.1
a_3		0.1	0.1	0.1	0.1	0.1	0.1	0.1		0.1	0.1	0.1	0.1	0.1							1
a_4		0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1		0.1	0.1	0.1	0.1							0.1
a_5		0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1		0.1	0.1	0.1							0.1
a_6		0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1		0.1	0.1							0.1
a_7		0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1		0.1							0.1
a_8		0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1								0.1
b_1		0.1	0.1	0.1	0.1	0.1										0.1	0.1	0.1	0.1	0.1	0.1
b_2		0.1	0.1	0.1	0.1	0.1									0.1		0.1	0.1	0.1	0.1	0.1
b_3		0.1	0.1	0.1	0.1	0.1									0.1	0.1		0.1	0.1	0.1	0.1
b_4		0.1	0.1	0.1	0.1	0.1									0.1	0.1	0.1		0.1	0.1	0.1
b_5		0.1	0.1	0.1	0.1	0.1									0.1	0.1	0.1	0.1		0.1	0.1
b_6		0.1	0.1	0.1	0.1	0.1									0.1	0.1	0.1	0.1	0.1		0.1
C																					

7.4 Scenario 1: Functional Adequacy, Efficiency and Robustness Testing

This first Scenario has been defined to evaluate the functional adequacy, efficiency and robustness of SApHESIA on the Missouri Toy Problem. From cycle 0 to 2000, every component systems are keeping up and running. At cycle 2000, the failure of the component system a_3 occurs. Then, every other component systems will not able to link or send signal to a_3. This event is used to study the robustness of the SoS. If the *SoS* is able to adapt the path in an efficient manner, the new path should be $Ground \rightarrow u_1 \rightarrow a_1 \rightarrow Carrier$ because these component systems have the second best p_{Send} (calculated thanks to table 7.5). At the same time (cycle 2000), the generation of signals will be interrupted. This event is important because it enables to compare at the end of the simulation the number of generated signals and the number of received signals (through the metric *TTS*). At cycle 7000, the simulation is over.

7.5 Scenario 2 Description: Strong Robustness Testing

Scenario 2 has been defined in order to generate more failure events to better test robustness of the SoS. A component system failure is now generated every 100 cycles to see how the

functioning of the SoS is impacted. As component systems do not have the same performance on F_{Send}, the order of component system failures has consequences on the evolution of the SoS functioning. Indeed, if all the component systems with a good performance fail first, the SoS will have more difficulty to recover from failures. Thus, 3 sequences will be created where the position of the failures of the most efficient component systems (a_1, a_3, u_1, u_3) will be changed. Then, the sub-sequence $< a_1, a_3, u_1, u_3 >$ will be respectively, at the end (s_{easy}), at the middle (s_{medium}) and at the beginning of the sequence (s_{hard}). *Ground* and *Carrier* are not concerned as they are respectively the source and the destination of the signals; their failures stop directly the sending of *Signal*:

- $s_{easy} = \{u_2, u_4, u_5, a_2, a_4, a_5, a_6, a_7, a_8, b_1, b_2, b_3, b_4, b_5, b_6, a_1, a_3, u_1, u_3\}$
- $s_{medium} = \{u_2, u_4, u_5, a_2, a_4, a_5, a_6, a_1, a_3, u_1, u_3, a_7, a_8, b_1, b_2, b_3, b_4, b_5, b_6\}$
- $s_{hard} = \{a_1, a_3, u_1, u_3, u_2, u_4, u_5, a_2, a_4, a_5, a_6, a_7, a_8, b_1, b_2, b_3, b_4, b_5, b_6\}$

The *TTS* metric will be used to highlight the inability of the SoS to send a *Signal* from *Ground* to *Carrier*.

7.6 Results Discussion

7.6.1 Scenario 1

Figure 7.1 shows the simulation results for the scenario 1. The curves **CostRatio**, **Signal**, **Criticality** and **TTS** show respectively the evolution of the *CostRatio*, *Signal* resource, criticality and *TTS* of the different component systems. The curve **F_Send_UAV1&3** shows the number of times on the last 500 cycles the *Ground* has used F_{Send} on u_1 and u_3. The curve **F_Send** shows the total number uses of F_{Send} by the component systems and with which component systems it was used.

After the first cycles of the simulation, *TTS* begins to increase (the number of signal transmitted is growing). Then, the *SoS* finds its functional adequacy early. Before cycle 2000, figure F_{Send} shows that *Ground* uses u_3 more than u_1 to send signal, showing that the most efficient path is used.
After cycle 2000 (corresponding to the failure of a_3), *TTS* is always increasing, showing that the *SoS* is still running even if a failure has occurred proving a first sight of robustness of the SoS. *TTS* is highly increasing after turn 2000 because the generation of signals is over. Moreover, *CostRatio* increases even more after cycle 2000 because the failure of a_3 leads u_3 and u_1 to find alternative paths that are less efficient than $Ground \rightarrow u_3 \rightarrow a_3 \rightarrow Carrier$ which is the current most efficient path. The F_{Send} curve shows that finally u_1 is preferred to u_3 because of this new most efficient path ($Ground \rightarrow u_1 \rightarrow a_1 \rightarrow Carrier$). The **F_Send_UAV1&3** curve confirms that the most used component systems are *Ground*, u_1, u_3, a_1 and a_3. Finally, $CostRatio < 1.4$, shows that even with failures, the SoS finds efficient alternative paths.

Ground starts by searching other component systems through the perception phase and creates links to them (to fulfill the G_{Link} goal). At the beginning, *Ground* does not choose

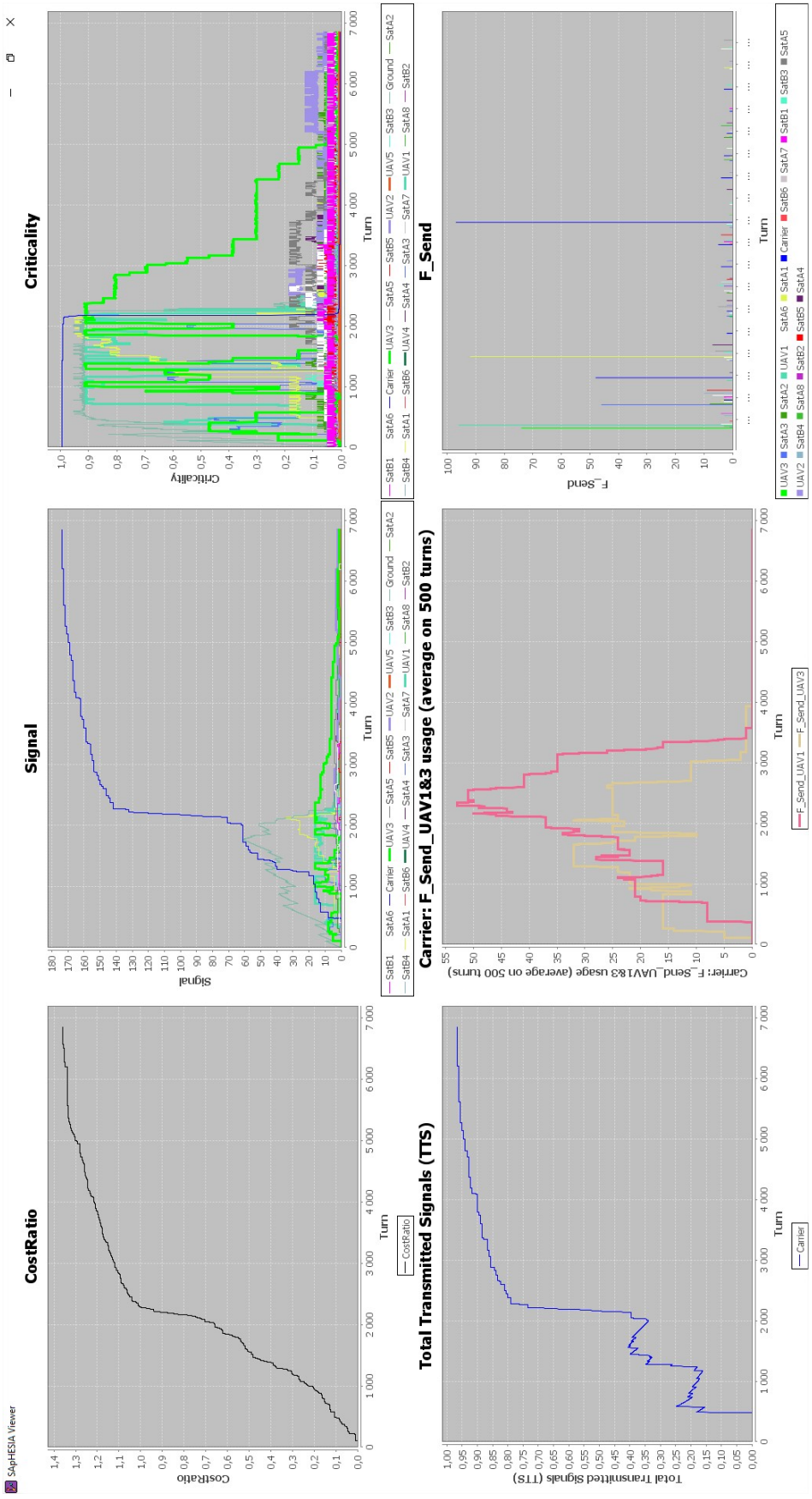


Figure 7.1 – Scenario 1: Simulation results

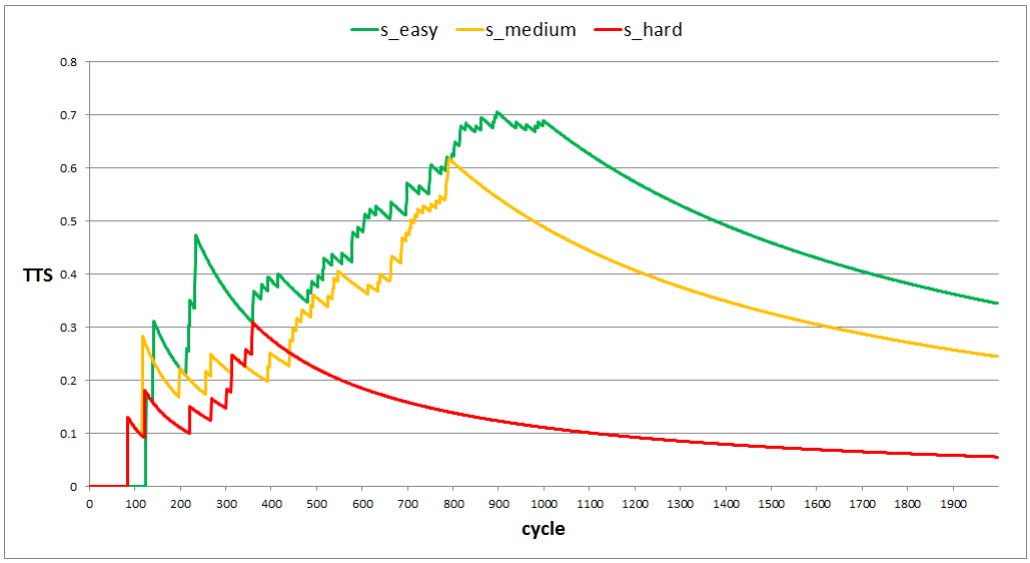


Figure 7.2 – Scenario 2: Evolution of TTS for the three sequences s_{easy}, s_{medium} and s_{hard}

always the most efficient component system for it (u_1 and u_3). Once the first links are created, *Ground* starts to send signals to other component systems (thanks to the newly created links). But, the p_{Send} limits the success of sending signals to less efficient component systems. For example, if *Ground* chooses u_2 ($p_{Send} = 0.1$), only 1 sending attempt on 10 is successful. But, during this time, the link self-destructs through the link destruction mechanism (here $LinkDestruction = 0.1$). The link is often destructed before a signal is sent (for less efficient component systems as u_2). It happens that *Ground* success to send signals to the less efficient component systems, but in this case, *Ground* does not send another signal because the criticality of the component system is increasing (through goal G_{Signal}). Then, less efficient component systems becomes "overloaded" by signals they transmit at a very low rate because of their low p_{Send}. Finally, most efficient component systems are mainly used because they succeed to send signals because of their high p_{Send}. In this way, *SoS* will make emerge the most efficient path.

7.6.2 Scenario 2

Figure 7.2 shows the evolution of *TTS* for the second scenario dealing with the robustness testing. Green, orange and red curves are respectively the results for the s_{easy}, s_{medium} and s_{hard} sequences of failures. Firstly, these results show that the order of failures has an important impact on the global functioning of the SoS as the functionalities of the component systems have different performances concerning F_{Send}. The *TTS* evolution shows that the SoS is no more able to send *Signal* at cycle 1000 for s_{easy}, at cycle 800 s_{medium} and at cycle 400 for s_{hard}. A failure appearing every 100 cycles, the SoS is able to cope with a failure rate of 52% (10 failures), 41% (8 failures) and 21% (4 failures) respectively for s_{easy}, s_{medium} and s_{hard} for a total of 19 component systems. This scenario illustrates the robustness of the SoS driven by our cooperative heuristic: even with several component system failures, the SoS is able to find alternative architectures to continue the sending of signals. These alter-

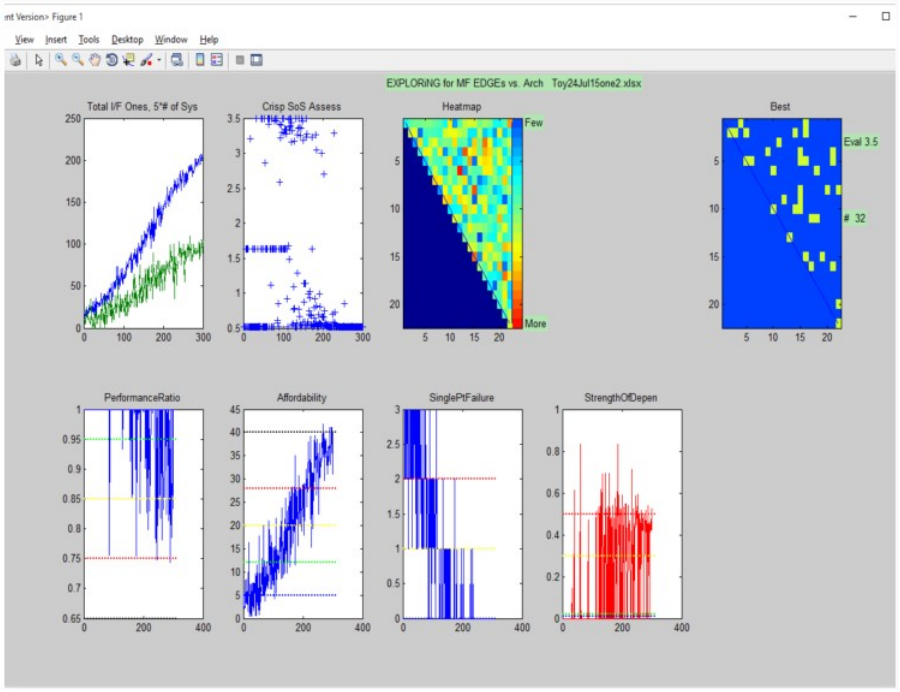


Figure 7.3 – Results of [Edward Pape II, 2016]

native architectures are found thanks to the ability of each component system to find new neighbors.

7.7 Conclusion

In the first scenario, the *SoS*, through the cooperative behavior of the component systems, is able to find architectures that are **robust**, **functionally adequate** and **efficient** according to our metrics. The scenario 2 confirms the robustness of our approach with in the worst case, a failure rate of 21%. Finally, The comparison of our solution with [Edward Pape II, 2016] has been difficult because the results are barely legible to compare it properly. The different curves of its experiments show global tendencies such as the evolution of the KPAs (such as Affordability and Performance) but the extraction of concrete data such as the *best architecture* is not possible (see figure 7.3) because of a lack of explanation. But, our decentralized heuristic has intrinsic advantages. For example, [Edward Pape II, 2016] does not use direct cooperation between component systems, disabling experiment on virtual and collaborative SoS.

8 CoCaRo: a Resource Transportation System

This chapter presents the instantiation of SApHESIA to a resource transportation system called CoCaRo (Color Carrier Robot) as well as obtained experimental results. This system, partially described in chapter 5 to introduce the concept of criticality in the AMAS approach has been used to validate two main hypotheses. The first one is that **the use of the criticality enables cooperation between agents**, improving the effectiveness and the robustness of an AMAS. The second one is the capability to use SApHESIA to **reify an AMAS as a component system** in a SoS. To reach these two hypotheses, three variations of the CoCaRo system are studied. The first one is the CoCaRo system without any cooperation between constituent elements. The second one is the CoCaRo system with cooperation through the exchange of the criticality between elements. Finally the last one is the CoCaRo system using SApHESIA and cooperation between elements. The first section introduces the CoCaRo system. Then, the second section describes more in details the three variations of the CoCaRo system. Finally, the third one presents the experimentations done and discusses the obtained results.

8.1 General Description

CoCaRo (Color Carrier Robot) is a resource transportation system by mobile robots. A robot (that may be red, blue or green) has to find and pick up a box (that also may be red, blue or green) and to drop it in an area (called the nest) having the same color as the box. Each robot has an initial amount of energy that it consumes at each movement, but when it drops a box in a nest, it receives a reward of energy allowing it to remain longer alive. The value of the reward depends on the color of the dropped box compared to the color of the robot (i.e., a red robot gets a better reward if it drops a red box). The next sections describe the environment and the agentification of CoCaRo.

8.1.1 Environment

The **environment** is composed of three different types of objects:

- The *Displacement Grid*: it is an object of the plan, made of 50x50 square cells. A cell may be empty or may contain a robot, a box and/or a nest;

- the *Nest*: it is an location in which robots drop boxes. The grid contains three nests (one blue, one red and one green) that are equidistant from each other in order to prevent bias related to the proximity of two nests.

- the *Boxes* that can be carried and dropped into nests by robots. A box can be red, blue or green. Boxes appear randomly on the grid, at regular time intervals.

8.1.2 CoCaRo as a Multi-Agent System

CoCaRo is composed of a single type of agent, the robot agent. The robot agent has its own states, attributes, perceptions and actions.

a) Perception

Each robot perceives (under a given perception limit, that will be given for the simulation) boxes around it. Two variables are used, the first one is used to save the box the robot wants to carry (the *targeted_box* variable) and the second one to save the box it is carrying (the *carried_box* variable). These two variables are used hereafter in the decision phase of the robot.

b) Actions and States

A **robot agent** performs one of the following *actions*:

- *move*, to move on the grid according to a Monte Carlo distribution until it finds a box;

- *deposit*, to drop a box on the grid or on a nest;

- *take*, to pick up a box from the grid;

- *go*, to go to a particular cell on the grid.

A robot agent has four *states*:

- *carried*: the agent is carrying a box;

- *target*: the agent has targeted a box;

- *onPosBox*: the agent is on the same cell as its targeted box;

- *onPosNest*: the agent is on the same cell as the nest corresponding to its carried box.

The agent chooses the most appropriate action according to its current state and its perceptions. The table 8.1 sums up them.

States	Actions
$\neg target \wedge \neg carried$	$move$
$target$	$go(targeted\_box)$
$target \wedge onPosBox$	$take(targeted\_box)$
$carried \wedge \neg onPosNest$	$go(nest)$
$carried \wedge onPosNest$	$deposit(carried\_box)$

Table 8.1 – Agent actions

c) Energy Level

Each robot can move on the grid thanks to an energy level that is given at the beginning of the simulation and received when it deposits a box in a nest. The energy level decreases of a value of *conso* at each time unit. When its energy level reaches zero, the robot cannot be used anymore: it is "dead". For a robot r_i, the energy level at time t is noted $Ne_{r_i}(t)$. We call B the set of boxes in the environment and $B_{r_i} = \{b_k \in B\}$ the subset of boxes the robot r_i brought back to the different nests between the time 0 and t. Then,

$$Ne_{r_i}(t) = \begin{cases} N_{r_i}(t) & \text{if } 0 < N_{r_i}(t) < Max_{Ne} \\ Max_{Ne} & \text{if } N_{r_i}(t) \geqslant Max_{Ne} \\ 0 & \text{else} \end{cases}$$

with:

- $N_{r_i}(t) = N_i + \sum_{B_{r_i}} rec_{r_i}(b_k) - conso * t$
- *conso*, the number of energy level units consumed during one time unit;
- Max_{Ne}, the maximum energy level;
- N_i, the initial energy level;
- rec_{r_i}, the reward function depending on the box color.

These differences of reward enable to simulate a specialization of each group of robots. Then, red robots are more efficient to bring back red boxes, blue ones to bring back blue boxes and finally green ones to bring back green boxes.

Consequently, the value of the reward depends on the box color. We call *color* the function that returns the color of a robot or a box given in parameter. If the robot r_i brings back a box b_k of its own color ($color(b_k) = color(r_i)$), then the reward is higher than for the other colors:

$$rec_{r_i}(b_k) = \begin{cases} \frac{2}{3} * Max_{Ne} & \text{if } color(b_k) = color(r_i) \\ \frac{1}{3} * Max_{Ne} & \text{else} \end{cases}$$

d) Speed

Each robot has a defined speed that depends on the energy level and is modeled by the function *Speed*. Thus, $Speed_{r_i}(t)$ is a function representing the speed of agent r_i. As this

speed decreases with $Ne_{r_i}(t)$, it enables to represent that a robot is less efficient when its energy level is low:

$$Speed_{r_i}(t) = \begin{cases} 1 & \text{if } \frac{2}{3} * Max_{Ne} \leq Ne_{r_i}(t) \leq Max_{Ne} \\ \frac{1}{2} & \text{if } \frac{1}{3} * Max_{Ne} \leq Ne_{r_i}(t) < \frac{2}{3} * Max_{Ne} \\ \frac{1}{3} & \text{if } 0 < Ne_{r_i}(t) < \frac{1}{3} * Max_{Ne} \\ 0 & \text{else} \end{cases}$$

For example, if the energy level of r_i, $Ne_{r_i}(t)$ is lower than $\frac{1}{3} * Max_{Ne}$, then it will take 3 time units for the robot to move to an adjacent cell.

e) Criticality and Anticipated Criticality

As presented before in chapter 5, the **criticality** indicates the level of difficulty of a robot agent and is defined according to its energy level (see chapter 5 for more details). In CoCaRo, the effectiveness of an agent related to its energy level may rapidly decrease. In this context, the agent is less effective when it has a lower level of energy and therefore deteriorates more rapidly. The criticality C_{r_i} of a robot agent r_i is a temporal function calculated as follows:

$$C_i(t) = Max_{Ne} - Ne_{r_i}(t)$$

With:

- $Ne_{r_i}(t)$ the perception of its energy level at time t;
- Max_{Ne}, the constant related to the goal of maximizing its energy level.

The anticipated criticality of an action a represents the future criticality of an agent if the next action it performs is the action a. It enables the agent to predict the effect of an action on its own criticality. Formally, the anticipated criticality of an agent i for an action a is defined as:

$$CA_i(t, a) = C_i(t) + Eff_i(a)$$

with $C_i(t)$ the criticality of agent i at time t and $Eff_i(a)$ a function giving the effect in term of criticality if the action a is done. Finally, the anticipated criticality can be calculated for a sequence of actions $A = \{a_1, a_2, ..., a_n\}$ as the following:

$$CA_i(t, A) = C_i(t) + \Sigma_{i \in [\![1,n]\!]} Eff_i(a_i)$$

For the robot agent r_i, the effects of the different actions of r_i is calculated as follow:

$$Eff_{r_i}(a) = \begin{cases} conso \times dist(b_j, r_i) & \text{if } a = go(b_j) \\ conso \times dist(nest, r_i) & \text{if } a = go(nest) \\ -rec_{r_i}(b_j) & \text{if } a = deposit(b_j) \\ 0 & \text{else} \end{cases}$$

Where:

- $go(b_j)$ (respectively $go(nest)$) is the action of moving to the box b_j (respectively to the nest *nest*);
- $dist(b_j, r_i) \in \mathbb{R}$ (respectively $dist(nest, r_i)$) is the distance between the box b_j (respectively between the nest *nest*) and the robot r_i;
- $deposit(b_j)$ is the action of dropping the box b_j in the nest of the same color;
- *conso* is the battery level units consumed during one time unit;
- rec_{r_i}, the reward function depending on the box color.

If the sequence of actions for bringing back a box b_j to the nest is defined as $bback(b_j) = \{go(b_j), take(b_j), go(nest), deposit(b_j)\}$, the anticipated criticality of $bback(b_j)$ is calculated as follow:

$CA_{r_i}(t, bback(b_j)) = C_{r_i}(t) + Eff_{r_i}(go(b_j)) + Eff_{r_i}(take(b_j)) + Eff_{r_i}(go(nest)) + Eff_{r_i}(deposit(b_j))$

The **anticipated criticality** allows the robot agent to know the criticality it is going to get once the box b_j is dropped in the nest. Thanks to this function, the agent can choose the most beneficial box for it (i.e., the box offering the most energy for it).

8.2 Description of the Three Systems: No Cooperation, With Cooperation, With SApHESIA

The case study of CoCaRo has to show the usefulness of the cooperation algorithm based on the criticality and the usefulness of SApHESIA to reify an AMAS as a component system. To reach these aims, three systems have been implemented. Each of these systems is described in the following sections.

8.2.1 System 1 : Non Cooperative Agents

In this system, robot agents do not use any cooperative mechanisms. An agent is looking for the most interesting box in order to get a maximum amount of energy and therefore to transport a maximum of boxes. In this system, the agents do not try to exchange boxes even if it could be advantageous for them.

Decision Algorithm - The decision algorithm is described in the algorithm 8.1 by reading only the black part. To achieve its goal (to maximize its energy level), a robot agent applies this algorithm using its current criticality and its anticipated criticality. At each turn of the simulation, the robot computes its anticipated criticality (line 3 or 6) depending on its state (carrying or targeting a box). After updating its perceptions (line 8), the robot checks if it has detected a box enabling to get more energy than it would have with the one it carries (line 9 to 23). If it has found a better box and was carrying a box, it drops it (line 19).

8.2.2 System 2 : Cooperative Agents

In this system, robot agents are cooperative. They can exchange boxes between them according to their current and anticipated criticalities.

Decision Algorithm The decision algorithm of a cooperative robot agent is described in algorithm 8.1 by reading the black and the red parts (the red parts are also surrounded by comments *Begin/End cooperative part*). When a robot agent r_i detects a box b_k enabling it to get more energy (line 11), r_i checks if the box is already owned by a robot agent r_j (line 12). If it is the case, the agent r_i sends a message to r_j which contains the criticality and the anticipated criticality of r_i for the box b_k (line 13). These information make the agent r_j to choose either to exchange the box b_k (if r_j does not become too critical because of the exchange), or to keep it (if r_j becomes too critical because of the exchange). This choice is made in the function *coop_request_processing* (line 25) and detailed in the algorithm 8.2 that runs as follow:

The agent r_j carrying the box b_k considers at each time step t, the cooperative request received at $t-1$. As it is cooperative, the agent r_j has to determine if the request sender r_i is more critical than it (line 1), in which case r_j has to give the box to r_i. For that, r_j compares the anticipated criticality of r_i with the current criticality of r_i and then checks if the exchange does not cause a criticality increase (line 2) otherwise the exchange is refused (line 7). If the agent r_j determines that the request sender r_i is less critical than it (line 10), r_j compares its own anticipated criticality with its current criticality and then checks if the exchange does not cause a criticality increase (line 11) otherwise the exchange is refused (line 12). The functions *accept_exchange* and *refuse_exchange* notify the sender of the request if r_j accepts or refuses the exchange.

The following variables are used in the algorith 8.1

- $CA_{current}$, the current criticality of r_i;
- $temp\_C_a$, the temporary minimal criticality given by boxes around r_i;
- *visible_boxes*, the boxes perceived by r_i.

8.2.3 System 3 : Instantiation of SApHESIA Model

To validate our architecting cooperative approach for SoS, we decide to instantiate CoCaRo as a SoS by using SApHESIA model. The main idea is to create 3 "meta"-robots, each representing a group of robots modeled as a component system of the SApHESIA model. Each group of robots of the same color is considered as an AMAS and may be reified as a component system (red, green and blue) of a SoS. While defining these 3 component systems (forming a SoS), their goals, functionalities, resources and links have also to be defined.

a) Justifications for Using SApHESIA

At a first glance, it seems that there is no need for SApHESIA in this kind of system. According to the AMAS approach, the cooperation between elements of the system is sufficient to

Algorithme 8.1 : Decision for a cooperative robot agent r_i

```
1  while N_e(t) ≠ 0 do
2      if carried then
3          CA_current = CA_{r_i}(t, bback(carried_box))
4      end
5      if target then
6          CA_current = CA_{r_i}(t, bback(targeted_box))
7      end
8      Update visible_boxes ;
9      forall b_k ∈ visible_boxes do
10         temp_C_a := CA_{r_i}(t, bback(b_k)) ;
11         if temp_C_a < CA_current then
12             /*Begin cooperative part                    */
13             if holder(b_k) ≠ null then
14                 SendCoopReq(r_j, C_{r_i}(t), CA_{r_i}(t, bback(b_k)))
15             end
16             /*End cooperative part                      */
17             else
18                 CA_current := temp_C_a ;
19                 targeted_box := b_k ;
20                 if carried then
21                     deposit(carried_box) ;
22                 end
23             end
24         end
25     end
26 end
27 /*Begin cooperative part                                */
28 coop_request_processing();
29 /*End cooperative part                                  */
```

have a functioning MAS (here a MAS that is able to bring back resource to predefined areas). But, first experimentations (presented in the next sections) will show that sometimes, a group of robot (red, green or blue) suffers more than the other especially when the color of the box apparition is not equiprobable. For example, if there is more probability for blue and green boxes to appear than red boxes, it would be better if the entire group of robots decides to help another group. The use of a SApHESIA enables to study the influence of the different groups of robot and to see if it can increase the overall performance of the system. This main idea is summed up in figure 8.1.

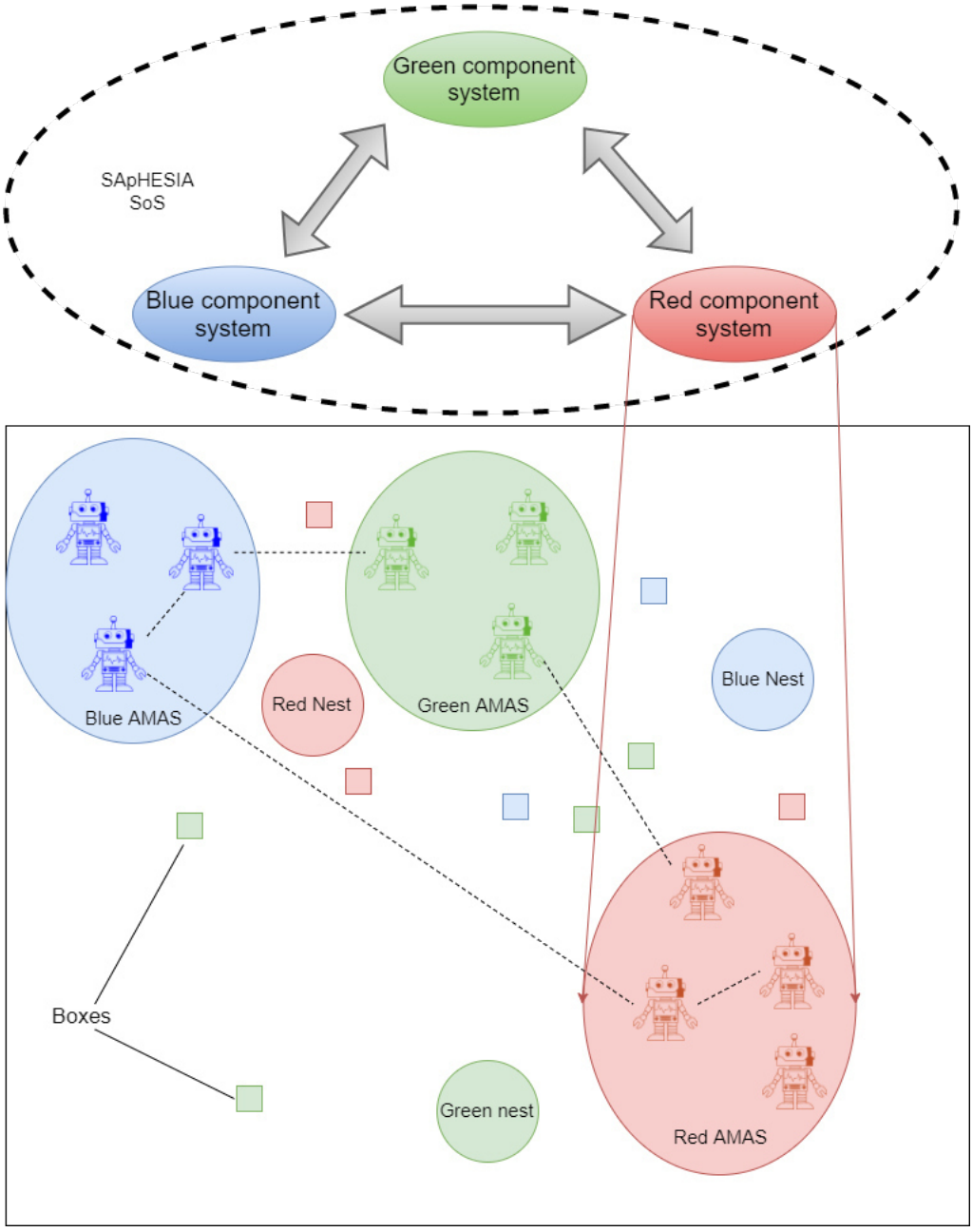


Figure 8.1 – CoCaRo with SApHESIA model

Algorithme 8.2 : coop_request_processing() of agent r_j carrying b_k

```
1  if C_{r_j}(t) < C_{r_i}(t) then
2      if CA_{r_i}(t, bback(b_k)) < C_{r_i}(t) then
3          accept_exchange();
4          deposit(b_k);
5      end
6      else
7          refuse_exchange() ;
8      end
9  end
10 else
11     if CA_{r_j}(t, bback(b_k)) < C_{r_j}(t) then
12         refuse_exchange() ;
13     end
14     else
15         accept_exchange() ;
16         deposit(b_k);
17     end
18 end
```

b) SApHESIA Model Definition for CoCaRo

We propose three component systems representing each a group of robots of the same color. Each of these component systems is able to change the perceptions of its robots concerning box rewards if and only if another component system needs help. We illustrate our reasoning by considering the red component system S_r. S_r has the functionalities to change the perceptions about blue and green boxes for red robots. Thus, red robots may consider these boxes as much important as red ones for themselves. In this way, red robots may take blue and/or green boxes with the same priority that red ones and then have more chances to give them to other robots. At the SoS level, it is a way for the red component system to help green and/or blue ones when they are more critical.
Thereafter are the details of the model for the red component system
$S_r = \{T_r, R_r, Acq_r, L_r, F_r, G_r, Cost_r\}$ with:

- $T_r = Red$;
- $R_r = \{R_{Red}, R_{Blue}, R_{Green}, R_{TotRobot}, R_{Dying}, R_{RedReward}, R_{BlueReward}, R_{GreenReward}, R_{MaxReward}\}$;
- $Acq_r = \emptyset$;
- $L_r = \{S_b, S_g\}$;
- $F_r = \{F_{Contact}, F_{HelpBlue}, F_{HelpGreen}\}$;
- $F_{Contact} = \{\{R_{Dying} > R_{TotRobot}/2\} \rightarrow \{S_b.R(R_{red}) \mathrel{+}= 1, S_g.R(R_{red}) \mathrel{+}= 1\}, t_c, p_c\}$;
- $F_{HelpBlue} = \{\{R_{Blue} == 1\} \rightarrow \{R_{BlueReward} = R_{MaxReward}\}, t_{hb}, p_{hb}\}$;

- $F_{HelpGreen} = \{\{R_{Green} == 1\} \rightarrow \{R_{GreenReward} = R_{MaxReward}, t_{hg}, p_{hg}\}\}$;
- $G_r = \{R_{Dying} = 0\}$;
- $Cost_r = 0$.

R_{Red}, R_{Blue} and R_{Green} are the resources enabling component systems to "send a message" to other ones through $F_{Contact}$. $R_{TotRobot}$ is the total number of red robots. R_{Dying} is the number of red robots having a low level of energy; it evolves thanks to an environment rule defined hereafter. $R_{RedReward}$ is the energy level used for rewarding a red robot bringing back a red box (defined in the same way as system 1 and 2: $\frac{2}{3} * Max_{Ne}$). $R_{BlueReward}$ (respectively $R_{GreenReward}$) is the energy level used to reward a red robot bringing back a blue box (respectively a green box) defined in the same way as system 1 and 2: $\frac{1}{3} * Max_{Ne}$. $R_{MaxReward}$ is the maximum level of reward (defined in the same way as system 1 and 2: $\frac{2}{3} * Max_{Ne}$). $F_{Contact}$ is a functionality used when S_r has a lot of number of robots with a low level of energy ($R_{Dying} > R_{TotRobot}/2$). This threshold value is used to detect the moment when more than 50% of the group has a really low speed and then becomes really inefficient. In this case, $F_{Contact}$ enables to send a R_{Red} resource to S_b and S_g. $F_{HelpBlue}$ and $F_{HelpGreen}$ are functionalities enabling S_r to respectively help S_b and S_g by respectively changing the value of $R_{BlueReward}$ and $R_{GreenReward}$ to $R_{MaxReward}$. If, for example, a R_{Blue} resource is received by S_r, the perceived value of reward of the red robots concerning blue boxes ($R_{BlueReward}$) will change thanks to $F_{HelpBlue}$. Acq_r is empty because is not used in this SApHESIA model. Each component system has already two links with the two other component systems. For example, S_r has the following set of links $L_r = \{S_b, S_g\}$. S_b is the component system representing the group of blue robots and S_g the group of green robots. Then, the goal G_r represents the fact that each component system tries to avoid robot with low energy. Finally, to decide if a component system helps others, it simply computes its criticality thanks to goal G_r and uses the cooperative algorithm (presented in chapter 7) to take its decision. The cost of the different component systems are set to 0 because they are not used in this simulation.

The green component system is defined as $S_g = \{T_g, R_g, Acq_g, L_g, F_g, G_g, Cost_g\}$ with:

- $T_g = Green$;
- $R_g = \{R_{Red}, R_{Blue}, R_{Green}, R_{TotRobot}, R_{Dying}, R_{RedReward}, R_{BlueReward}, R_{GreenReward}, R_{MaxReward}\}$;
- $Acq_g = \emptyset$;
- $L_g = \{S_b, S_r\}$;
- $F_g = \{F_{Contact}, F_{HelpBlue}, F_{HelpRed}\}$;
- $F_{Contact} = \{\{R_{Dying} > R_{TotRobot}/2\} \rightarrow \{S_b.R(R_{Green}) \mathrel{+}= 1, S_r.R(R_{Green}) \mathrel{+}= 1\}, t_c, p_c\}$;
- $F_{HelpRed} = \{\{R_{Red} == 1\} \rightarrow \{R_{RedReward} = R_{MaxReward}\}, t_{hr}, p_{hr}\}$;
- $F_{HelpBlue} = \{\{R_{Blue} == 1\} \rightarrow \{R_{BlueReward} = R_{MaxReward}\}, t_{hb}, p_{hb}\}$;

- $G_g = \{R_{Dying} = 0\}$;
- $Cost_g = 0$.

The blue component system is defined as $S_b = \{T_b, R_b, Acq_b, L_b, F_b, G_b, Cost_b\}$ with:

- $T_b = Blue$;
- $R_b = \{R_{Red}, R_{Blue}, R_{Green}, R_{TotRobot}, R_{Dying}, R_{RedReward}, R_{BlueReward}, R_{GreenReward}, R_{MaxReward}\}$;
- $Acq_b = \emptyset$;
- $L_b = \{S_r, S_g\}$;
- $F_b = \{F_{Contact}, F_{HelpRed}, F_{HelpGreen}\}$;
- $F_{Contact} = \{\{R_{Dying} > R_{TotRobot}/2\} \rightarrow \{S_r.R(R_{Blue}) \mathrel{+}= 1, S_g.R(R_{Blue}) \mathrel{+}= 1\}, t_c, p_c\}$;
- $F_{HelpRed} = \{\{R_{Red} == 1\} \rightarrow \{R_{RedReward} = R_{MaxReward}\}\}$;
- $F_{HelpGreen} = \{\{R_{Green} == 1\} \rightarrow \{R_{GreenReward} = R_{MaxReward}\}, t_{hg}, p_{hg}\}$;
- $G_b = \{R_{Dying} = 0\}$;
- $Cost_b = 0$.

The following entities *RedRobot*, *BlueRobot*, *GreenRobot* of the environment are defined and used to save to the current level of energy (through the resource *Energy*) of the robot and if they have a low energy level (through the resource *Dying*). $Dying = 1$ means that the robot has a low energy level and is used in the rules $Rule_{Red}$, $Rule_{Blue}$ and $Rule_{Green}$ defined hereafter.

$RedRobot = \{T_{RedRobot}, R_{RedRobot}, Acq_{RedRobot}, L_{RedRobot}, F_{RedRobot}, G_{RedRobot}\}$ with:

- $T_{RedRobot} = RedRobot$;
- $R_{RedRobot} = \{Dying, Energy\}$;
- $Acq_{RedRobot} = \emptyset$;
- $L_{RedRobot} = \emptyset$;
- $F_{RedRobot} = \emptyset$;
- $G_{RedRobot} = \emptyset$.

The resource *Energy* is used to save the current level of energy of the robot. *Dying* is used to know if the robot have a low level of energy (used in the rules presented hereafter).

$BlueRobot = \{T_{BlueRobot}, R_{BlueRobot}, Acq_{BlueRobot}, L_{BlueRobot}, F_{BlueRobot}, G_{BlueRobot}\}$ with:

- $T_{BlueRobot} = BlueRobot$;

- $R_{BlueRobot} = \{Dying, Energy\}$;
- $Acq_{BlueRobot} = \emptyset$;
- $L_{BlueRobot} = \emptyset$;
- $F_{BlueRobot} = \emptyset$;
- $G_{BlueRobot} = \emptyset$.

$GreenRobot = \{T_{GreenRobot}, R_{GreenRobot}, Acq_{GreenRobot}, L_{GreenRobot}, F_{GreenRobot}, G_{GreenRobot}\}$ with:

- $T_{GreenRobot} = GreenRobot$;
- $R_{GreenRobot} = \{Dying, Energy\}$;
- $Acq_{GreenRobot} = \emptyset$;
- $L_{GreenRobot} = \emptyset$;
- $F_{GreenRobot} = \emptyset$;
- $G_{GreenRobot} = \emptyset$.

These entities are used in the three following rules to count the number of robots that have a low energy level:

$$Rule_{Red} = \{RedRobot.R(Energy) < Max_{N_e}/3, RedRobot.R(Dying) == 0\} \rightarrow \{S_r.R(R_{Dying}) \mathrel{+}= 1, RedRobot.R(Dying) \mathrel{+}= 1\}$$

$$Rule_{Blue} = \{BlueRobot.R(Energy) < Max_{N_e}/3, BlueRobot.R(Dying) == 0\} \rightarrow \{S_b.R(R_{Dying}) \mathrel{+}= 1, BlueRobot.R(Dying) \mathrel{+}= 1\}$$

$$Rule_{Green} = \{GreenRobot.R(Energy) < Max_{N_e}/3, GreenRobot.R(Dying) == 0\} \rightarrow \{S_g.R(R_{Dying}) \mathrel{+}= 1, GreenRobot.R(Dying) \mathrel{+}= 1\}$$

If, for example, the robot *RedRobot* has a low level of energy ($RedRobot.R(Energy) < Max_{N_e}/3$) and does not have already its resource *Dying* equal to 1, then the component system S_r will increment its resource R_{Dying}. With this instantiation, the red component system is able to send a resource with the functionality $F_{Contact}$ to blue and green one. It happens when at least 50% of the robots have their energy level under $Max_{N_e}/3$. Then, when blue and green components (S_b and S_g) receive the message (through the resource R_{Red}), S_g and S_b change the perception of their respective robots concerning the reward of red boxes. Finally, each blue and green robot starts to bring back in priority red boxes. This fact naturally increases the probability that a blue or green robot meet a red robot with a low energy and then exchange their boxes. Finally, S_b and S_g can use the same process with their own $F_{Contact}$ functionality.

8.3 Experimentations

The three systems have been implemented using GAMA [Drogoul et al., 2013]. GAMA is a platform to model and simulate large-scale multi-agent systems, easy to use, which integrates analysis and performance visualization tools. Each system is then assessed in terms of efficiency and robustness using three metrics representing the state of the system over time: the number of functional robots, the average energy level of the agents and the number of boxes present in the environment.

8.3.1 Description

The simulations for the 3 systems have the same initial conditions:

- the number of robots is set to 90, 30 for each color;
- the initial energy level (N_i) is set to 300;
- the maximum energy level (Max_{N_e}) is set to 300;
- the energy consumption per time ($conso$) is set to 1;
- perception and communication scopes of a robot are set to a radius of 3 squares around its current position;
- the number of boxes appearing in the environment is set to 1 every 3 time units;
- the initial placements of boxes and robots are randomly chosen but are the same for all simulations.

The two next sections present the obtained results that are discussed in a third section.

8.3.2 Experimentation 1: Equiprobability of Boxes Apparitions

In this experiment, the apparition of a new box in the environment is equiprobable (in term of color) and enables to model an environment without perturbations and each kind of robot (red, green and blue) has the same probability to survive during time. This simulation enables to evaluate the performance of the three systems.

In figures 8.2, 8.3 and 8.4 the black curves represent the system without cooperation, the light gray color curves represent the system with cooperation and the dark gray curves represent the SoS according to SApHESIA. The figure 8.2 presents the mean energy level of robots, the figure 8.3, the number of boxes in the environment and figure 8.4, the number of alive red robots in the environment. The presented results are the average on 10 simulations.

Figures 8.2 and 8.4 show that the mean energy and the number of alive robots of the cooperative system is 100% higher that the non-cooperative one. Furthermore, the SApHESIA system has globally a higher mean energy and a higher number of alive robots. The figure 8.3 shows that non-cooperative system has a number of boxes in the environment that increases with time, contrary to cooperative and SApHESIA one that converge around 30 boxes after cycle 1000.

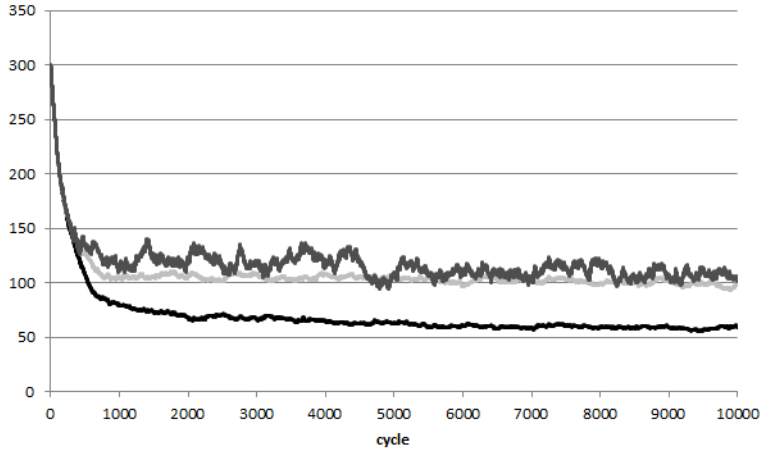


Figure 8.2 – Mean energy of alive robots (experimentation 1)

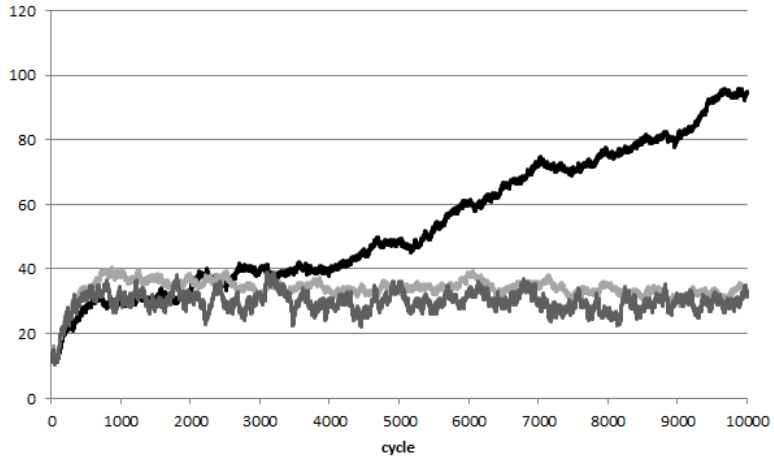


Figure 8.3 – Number of boxes in the environment (experimentation 1)

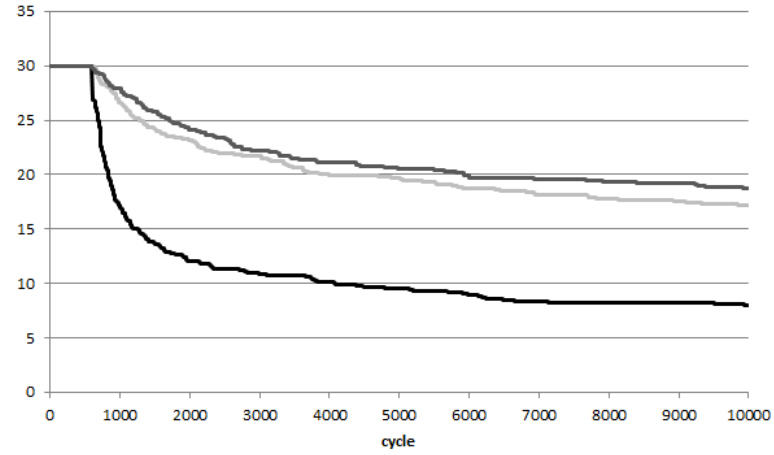


Figure 8.4 – Number of alive red robots (experimentation 1)

8.3.3 Experimentation 2: Wave of Boxes of the Same Color

The creation of box color within the environment is not equiprobable in this second experiment. Indeed, during 500 time units, only one box color is generated. At the end of 500 time units, another color is generated and so on. This 'hostile' environment enables to evaluate the robustness of the system because it is not intended to work on such a case. Indeed, the environment generates only a type of box and the robots have different rewards depending on the box color. Then, some groups of robots have more difficulties than other during a given period of time.

In figures 8.5, 8.6 and 8.7 the black curves represent the system without cooperation, the light gray color curves represent the system with cooperation and the dark gray curves represent the SoS according to SApHESIA. The figure 8.5 presents the mean energy level of robots, the figure 8.6, the number of boxes in the environment, the figure 8.7, the number of alive red robots in the environment. The results presented are the average on 10 simulations. Finally, figures 8.8 and 8.9 represent the average level of energy for each group of robot (red, green and blue) for respectively the system 1 (with no cooperation) and the system 2 (with cooperation).

The figure 8.5 shows that the mean energy of the cooperative system is around 50% higher that the non-cooperative one. Furthermore, the SApHESIA system has globally a higher mean energy. The figure 8.3 shows that non-cooperative system has a number of boxes in the environment that increases with time, contrary to cooperative and SApHESIA ones. The figure 8.6 shows that non-cooperative and cooperative systems have a number of boxes in the environment that increases with time, but more rapidly for the non-cooperative one, contrary to SApHESIA one that stabilizes around cycle 2000. The figure 8.7 shows that the number of red robots in hostile environment is 200% higher (respectively 300%) for cooperative system (respectively for SApHESIA) than in non-cooperative one. Finally, figures 8.8 and 8.9 show that in the hostile environment, the non-cooperative system has no red robots left after cycle 1000 and the cooperative one balances the mean energy levels of the different groups of robots.

8.3.4 Discussion

Our first hypothesis states that the criticality enables the cooperation between agents and improves the effectiveness and the robustness of an AMAS: the comparison of the three systems clearly shows that the exchange of criticality enables the AMAS to improve its effectiveness and its robustness. Concerning robustness, the underlying mechanism of cooperation allows a greater number of robots to survive in both experiments ('hostile' and classic environments) as we can see in figures 8.4 and 8.7. Then, the cooperation enables the AMAS to adapt itself to different kinds of dynamical environment. Figure 8.8 shows that the system without cooperation suffers more that the two others in the 'hostile' environment. Indeed, the number of red robots is equal to 0 after the 1000th cycle. Then, the non-cooperative system, by losing an entire type of agents, loses its diversity so its capacity to adapt to future changes in the environment. This is not the case for the cooperative and the SApHESIA systems: the figure 8.9 shows that the cooperative system tends to balance

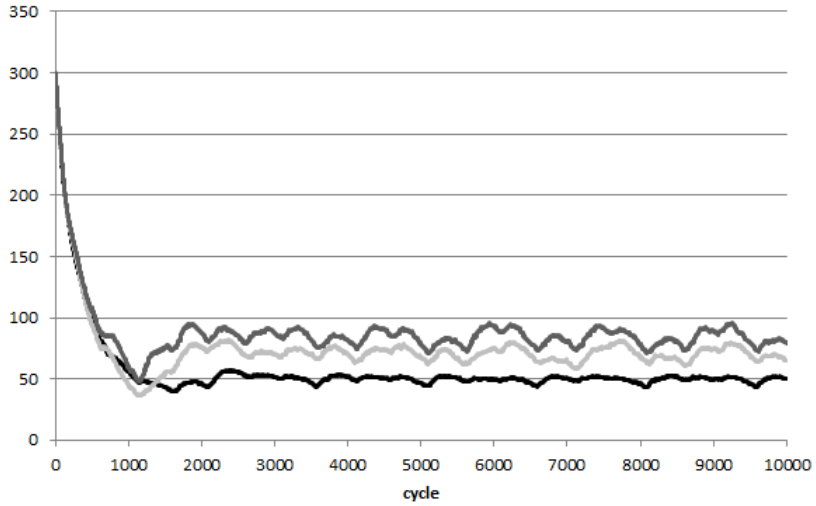


Figure 8.5 – Mean energy of alive robots (experimentation 2)

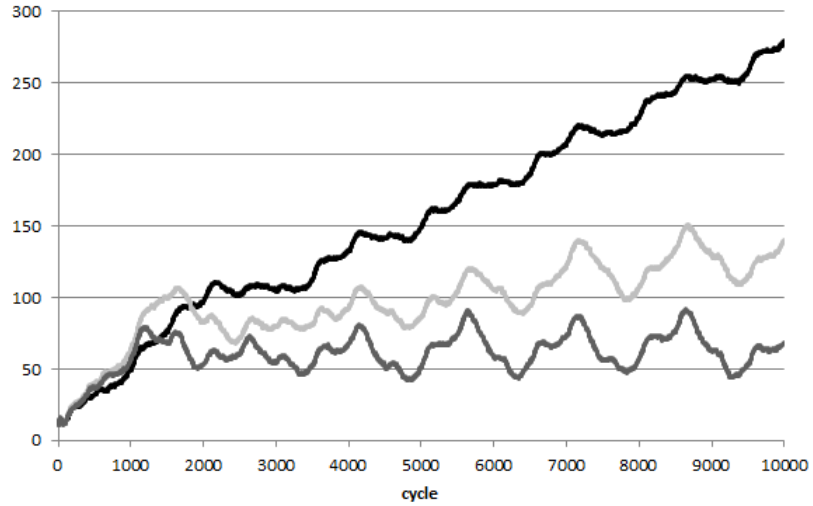


Figure 8.6 – Number of boxes in the environment (experimentation 2)

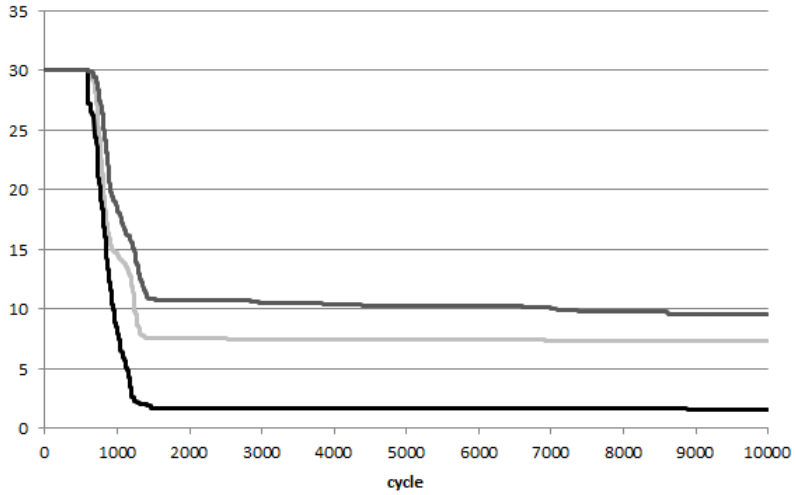


Figure 8.7 – Number of alive red robots (experimentation 2)

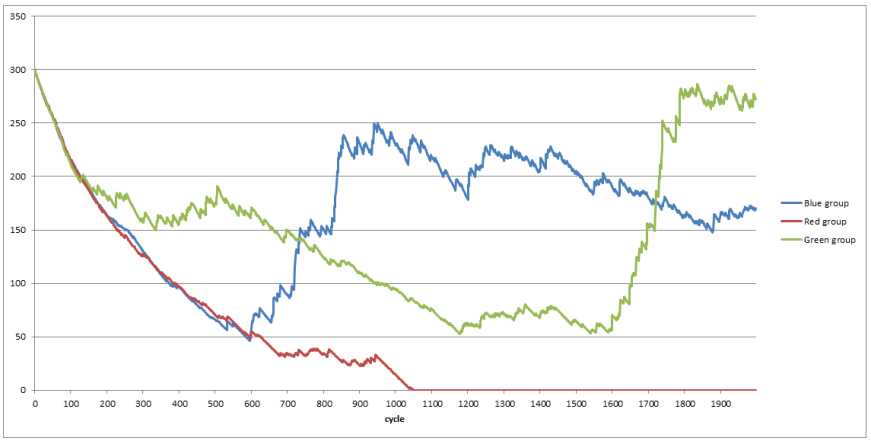


Figure 8.8 – Mean energy of red, green and blue group of robots **without** cooperation (system 1)

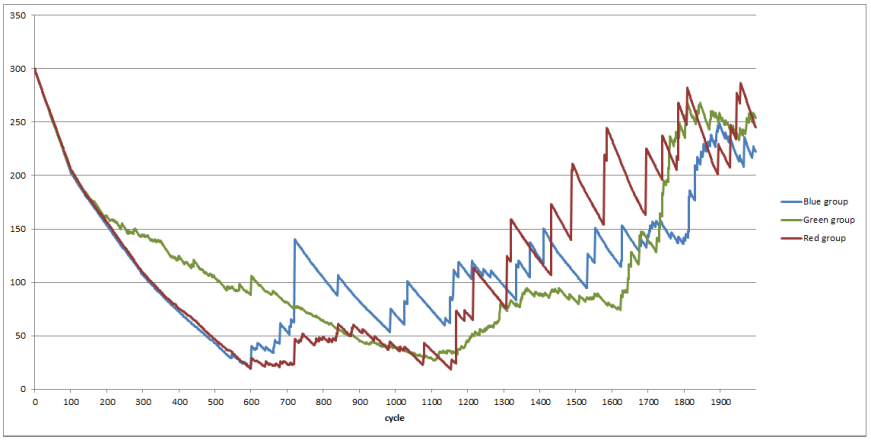


Figure 8.9 – Mean energy of red, green and blue group of robots **with** cooperation (system 2)

the difference level of energy: it does not lose an entire part of the agents and is then more able to adapt to future changes in the environment. Concerning the effectiveness of the three systems, the non-cooperative one has not enough robots left after cycle 4000 (figure 8.3) to stabilize the number of boxes in the environment contrary to the system with cooperation and with SApHESIA.

8.4 Conclusion

The study of these three systems through this two experimentations show that cooperation enables better effectiveness and robustness of the AMAS. Furthermore, the slightly better results with SApHESIA show that the use of SApHESIA component systems enables cooperation at a higher level, which improves the global performance of the system (number of boxes, energy level and number of alive red robots). These results legitimate the study of cooperation in SoS and are encouraging regarding the reification of an AMAS to a component system notably when an inter-AMAS cooperation will be addressed.

9 UAV Obstacle Avoidance System

This chapter presents another instantiation of SApHESIA concerning the formation flying of Unmanned Aerial Vehicles (UAVs). The aim of this experiment is to compare the effectiveness and the scalability of our formation heuristics with satisficing games presented in chapter 2. Satisficing games have been chosen because it is the basis of the collaborative formation which is another SoS architecting heuristic presented in chapter 2. To do that, the results obtained with our heuristic are presented with regards to results found in existing literature concerning the same UAV obstacle avoidance experiment presented in [Stirling and Frost, 2005].

9.1 Experiment Description

In this experiment, several UAVs have to fly through an environment that owns targets (T) and hazards (H). Each UAV has to avoid the hazards and go through the targets. It cannot go to the same place than another UAV and has to respect a maximum distance between itself and other UAVs. The six following points describe the problem in details:

1. The field of action consists of a grid divided into cells such that each target and each hazard are contained in one and only one cell. No cell may contain both a target and a hazard;
2. The vehicles move at constant forward velocity but variable lateral velocity in a three-abreast formation. The forward velocity is 1 cell per time unit. The lateral velocity is drawn from the set cells per time unit, where negative signifies a move ahead and to the left, zero a move straight ahead, and positive a move ahead and to the right. Each cell may be occupied by, at most, one vehicle (i.e., they cannot cross each other);
3. Each vehicle is able to detect all targets and hazards within a static distance of cells in the forward direction from their current cells, with unlimited lateral detection;
4. If a vehicle enters a cell that contains a target, the group of UAV scores one point;
5. If a vehicle enters a cell that contains a hazard, the group of UAV loses one point;
6. The goal of this problem is to cross the action field (the grid) by avoiding hazards and cross targets (i.e., to maximize the number of points).

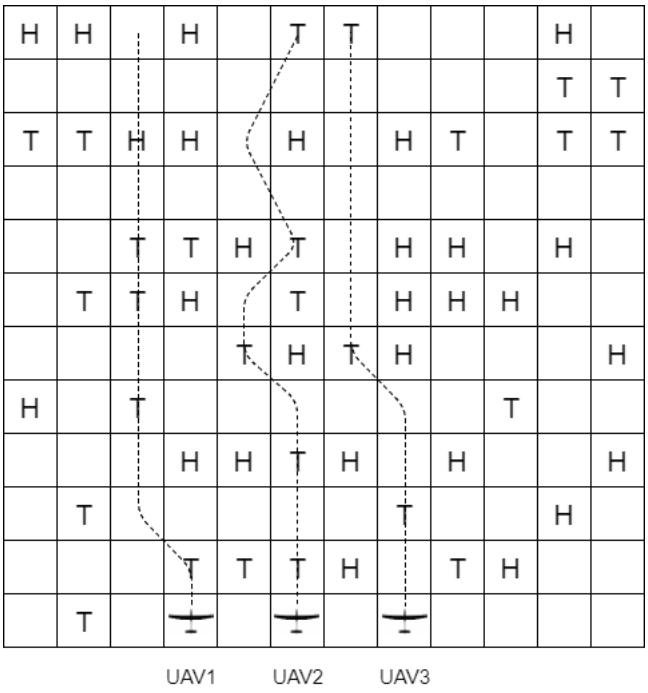


Figure 9.1 – Examples of UAVs trajectories

Figure 9.1 illustrates an example of this problem with three UAVs on the grid. H and T represent cells containing respectively Hazard and Target. The three lines represent the trajectories of 3 UAVs.

9.1.1 UAVs SApHESIA Model

We propose to instantiate this problem with SApHESIA and to solve it with our cooperative decision process. In this way, we propose the following SApHESIA model for the SoS composed of three UAVs:

$$SoS = \{UAV_1, UAV_2, UAV_3\}$$

Then, each UAV UAV is a component system defined as :

$$UAV = \{T, R, Acq, L, F, G, Cost\}$$

a) UAV Type and Resources

Each UAV has the following type $T = UAV$ and the following set of resources R:

$R = \{X, Y, ClosestTarget_X, ClosestTarget_Y, ClosestHazard_X, ClosestHazard_Y,$
$ClosestEmpty_X, ClosestEmpty_Y, ClosestTarget, ClosestHazard, ClosestEmpty, FieldOfView\}$

With:

- X, Y representing its position;

- $ClosestTarget_X$, $ClosestTarget_Y$, $ClosestHazard_X$, $ClosestHazard_Y$, $ClosestEmpty_X$, $ClosestEmpty_Y$, representing the coordinates of the closest target, hazard and empty cells;
- $ClosestTarget, ClosestHazard, ClosestEmpty$ representing the Euclidean distance of the closest target, hazard and empty cell;
- $FieldOfView$ representing the maximum distance of an UAV to detect targets and hazards.

b) UAV Acquaintances and Links

Each UAV contains the following set of acquaintances $Acq = \emptyset$. Acq is empty because in this problem, each UAV has already links with other UAVs. Moreover, each UAV contains a set of links L that represents the two other UAVs. For example, UAV_1 has the set L_1 defined as:

$$L_1 = \{\{UAV_1, 1\}, \{UAV_2, 1\}\}$$

c) UAV Functionalities

Each UAV contains the set of functionalities F:

$$F = \{F_{X_+}, F_{X_-}, F_{X_0}\}$$

With:

- $F_{X_+} = \{\{UAV.X \neq X + 1\} \rightarrow \{X \mathrel{+}= 1\}, \{Y \mathrel{+}= 1\}, t_{X+}, p_{X+}\}$;
- $F_{X_-} = \{\{UAV.X \neq X - 1\} \rightarrow \{X \mathrel{+}= -1\}, \{Y \mathrel{+}= 1\}, t_{X-}, p_{X-}\}$;
- $F_{X_0} = \{\{\emptyset\} \rightarrow \{X \mathrel{+}= 0\}, \{Y \mathrel{+}= 1\}, t_{X_0}, p_{X_0}\}$;
- $t_{X+} = t_{X-} = t_{X_0} = 1$;
- $p_{X+} = p_{X-} = p_{X_0} = 1$;

The different execution times (time to execute of the functionalities) and performances (probability of the functionalities to success) $t_{X+}, t_{X-}, t_{X_0}, p_{X+}, p_{X-}, p_{X_0}$ are set to 1 because in this experiment they cannot fail and have the same time duration. These functionalities represent respectively a movement to the right, to the left and straight forward. As we can see in the conditions of F_{X_+} and F_{X_-}, the UAVs cannot collide with each other ($\{UAV.X \neq X + 1\}$ and $\{UAV.X \neq X - 1\}$).

d) UAV Goals and Cost

Each UAV has the set of goals G:

$$G = \{G_{H_X}, G_{T_X}, G_{E_X}, G_X\}$$

With:

- $G_{H_X} = \{ClosestHazard_X \neq X, 1\}$
- $G_{H_Y} = \{ClosestHazard_Y \neq Y, 1\}$
- $G_{T_X} = \{ClosestTarget_X = X, 2\}$
- $G_{T_Y} = \{ClosestTarget_Y = Y, 2\}$
- $G_{E_X} = \{ClosestEmpty_X = X, 1\}$
- $G_{E_Y} = \{ClosestEmpty_Y = Y, 1\}$
- $G_X = \{UAV.X \neq X, 1\}$.

These goals represent respectively the fact that an UAV tries to avoid hazard (G_{H_X}, G_{H_Y}), to reach targets (G_{T_X}, G_{T_Y}) and empty cells (G_{E_X}, G_{E_Y}). For example, the goal G_{H_X} represents that the UAV wants its resource X (representing its x coordinate) to have a different value from its resource $ClosestHazard_X$ (representing the x coordinate of the closest hazard perceived by the UAV thanks to the rule $Rule.2$ presented hereafter). It enables the UAV to avoid the closest hazard. The goal G_X represents that the UAV wants its resource X to have a different value from the resource X of the other UAVs ($UAV.X$). It enables the UAVs to not be to close from each other. The priority of goals have been set to 1, except for G_{T_X} and G_{T_Y} that have been set to 2. It enables to prioritize the destination of the UAVs by going on target cells rather than empty cells. The *Cost* is set to 1 and is unused is this simulation.

e) Environment Model

In this simulation, there are three kinds of entities: Hazard, Target and Empty cells. These entities are passive (i.e. they do not have functionalities) and static. They only contain their positions represented by resources. Environment rules enable to model the perceptions of new targets, hazards and empty cells. For example, the rule 1 enables to detect the closest target by checking:

- if the distance between the target T and the UAV are under the value $FieldOfView$ with the following condition $\sqrt{(UAV.X - T.X)^2 + (UAV.Y - T.Y)^2} < UAV.FieldOfView$;
- if this target T is the closest target known by the UAV by checking if the distance between the target T and the UAV are under the value $ClosestTarget$ with the following condition $\sqrt{(UAV.X - T.X)^2 + (UAV.Y - T.Y)^2} < UAV.ClosestTarget$.

If these conditions are verified, the UAV will save the new target coordinates thanks to resources $ClosestTarget_X$, $ClosestTarget_Y$ and the Euclidean distance thanks to $ClosestTarget$. Finally, the rules 2 and 3 have the same functioning for hazard and empty cells.

Rule 1: Detect the "closest target"

$$\sqrt{(UAV.X - T.X)^2 + (UAV.Y - T.Y)^2} < UAV.ClosestTarget$$
$$\sqrt{(UAV.X - T.X)^2 + (UAV.Y - T.Y)^2} < UAV.FieldOfView$$
$$\rightarrow$$
$$UAV.ClosestTarget_X = T.X$$
$$UAV.ClosestTarget_Y = T.Y$$
$$UAV.ClosestTarget = \sqrt{(UAV.X - T.X)^2 + (UAV.Y - T.Y)^2}$$

Rule.2: Detect the "closest hazard"

$$\sqrt{(UAV.X - H.X)^2 + (UAV.Y - H.Y)^2} < UAV.ClosestHazard$$
$$\sqrt{(UAV.X - H.X)^2 + (UAV.Y - H.Y)^2} < UAV.FieldOfView$$
$$\rightarrow$$
$$UAV.ClosestHazard_X = H.X$$
$$UAV.ClosestHazard_Y = H.Y$$
$$UAV.ClosestHazard = \sqrt{(UAV.X - H.X)^2 + (UAV.Y - H.Y)^2}$$

Rule.3: Detect the "closest empty cell"

$$\sqrt{(UAV.X - E.X)^2 + (UAV.Y - E.Y)^2} < UAV.ClosestEmpty$$
$$\sqrt{(UAV.X - E.X)^2 + (UAV.Y - E.Y)^2} < UAV.FieldOfView$$
$$\rightarrow$$
$$UAV.ClosestEmpty_X = E.X$$
$$UAV.ClosestEmpty_Y = E.Y$$
$$UAV.ClosestEmpty = \sqrt{(UAV.X - E.X)^2 + (UAV.Y - E.Y)^2}$$

9.1.2 Results

To compare the effectiveness of our cooperative approach with [Stirling and Frost, 2005], 100 simulations with 3 UAVs in different environments have been done. To study its scalability of it, 4 other simulations are presented with respectively 8, 10, 15 and 20 UAVs. We present an example of a 3-UAVs simulation in figure 9.2 run with SApHESIA tools.

Table 9.1 – Result for 3-UAVs simulations

	Cooperative	Satisficing	Optimal
Mean	-1.54	1.44	2.95
Std deviation	3.17	4.62	2.58

Table 9.2 – Simulation time for 3, 8, 10, 15 and 20 UAVs simulations

	3-UAVs	8-UAVs	10-UAVs	15-UAVs	20-UAVs
Time (s)	4.44	8.15	9.91	15.05	18.08

For each simulation, the environment (a grid of 50x10 cells) is created with a probability of 0.1 for a target and 0.7 for a hazard to appear on each cell. Table 9.1 shows the mean score

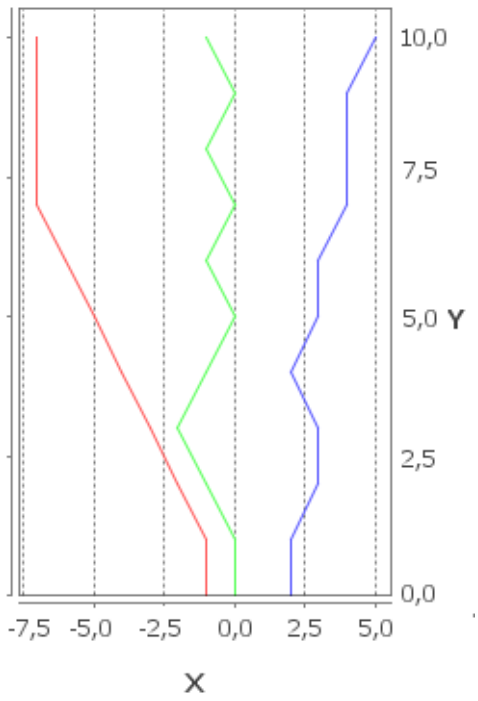


Figure 9.2 – Examples of cooperative UAVs trajectories

such as the standard deviation of 100 simulations for each approach (cooperative and satisficing) compared to the optimal. For each environment, we calculated the optimal trajectories of the 3 UAVS by searching the maximal score the 3 UAVs can reach for each simulation. The following parameters have been set for 3-UAVs simulations:

- $FieldOfView = 3$,
- $ClosestTarget = ClosestHazard = ClosestEmpty = 99$,
- $UAV_1.X = -1$ and $UAV_1.Y = 0$,
- $UAV_2.X = 0$ and $UAV_2.Y = 0$,
- $UAV_3.X = 2$ and $UAV_3.Y = 0$.

We present in table 9.2 the time duration of 3, 8, 10, 15 and 20 UAVs simulations.

9.1.3 Discussion

Obtained results show that even if cooperative approach is competitive, satisficing algorithm is slightly closest to the optimal regarding to the mean score for 3 UAVs. This difference comes from that our approach is less effective to prevent long-term difficulties.
Nevertheless, simulations with 8, 10, 15 and 20 UAVs show that our approach is scalable: time duration seems to evolve in $O(log(n))$ with n the number of UAVs. Indeed, the cooperation is a local heuristic approach more simple to implement than satisficing for the three following reasons:

- ▷ the satisficing approach needs the construction of the praxeic network and designers need to know all the interdependencies between all component systems before simulating this kind of application and it may be very difficult to represent them;
- ▷ computation of satisficing global function is resolved with the Pearl belief propagation algorithm that does not allowed cycles in praxeic graph [Stirling and Frost, 2005] (i.e. retro action loop);
- ▷ the cooperative approach does not require global function to calculate satisficing solutions, so it does not need to define a praxeic network;

The construction of praxeic network for 8 UAVs (and also for 10, 15 and 20) is complex because interdependencies between UAVs and possible retro-action loops. For example, UAV in (-1,1) has influences on (0,0) that has influences on (1,1) that has influences on (-1,1). These cycles forbid to use the Pearl belief propagation algorithm [Stirling and Frost, 2005] in order to solve the Bayesian network associated with praxeic network. Finally, in the satisficing approach, adding a new UAV leads to reconstruct the praxeic network so the global function. At the opposite, our approach enables to easily add or delete component system during functioning because it is not based on any graph.

9.2 Conclusion

This experimentation proposes a comparative evaluation of our cooperative approach for SoS architecting based on SApHESIA. We have instantiated, evaluated our approach to a UAV flight scenario and compared it to satisficing games used in another collaborative approach for SoS architecting [Caffall and Michael, 2009]. Simulations done with our approach shows competitive results with regards to the satisficing approach. Moreover, our approach goes through several limitations such as the definition and the computation of a global function during the design phase. Finally, last simulations with more UAVs show that our approach is easily scalable and enables interdependencies cycles that are really strong advantages for SoS architecting evolution.

10 Interdependence between two SoS: Combination of CoCaRo with UAVs

This chapter presents experimentations about reuse and interdependence between existing SoS. The aim is to validate that two SoS that have been independently built using SApHESIA can serve each other and then increase their efficiency. The first two sections present the two distinct SoS: the SoS version of CoCaRo and a slightly modified version of the UAV SoS presented in chapter 9. The last sections present the experimentation, the results and the discussion about them.

10.1 First SoS: CoCaRo

In this experimentation, we will reuse the component systems presented in chapter 8. The robots have been removed from the simulation and are now aggregated in component systems through functionalities defined in the next sections; so three component systems have been defined (red, blue and green). As the aim is to study the interdependence between SoS, the dynamic of exchange between robots are not needed in this simulation. Each component system consumes a certain rate of energy per time unit and is able to use boxes directly to create energy. The following sections describe entirely this SoS.

10.1.1 CoCaRo SoS

The SApHESIA model of the CoCaRo SoS is the following:

$$SoS = \{Red, Green, Blue\}$$

Where *Red*, *Green* and *Blue* are component systems representing a group of robots and are defined as $RG = \{T_r, R_r, Acq_r, L_r, F_r, G_r, Cost_r\}$ with:

- $T_r = \{Red, Green, Blue\}$;
- $R_r = \{R_{Turn}, R_{Robot}, R_{Energy}, R_{BoxRate}, R_{EC}, R_{Box}, R_{HighValue}, R_{LowValue}\}$;
- $Acq_r = \{\emptyset\}$;

- $L_r = \{RG\}$;
- $F_r = \{F_{CreateEnergy}, F_{GiveBox}\}$;
- $G_r = \{G_{Box}, G_{Energy}\}$;
- $Cost_r = 0$.

Concerning type T_r, this one can be equal to *Red*, *Green* or *Blue* representing respectively the red, green or blue group of robots.
Concerning resources R_r, R_{Turn} is used to save the current simulation step and is used in the rules of the environment presented hereafter. R_{Robot} represents the number of active robots in the group. As explain before, a robot is now just a resource that will be used in a functionality to create energy. By consequence, a robot is not an active entity anymore. R_{Energy} represents the global energy level of the robots group. It can be seen as the average level of energy of all robots in the group. $R_{BoxRate}$ represents the rate of boxes apparition in the robots group. In CoCaRo, the boxes appear randomly in the environment. R_{EC} represents the energy consumption of the robots group by turn. R_{Box} represents the number of available boxes in the robots group. It is used to model the number of boxes that are normally perceived and brought back by the robots group. Finally, $R_{HighValue}, R_{LowValue}$ are respectively the high reward value and the low reward value when the component system uses a box (in CoCaRo, the boxes give different rewards to robot groups depending on their color). All of these resources are used in functionalities and environment rules presented in the following sections.

Concerning Acq_r, this one starts empty and does not evolve over time because each group of robots are linked to others. For example, the set of links of *Red* is $L_r = \{Green, Blue\}$. Indeed, in CoCaRo, each robot can exchange boxes or information whatever their colors. Moreover, these links already exist in the SApHESIA version of CoCaRo (see chapter 8).

Concerning functionality F_r, $F_{CreateEnergy}$ represents the capacity of the robots group to change box to energy. The, the component system is able to create energy if it has one available box and one available robot ($\{R_{Box} \geqslant 1\}, \{R_{Robot} \geqslant 1\}$).

$$F_{CreateEnergy} = \{\{R_{Box} \geqslant 1\}, \{R_{Robot} \geqslant 1\}\} \rightarrow \{R_{Box} \mathrel{+}= -1\}, \{R_{Energy} \mathrel{+}= 1\}, t_c, p_c\}$$

t_c and p_c are set to 1 as for the other functionalities because in this experiment we consider that all the functionalities are the same time duration and cannot fail. $F_{GiveBox}$ represents the capacity of a robots group to give boxes to another robots group.

$$F_{GiveBox} = \{\{R_{Box} \geqslant 1\} \rightarrow \{\{RobotGroup\, R_{Box} \mathrel{+}= -1\}, t_g, p_g\}\}$$

t_g and p_g are set to 1.
Concerning goals G_r, G_{Box} represents the goal of the robots group (to have available boxes as much as possible). Formally, the goal is defined as $G_{Box} = \{R_{Box} > 0\}$. G_{Energy} represents the goal of the robot group to have as much energy as possible. Formally, the goal is defined as $G_{Energy} = \{R_{Energy} > 20\}$. 20 has been chosen arbitrarily, but a value superior to 0 is

important because if component system has no more energy, this one is considered as 'dead' and cannot be used anymore. It represents the fact that in CoCaRo, a robot with no more energy is 'dead' (i.e., it cannot be used anymore).

10.1.2 CoCaRo Environment

The SApHESIA model of the environment is the following:

$$E = \{Rule_{Apparition}, Ru_{GreenComsumption}, Ru_{BlueComsumption}\}$$

In the CoCaRo environment, there is no entity. The boxes are directly considered as resources. But, to simulate the boxes apparition and the energy consumption, four rules have been defined. First, the rule $Ru_{Apparition}$ enables to generate new boxes in the component systems $RobotGroup$. Basically, at each turn of the simulation, a random number is taken and if this one is under the resource $R_{Apparition}$, the resource R_{Box} is increased. Formally, the rule is defined as:

$$Ru_{Apparition} = \{\{100 \times random() < RG.R(R_{Apparition})\} \rightarrow \{RG.R(R_{Box}) \mathrel{+}= 1\}\}$$

Where $random()$ is a function giving a random number between 0 and 1. In chapter 8, we defined an experiment with an 'hostile' environment where the boxes that appear have all the same color during a fixed period of time. It enabled to test the robustness of CoCaRo by giving, during a fixed period of time, only boxes that are efficient for only one kind of robots. To reproduce this kind of environment, we add two rules that will change the energy consumption of each of the robots groups. The aim is to reproduce that each robots group has, during a fixed period of time, only boxes that are efficient for it. Then, during the first 1000 steps of the simulation, the *Red* robots group has a high value of energy consumption ($R_{EC} = R_{HighValue}$) (representing that there is no red boxes in the environment). Then, during the next 1000 steps of the simulation (step 1000 to 2000), the *Green* robots group will have a high value of energy consumption ($R_{EC} = R_{HighValue}$) and *Red* will come back to a low consumption (thanks to the rule $Ru_{GreenComsumption}$). Finally, during the last 1000 steps of the simulation (step 2000 to 3000), the *Blue* robots group will have a high value of energy consumption ($R_{EC} = R_{HighValue}$) and *Green* will come back to a low consumption (thanks to the rule $Ru_{BlueComsumption}$). These dynamics are represented through the two following rules:

$$Ru_{GreenComsumption} = \{\{Green.R(R_{Turn}) == 1000\}, \{Green.R(R_{EC}) == Green.R(R_{HighValue})\} \rightarrow \{Red.R(R_{EC}) = Red.R(R_{LowValue})\}$$

$$Ru_{BlueComsumption} = \{\{Blue.R(R_{Turn}) == 2000\}, \{Blue.R(R_{EC}) == Blue.R(R_{HighValue})\} \rightarrow \{Green.R(R_{EC}) = Green.R(R_{LowValue})\}\}$$

10.2 The Second SoS: UAVs

This section presents the second SoS used in this experimentation inspired from the SoS presented in chapter 9. This SoS is composed of three component systems UAVs that are able to convert them to energy such as robots group but also to navigate to boxes.

10.2.1 UAVs SoS

Formally, the SoS is defined as:

$$SoS_{UAV} = \{UAV_1, UAV_2, UAV_3\}$$

A component system UAV models a drone that detects, takes and converts box to energy. The aim of these UAVs is to get a level of energy sufficiently high to keep functioning during time. An UAV is able to give box to other UAVs if needed. The formal SAphESIA definition of a component system UAV representing a drone is the following: $UAV = \{T_u, R_u, Acq_u, L_u, F_u, G_u, Cost_u\}$ with:

- $T_u = UAV$;
- $R_u = \{R_{Energy}, R_{Box}, R_{DistBox}\}$;
- $Acq_u = \{\emptyset\}$;
- L_u that depends on the UAV;
- $F_u = \{F_{CreateEnergy}, F_{GiveBox}\}$;
- $G_u = \{G_{Box}, G_{Energy}, G_{DistBox}\}$;
- $Cost_u = 0$.

Concerning resources, R_{Energy} represents the energy level of the UAV. R_{Box} represents the number of boxes available for the UAV. Finally, to simplify the model of the UAV movement, the use of X and Y coordinates has been replaced by only one resource $R_{DistBox}$ that represents the distance between a box and the UAV.

Each UAV contains the following set of acquaintances $Acq = \emptyset$. Acq is empty because in this problem, each UAV has already links with other UAVs. Moreover, each UAV contains a set of links L that represents the two other UAVs. For example, UAV_1 has the set L_1 defined as:

$$L_{UAV_1} = \{\{UAV_1, 1\}, \{UAV_2, 1\}\}$$

Concerning functionality, $F_{CreateEnergy}$ represents the capacity of an UAV to change box to energy.

$$F_{CreateEnergy} = \{\{R_{Box} \geqslant 1\}, \{R_{Robot} \geqslant 1\} \rightarrow \{R_{Box} \mathrel{+}= -1\}, \{R_{Energy} \mathrel{+}= 1\}, t_c, p_c\}$$

t_c and p_c are set to 1. $F_{GiveBox}$ represents the capacity of an UAV to give box to another UAV.

$$F_{GiveBox} = \{\{R_{Box} \geqslant 1\} \rightarrow \{UAV, R_{Box} \mathrel{+}= -1\}, t_g, p_g\}$$

t_g and p_g are set to 1. $F_{MoveToBox}$ represents the capacity to go to a box. If the UAV has 0 box and is not on the box ($\{R_{Box} = 0\}, \{R_{DistBox} \neq 0\}$), it will go to the box.

$$F_{MoveToBox} = \{\{R_{Box} = 0\}, \{R_{DistBox} \neq 0\} \rightarrow \{R_{DistBox} \mathrel{+}= -1\}, t_m, p_m\}$$

t_m and p_m are set to 1. $F_{TakeBox}$ represents the capacity to take a box. If the UAV has 0 box and is on the box ($\{R_{Box} = 0\}, \{R_{DistBox} = 0\}$), it will take the box.

$$F_{TakeBox} = \{\{R_{Dist} = 0\}, \{R_{Box} = 0\} \rightarrow \{R_{Box} \mathrel{+}= 1\}, t_t, p_t\}$$

t_t and p_t are set to 1.

Concerning goals, G_{Box} represents the goal of the UAV to have as much as possible available boxes. Formally, the goal is defined as $G_{Box} = \{R_{Box} > 0\}$. G_{Energy} represents the goal of the UAV to have as much as possible energy. Formally, the goal is defined as $G_{Energy} = \{R_{Energy} > 0\}$. Finally, $G_{DistBox}$ represents the goal of the UAV to minimize its distance to a box. Formally, the goal is defined as $G_{DistBox} = \{R_{DistBox} == 0\}$.

10.2.2 UAVs Environment

In this environment, boxes are entities. Such an entity is defined as $Box = \{T_b, R_b, Acq_b, L_b, F_b, G_b\}$ with:

- $R_b = \{R_{Availability}\}$;
- $Acq_b = \{\emptyset\}$;
- $L_b = \{\emptyset\}$;
- $F_b = \{\emptyset\}$;
- $G_b = \{\emptyset\}$.

A box is a simple entity where $R_{Availability}$ represents the availability of the box for the UAVs. This resource is used as a threshold to simulate the detection of a box by an UAV in rule $Rule_{DetectBox}$. Formally, the rule is defined as:

$$Ru_{DetectBox} = \{100 \times random() < Box.R_b(R_{Availability}) \rightarrow UAV.R_u(R_{DistBox}) = Box.R_b(R_{Availability})\}$$

Then, if the condition $100 \times random() < Box.R_b(R_{Availability})$ is verified, then the UAV has detected this box and save its distance in its resource $R_{DistBox}$.

10.3 Experimentations

This section describes the experimentations concerning the CoCaRo SoS, the UAV SoS and the union of the two SoS, particularly the values of the initial parameters, the number of component systems and their initial links.

Table 10.1 – Initialization of the three robots groups

	Green	*Blue*	*Red*
R_{Robot}	20	20	20
R_{Energy}	60	60	60
$R_{BoxRate}$	1	1	1
R_{EC}	$R_{LowValue}$	$R_{LowValue}$	$R_{HighValue}$
R_{Box}	0	0	0
R_{Turn}	0	0	0

10.3.1 CoCaRo SoS Experimentation

In this version of CoCaRo, the robots are not active entities; their actions (bringing back boxes and convert boxes into energy) are modeled with the functionality $F_{CreateEnergy}$. The aim of this first CoCaRo SoS experimentation is to show that the dynamics of this system (without active robots) is the same than the system CoCaRo with cooperation presented in chapter 8 (with active robots).

a) Description

For the CoCaRo SoS, three component systems *RobotGroup* have been instantiated: *Green*, *Red*, *Blue*. Each of them has been initialized with the different values that are summed up in the table 10.1. In this table, $R_{LowValue} = 0.015$ and $R_{HighValue} = 0.04$. This value has been chosen arbitrarily. The idea is to begin with the *Red* robots group with a high value of energy consumption and validate that the two other robots groups will help it by giving extra boxes.

b) Results

First, figure 10.1 shows (through the graph R_{Energy}) that the dynamics of the environment impacts the functioning of each robots group during time. For example, from cycle 0 to cycle 1000, the R_{Energy} resource is lower for the *Red* component system because of the initialisation of R_{EC} to $R_{HighEnergy}$. This is the same reasoning for *Green* and *Blue* respectively between cycle 1000 and 2000 and between cycle 2000 and 3000. Secondly, the dynamics of the system CoCaRo with cooperation presented in chapter 8 is always present. Indeed, each robots group tends to balance the mean level of energy of each other. This fact can be seen with the R_{Energy} and criticality graph. Even if a robots group is more constrained by the environment, the two others, thanks to the cooperation algorithm, are able to help it. Thirdly, figure 10.1 shows that all the robots groups finish the simulation by having a very low level of energy.

10.3.2 UAVs SoS Experimentation

The aim of this UAV SoS experimentation is to present the global functioning of an independent SoS using the cooperation mechanism. UAVs can cooperate each other by using the

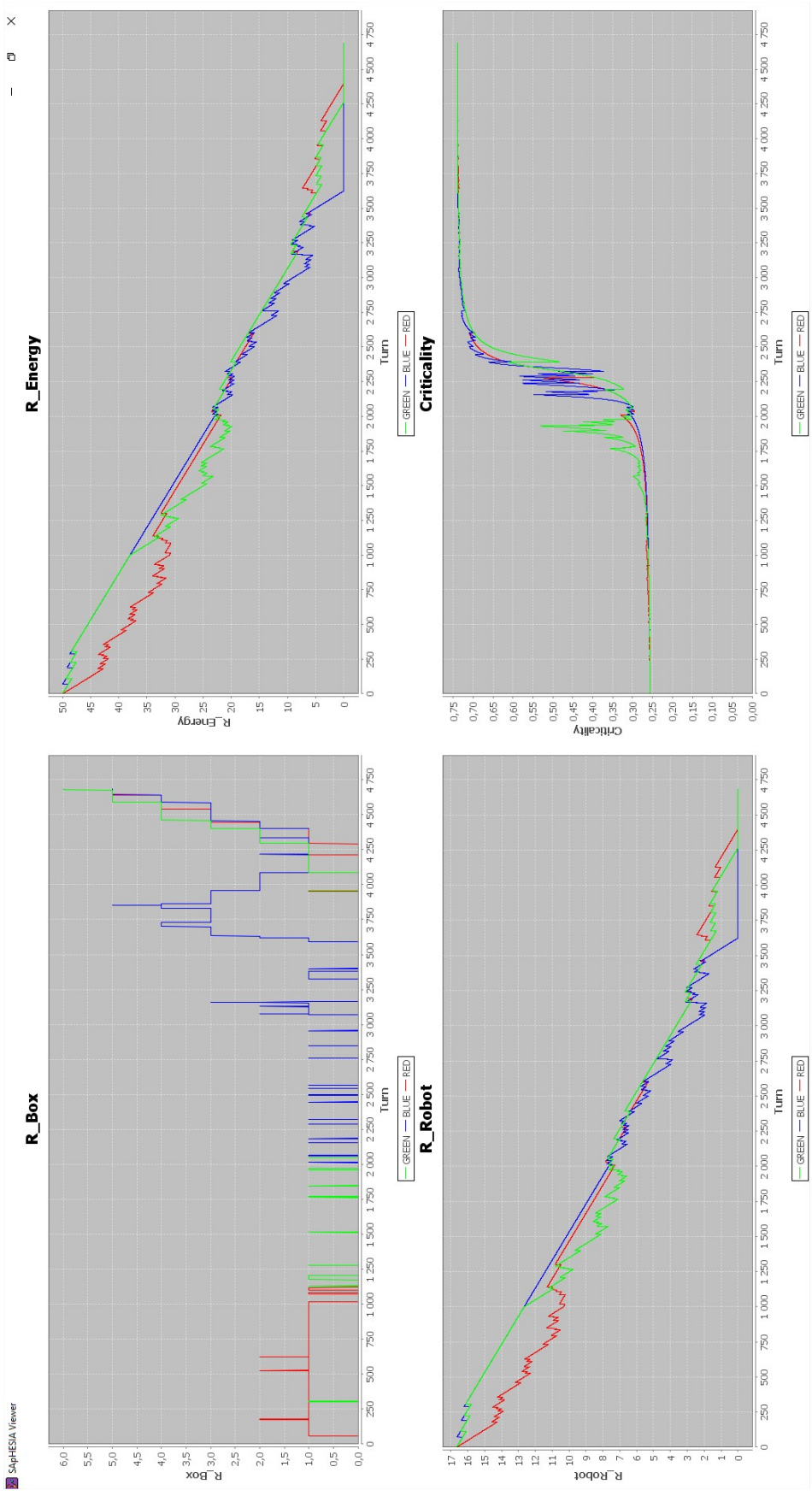


Figure 10.1 – Results of the simulations of CoCaRo SoS

Table 10.2 – Initialization of the three UAVs

	$UAV1$	$UAV2$	$UAV3$
R_{Energy}	60	60	60
$R_{BoxRate}$	1	1	1
R_{EC}	0.08	0.015	0.015
R_{Box}	0	0	0

$F_{GiveBox}$ functionality. For the UAV SoS, three component systems UAV have been instantiated: UAV_1, UAV_2 and UAV_3.

a) Description

Each of the component system UAV has been initialized with the different values that are summed up in the table 10.2. The value have been chosen arbitrarily. Then, UAV_1 is a less effective than the others. This lack of effectiveness enables to add some diversities in the SoS and validate that the cooperation algorithm is functioning in this SoS.

b) Results

Figure 10.2 shows the results of the experimentation for the UAV SoS. The level of energy of $UAV1$ decreases quicker than the others (because of the higher value of R_{EC}). Because of the goal G_{Energy}, a quick increase of the $UAV1$ criticality from cycle 0 to cycle 650 is visible. But, after cycle around 650, the cooperation algorithm makes $UAV2$ and $UAV3$ help $UAV1$ by giving boxes. The result of this cooperative behavior is the increase of the criticality of $UAV2$ and $UAV3$, because they do not use their boxes for creating energy anymore. But, a stabilization of the level of energy of $UAV1$ is visible from cycle around 650. Then, the levels of energy of $UAV2$ and $UAV3$ are decreasing until cycle 5000 because these two UAVs continue to help $UAV1$ to survive. Finally, after cycle 5000 the criticality and the energy level are balanced and stable for the three component systems. The final stabilization comes once again from cooperation algorithm. Indeed, as $UAV1$ shows that it has the same level of criticality than other, it starts to bring back boxes too and share it with others UAVs. Finally, we validate that in this case the UAV SoS is able to adapt itself through the cooperation.

10.3.3 CoCaRo + UAVs SoS

This final experimentation validates that two interdependent SoS are able to reach a better functioning in term of performance by using cooperation. To do that, the two previous SoS are slightly modified to create dependencies in terms of functionalities. The aim is to design the SoS composed of $RobotGroup$ more efficient concerning the transformation of boxes into energy and the SoS composed of UAV more efficient concerning the detection and the transport of the boxes. Then, this experiment will show if the two SoS,thanks to our cooperative heuristic, are able to naturally take advantage of this interdependence.

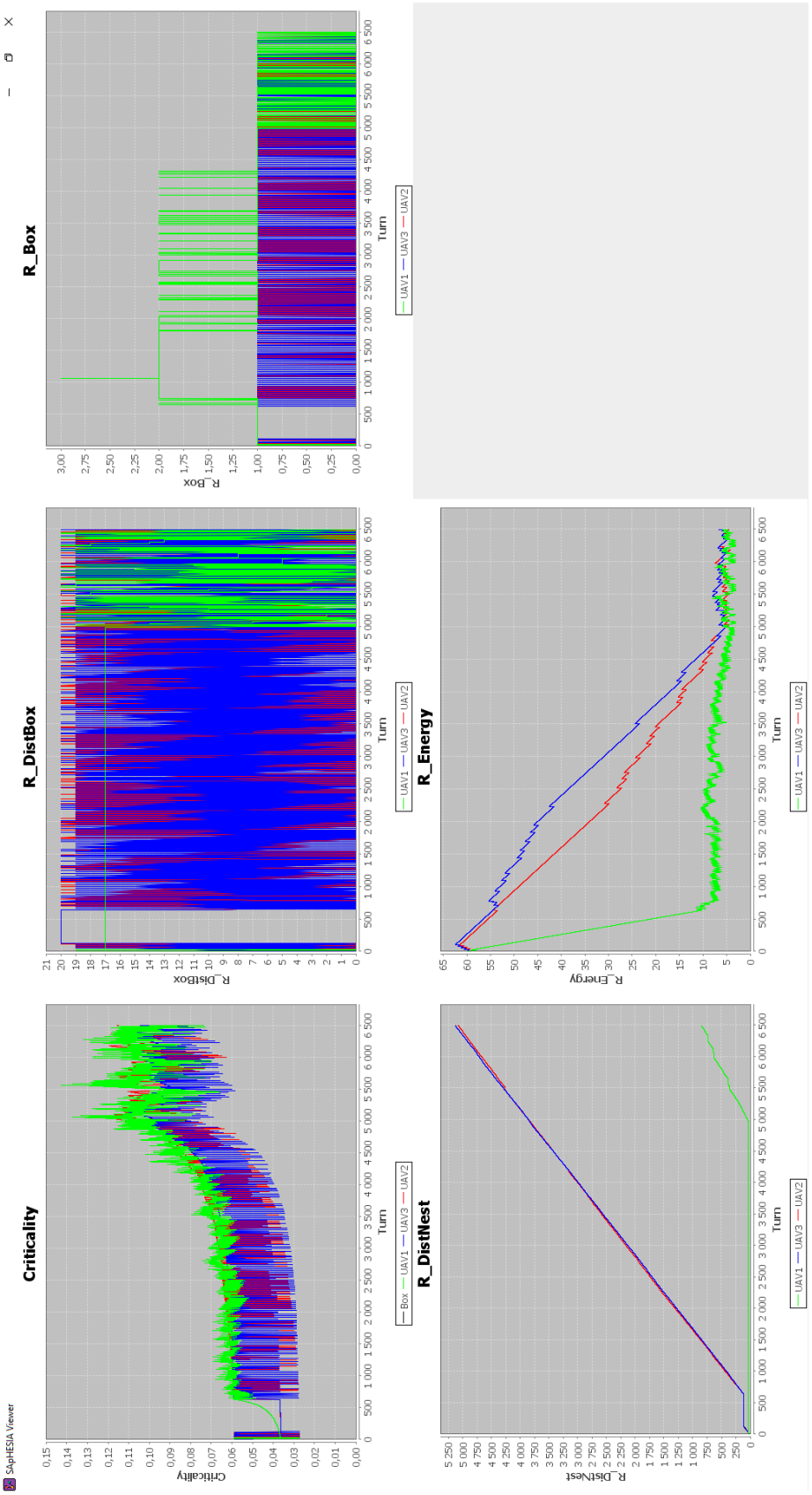


Figure 10.2 – Results of the simulations of UAVs SoS

a) Description

These two SoS are close to the two described in the previous sections:

- 3 UAVs have been instantiated in the SoS $SoS_{UAV} = \{UAV_1, UAV_2, UAV_3\}$;
- 3 robots groups have been instantiated in the $SoS_{RG} = \{Red, Green, Blue\}$.

The UAV component system has been modified to create interdependences between UAV and $RobotGoup$. An UAV is then defined as $UAV = \{T_u, R_u, Acq_u, L_u, F_u, G_u, Cost_u\}$ with:

- $T_u = UAV$;
- $R_u = \{R_{Energy}, R_{Box}, R_{DistBox}, R_{DistToRG}, R_{DistToBox}\}$;
- $Acq_u = \{\emptyset\}$;
- L_u that depends on the UAV;
- $F_u = \{F_{CreateEnergy}, F_{GiveBox}, F_{MoveToBox}, F_{TakeBox}, F_{MoveToRG}, F_{Drop}\}$;
- $G_u = \{G_{Box}, G_{Energy}, G_{DistBox}\}$;
- $Cost_u = 0$.

$R_{DistToRG}$ and $R_{DistToBox}$ have been added in order to respectively represent the distance to a $RobotGroup$ and the distance to a Box. Concerning functionalities, the UAV is now able to share boxes with a $RobotGroup$ thanks to four new functionalities $F_{MoveToBox}$ $F_{TakeBox}$, $F_{MoveToRG}$ and F_{Drop}. $F_{MoveToBox}$ enables to an UAV to go to a Box by decreasing the resource $R_{DistToBox}$. $F_{TakeBox}$ enables an UAV to take a box when $R_{DistToBox}$ is equal to 0. $F_{MoveToRG}$ enables to an UAV to go to a $RobotGoup$ by decreasing the resource $R_{DistToRG}$. Then, when $R_{DistToRG}$ is equal to 0, F_{Drop} enables UAV to give a box to $RobotGroup$. To create dependencies in both ways (i.e., interdependencies), the use of F_{Drop} also rewards the UAV by giving extra R_{Energy}. F_{Drop} has been designed to make $RobotGroup$ more efficient to transform boxes into energy. In this way, it is more efficient for the two SoS that the UAVs bring back boxes to the $RobotGroup$s. Formally, the four added functionalities are defined as:

$$F_{MoveToBox} = \{\{R_{DistToBox} \neq 0\}, \{R_{Box} == 0\} \rightarrow \{R_{DistToRG} \mathrel{+}= -1\}, t_{mb}, p_{mb}\}$$

$$F_{TakeBox} = \{\{R_{DistToBox} == 0\}, \{R_{Box} == 0\} \rightarrow \{UAV, R_{Box} \mathrel{+}= 1\}, t_{tb}, p_{tb}\}$$

$$F_{MoveToRG} = \{\{R_{DistToRG} \neq 0\}, \{R_{Box} \geqslant 1\} \rightarrow \{R_{DistToRG} \mathrel{+}= -1\}, t_m, p_m\}$$

$$F_{Drop} = \{\{R_{DistToRG} = 0\}, \{R_{Box} \geqslant 1\} \rightarrow \{UAV, R_{Box} \mathrel{+}= -1\}, \{RG, R_{Box} \mathrel{+}= 1\}, t_d, p_d\}$$

$t_{mb}, p_{mb}, t_{tb}, p_{tb}, t_m, p_m, t_d$ and p_d are set to 1. The environment is composed of the union of the two previous environments:

$$E = \{Rule_{Apparition}, Ru_{GreenComsumption}, Ru_{BlueComsumption}, Boxes\}$$

Where $Rule_{Apparition}$, $Ru_{GreenComsumption}$ and $Ru_{BlueComsumption}$ are the rules presented in the CoCaRo SoS experiment (see section 10.1.2) and $Boxes$ are a set of entities of type Box presented in the UAVs experiment (see section 10.2.2) .

b) Results

Figure 10.3 shows the results of 8000 cycles of simulation. We see that the R_{Energy} of the *RobotGroup*s tends to be balanced between each different *RobotGroup*s (as the previous CoCaRo SoS simulation). But contrary to the previous simulation with the CoCaRo SoS, this resource is stable at an earlier stage of the simulation and is increasing after the cycle 1000. This is due to a high criticality of the *RobotGroup*s at the beginning of the simulation leading the UAVs to give extra boxes thanks to their new functionality F_{Drop} and their cooperative behavior. As a result, the three *RobotGroup*s are able to function during the 8000 cycles of simulation. Furthermore, thanks to the extra energy given by the F_{Drop} functionality, the UAVs have a level of energy (represented also by R_{Energy}) that is equivalent to the one in the experiment with the UAV SoS alone. Finally, the less efficient component systems (that are the UAV_1 and the *Red* robots group, because they have been define with a higher R_{EC} than others) are helped more than the other thanks to the cooperation mechanism. Indeed, their level of energy are equivalent to the others.

10.4 Conclusion

In this chapter, two SoS have been instantiated independently and have been modified to create interdependencies between each other. Our experiments have shown that the two SoS tend to find naturally the best way to solve the constraints of the environment. In the last experiment, the UAVs have brought back boxes to *RobotGroup*s, thanks to their cooperative behavior. Indeed, UAVs have also increased their energy because giving boxes to *RobotGroup* rewards them with extra energy. Then, the two SoS have naturally found their interdependencies and a way to solve them. Our cooperative heuristic through the criticality comparison enables to solve in generic way interdependencies problems without re-design them such as [Lemouzy, 2011] has to do. In a more generic way, we think that the criticality comparison can be a solution to solve interdependencies of systems.

Nevertheless, the goals definitions of the different component system has to be made with attention. For example, some tests have been made by changing *RobotGroup* goals about R_{Energy} by the following : $G_{Energy} = \{R_{Energy} > 0\}$ instead of $G_{Energy} = \{R_{Energy} > 20\}$. In this case, the criticality is really high only when $R_{Energy} is closeto 0$, then, other component systems start to help only when $R_{Energy} is closeto 0$ so the cooperation between the UAVs and the robots groups starts too late to be useful.

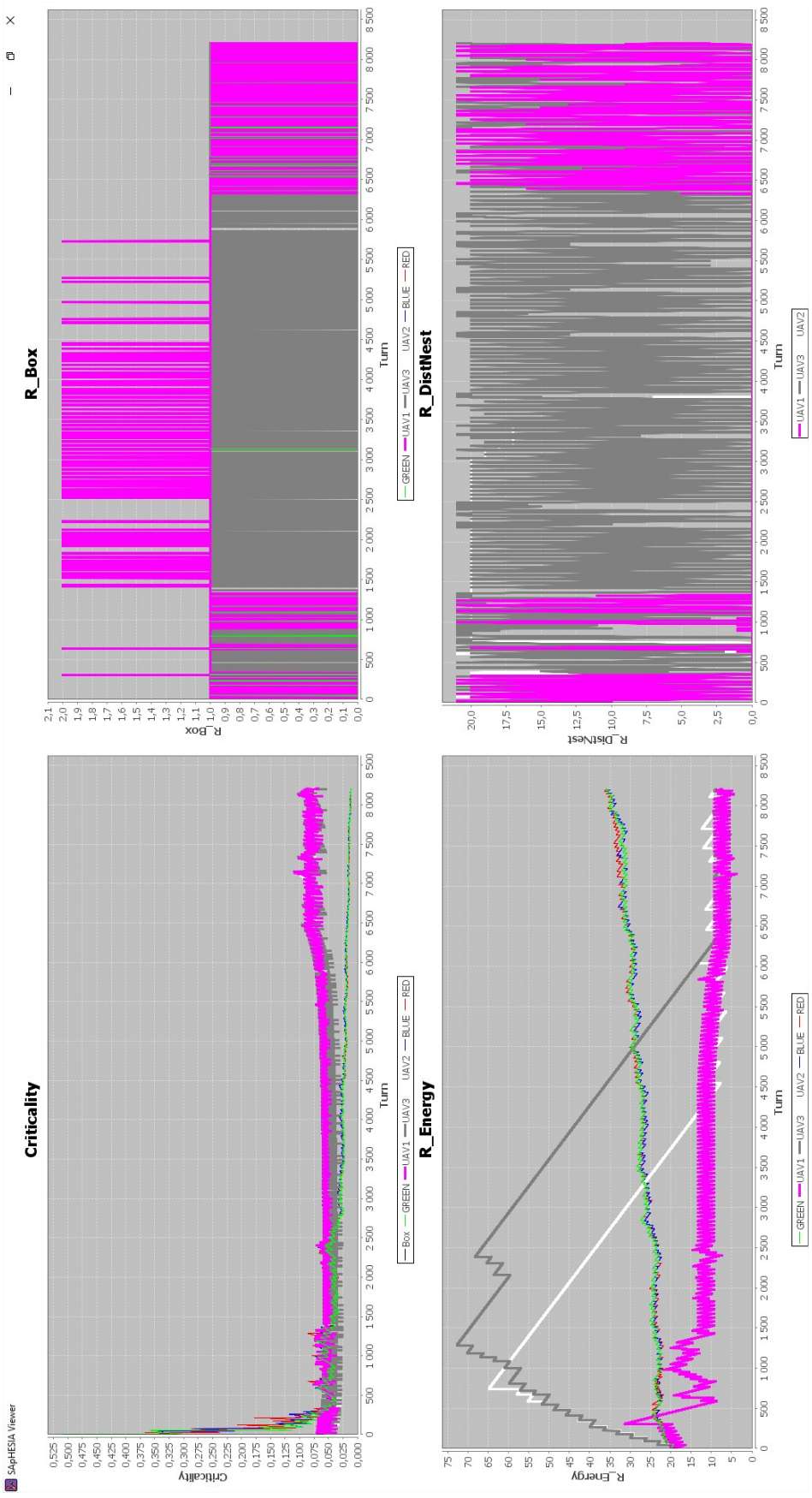


Figure 10.3 – Results of the simulations of UAVs and CoCaRo SoS

11 Self-Adaptation of the Criticality

This final chapter proposes to improve the cooperative algorithm proposed in chapter 5. We propose a new algorithm called the **criticality adaptation algorithm** used at the component system level to solve problems concerning the use of the criticality when trying to unify two or more SoS. This algorithm is evaluated through an experiment dealing with product manufacturing.

11.1 The Problem of the Criticality Adaptation in the AMAS approach

The problem of the criticality adaptation in the AMAS approach appears because the problems to solve are becoming more and more complex and because of a strongly dynamic environment in which the system evolves (mainly because unexpected events may occur during its functioning) [Lemouzy, 2011]. In this case, several systems (possibly designed by different designers) may have to interact to solve a particular problem. It leads to the need of putting together two or more AMAS that have been independently designed but are interdependent. But, the AMAS approach does not deal with how to manage interactions between agents coming from different AMAS, notably concerning the use of the criticality. Indeed, the criticality of an AMAS A can have a different meaning than the criticality of an AMAS B. Indeed, as the two AMAS have been independently designed, there is no reason that, for example, the value of criticality 0.7 has the same meaning in term of difficulty for A and for B. This issue is called the normalization of the criticalities. [Lemouzy, 2011] pointed out that, creating a second AMAS in an environment where a first AMAS is already existing can lead to **re-design** (at least in part) the criticality of the first AMAS. This task is and will be more and more fastidious because of the increasing complexity of the AMAS designed. That is why [Lemouzy, 2011] argues that the criticality of an agent has to adapt itself when it has to work with agents coming from other AMAS.

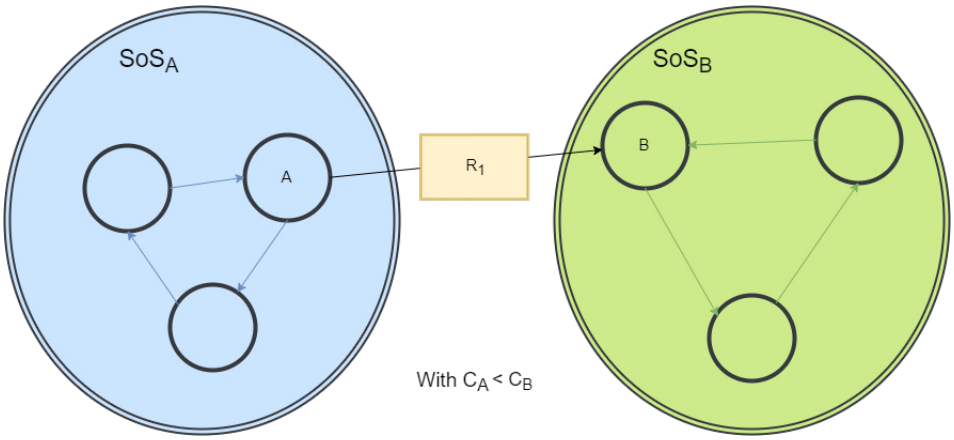


Figure 11.1 – Example of criticality comparison issue between two SoS

11.2 The Problem of the Criticality in SApHESIA

During first tests of the cooperative algorithm on several SoS, issues appear concerning the criticality comparison between different SoS. Indeed, if the SoS are independently designed and put together, the criticalities of some component systems (of a first SoS) may be always upper than others (of a second SoS). It comes from the fact that SoS are defined by different designers and then the criticality in each SoS is not computed in the same way. For example, a criticality of a SoS can takes its values in $[0, 0.5[$ and the other in $[0.5, 1[$. In this case and because of our cooperative algorithm, some component systems may always help others (because of a higher predefined criticality) without fulfilling their own goals. Figure 11.1 shows this issue. the component A of SoS_A is always giving resource R_1 to B of SoS_B because $\forall t, C_A(t) < C_B(t)$. The combination of several SoS can be then counter-productive in such cases, making a SoS (where component systems of a SoS work fine when working between them) that never reaches its adequate functionality. Finally, these heterogeneous intervals lead to a problem of normalization.

11.3 Proposition

Let $\mathcal{S}$ be a set of SoS and $SoS_A, SoS_B \in \mathcal{S}$ two different SoS. A **criticality comparison problem** occurs when the range of criticality of $A \in SoS_A$ is always lower than the one of a component system B of another SoS_B, leading A to always uses functionalities that help component systems of SoS_B without fulfilling its own goals. To avoid this problem, we propose to give to A the capacity to compute new criticalities for component systems of SoS_B that are modified when A detects a **criticality comparison problem**.

For all $B \in SoS_B$, this new criticality of B is called the **adapted criticality of** B **for** A and is noted $C_{B/A}(t)$. This adapted criticality is used instead of the standard criticality of B ($C_B(t)$) in the cooperative algorithm defined in chapter 5.

To know when modifying the **adapted criticality of** B **for** A, the occurrence of a **criticality comparison problem with** SoS_B is defined as the occurrence of these two propositions detected by A **simultaneously**:

1. A cannot fulfill its own goals;

2. A helps during a long period of time the component systems coming from SoS_B .

We propose to define formally, $\forall B \in SoS_B$, the **adapted criticality of** B **for** A $C_{B/A}(t)$ such as $C_{B/A}(t) = \alpha_{SoS_B/A} \times C_B(t)$ with:

- $\alpha_{SoS_B/A} \in]0,1]$ is the **adaptive coefficient** that decreases when A detects phenomena (1) and (2) with a component system of SoS_B;

- $C_B(t)$ is the criticality of B.

The aim of this algorithm is :

- to decrease $\alpha_{SoS_B/A}$ when a **criticality comparison problem with** SoS_B occurs;

- to increase $\alpha_{SoS_B/A}$ when no **criticality comparison problem with** SoS_B occurs;

In this way, if A helps too often component systems of SoS_b without fulfilling its own goals, A decreases $\alpha_{SoS_B/A}$ then decreases $C_{B/A}(t)$ until this one comes to an equivalent level to its own criticality, enabling A to help itself again.

Finally, we propose an algorithm that generalizes this behavior of A for all $SoS \in \mathcal{S}$. Let $SoS \in \mathcal{S}$ a SoS, each component system $s \in SoS$ is able to compute a set of **adaptive coefficients** $Alpha^s = \{\alpha_{SoS_1/s}, \alpha_{SoS_2/s}, ..., \alpha_{SoS_n/s}\}$ where $\alpha_{SoS_i/s}$ is the adaptive coefficient of s computed when a **criticality comparison problem with** SoS_i occurs.

Algorithm description: Each component system A is able to compute a set of coefficients $Alpha^A = \{\alpha_{SoS_1/A}, \alpha_{SoS_2/A}, ..., \alpha_{SoS_n/A}\}$ implemented thanks to a dictionary (called *alphaTable* in the algorithm 11.1). The keys of this dictionary are the name of the SoS and the values are the different α_{SoS_i} coefficients. $\alpha_{SoS_i/A}$ is decreased if A detects the phenomena (1) and (2) with component systems of SoS_i.

To detect that A cannot fulfill its own goals (1), A starts by computing a variable FI that represents its distances to its own goals. Formally, FI is the sum of the integral on $[t - \Delta, t]$ of the difference between its current states (representing by its resources) and its different goals (from line 4 to 6). The more FI is high the more the component system cannot fulfill its own goals. This computation is represented by the blue area on figure 11.2. FI is calculated on a shifted window of size Δ, avoiding to use too old values that are not representative anymore. This delta is a parameter that can be chosen by the designer.

To detect that A has helped during a long period of time the component systems coming from SoS_i (2), the second part of the computation of $\alpha_{SoS_i/A}$ (made from line 7 to 9) saves in a dictionary of past interactions (called *pastInteractions*) the number of times A has used a functionality that helps the component systems of SoS_i. Then, the more *pastInteraction* contains a high value for SoS_i (given by $pastInteractions[S_i.getSoS()]$, S_i being the last helped component system of SoS_i), the more it has helped SoS_i.

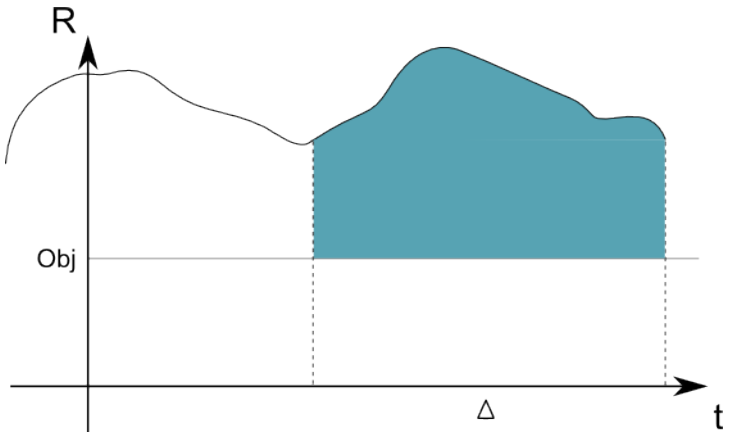


Figure 11.2 – Alpha extraction for a goal $g = \{R == Obj\}$

Finally, the $\alpha_{SoS_i/A}$ coefficient is computed (line 10) as the inverse of FI and the number of past interactions $pastInteractions[S_i.getSoS()]$. Then, when A uses its cooperative algorithm 5.1 in chapter 5, it can uses $\alpha_{SoS_i/A}$ to compute $C_{S_i/A}(t)$ with the criticality given by a component system S_i from SoS_i and then avoiding to help infinitely component systems that have always a higher criticality.

Algorithme 11.1 : Criticality adaptation algorithm

```
1  /*Update the α coefficients for criticality adaptation*/
2  updateAlphas(s As ComponentSystem, alphaTable As Dictionary<String, Float>,
   pastInteractions As Dictionary<String, Float>):
3  for (α ∈ alphaTable) do
4  |  for g ∈ G do
5  |  |  FI += ∫_{t-Δ}^{t} (g.Qu − S.R(g.Re)(t));
6  |  end
7  |  if (s.getSoS() == alphaTable.getKey()) then
8  |  |  pastInteractions[s.getSoS()] + + ;
9  |  end
10 |  alphaTable(s.getSoS()) = FI^{-1} × pastInteractions[s.getSoS()]^{-1} ;
11 end
```

11.4 Experiment Description

We propose the following experiment with two SoS (SoS_A and SoS_B) composed of 3 component systems (figure 11.3). In figure 11.3, yellow boxes represent resources and black arrows represent links between component systems. In this experiment, SoS_A is composed of 3 component systems A_1, A_2, A_3 that are linked together to transform *RawMaterial* resources to *FinalProduct* resources. *RawMaterial* is generated from the environment (thanks to the rule Ru_{Raw}). A_1 through the functionality $F_{ProductMachined}$ transforms it to *MachinedProduct* and produces it as much as possible (thanks to goal $\{MachinedProduct > 0,1\}$). Then A_2 transforms *MachinedProduct* through the functionality $F_{ProductPainted}$ to *PaintedProduct* and produces it as much as possible (thanks to goal $\{PaintedProduct > 0,1\}$). 1 represents the

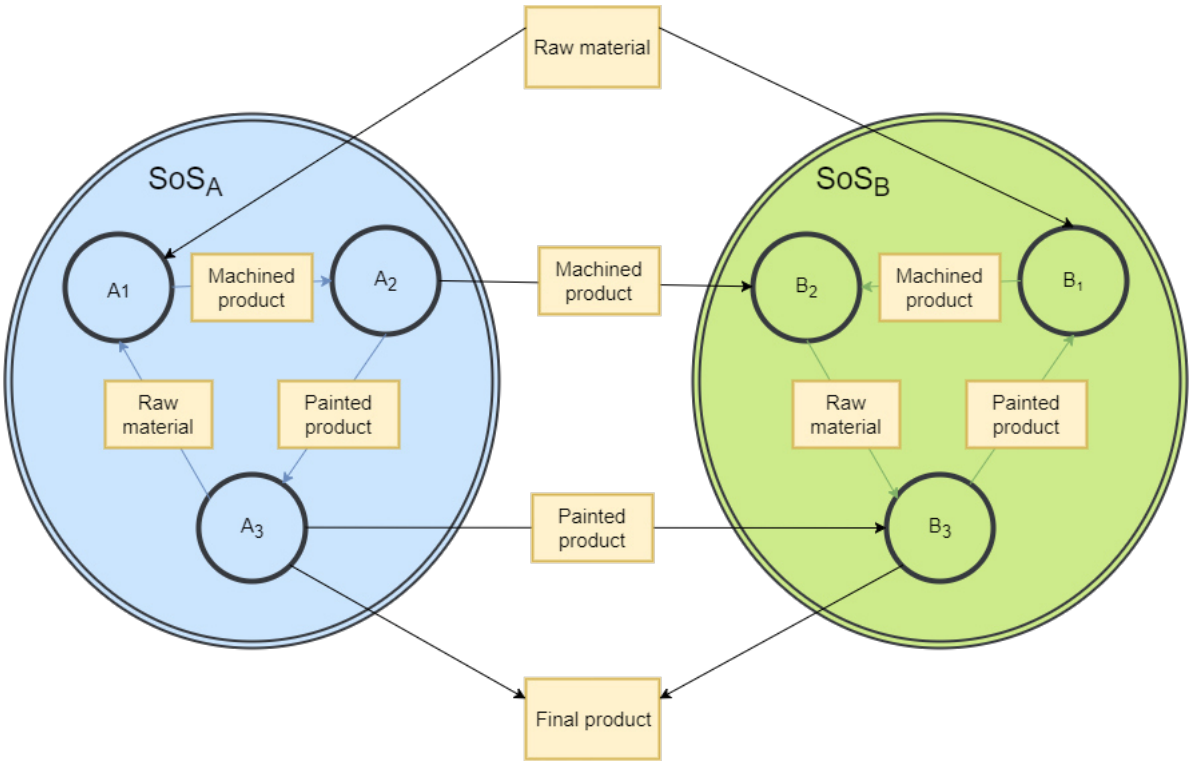


Figure 11.3 – Example with two SoS

priority of the goal. The more the priority is high, the more the goal has an importance in the criticality computation (see section 5.3). Finally A_3 transforms $PaintedProduct$ through the functionality $F_{ProductFinal}$ in $FinalProduct$ resource and tries to produce 5 units of it thanks to the goal $\{FinalProduct == 5, 1\}$. As shown in figure 11.3, the SoS SoS_B has exactly the same functioning with B_1, B_2 and B_3, except that B_3 has to produce 10 $FinalProduct$ (through the goal $\{FinalProduct == 10, 1\}$).

To show the problem of criticality adaptation, two links between SoS_A and SoS_B have been added: a link from A_2 to B_2 and from A_3 to B_3. A_2 is able to give extra $PaintedProduct$ to B_2 (through $F_{SendPainted}$) and A_3 is able to give extra $PaintedProduct$ to B_3. Finally, to cause a criticality comparison problem in this experiment, the component systems of the SoS_A have an additional goal $G_{DistordCriticality} = \{R_{On} == 1, 1\}$ that is always reached for component systems of SoS_A (because $\forall a \in SoS_A, a.r(R_{On}) == 1$). Then, the criticality of component system of SoS_A is always lower than the component systems of SoS_B.

Finally two rules are defined in the environment Ru_{Raw} to produce $RawMaterial$ and $Ru_{FinalProduct}$ to model a consumption of $FinalProduct$ by the environment. The full SApH-ESIA description of the problem is the following:

$$A_1 = \{T_{A_1}, R_{A_1}, Acq_{A_1}, L_{A_1}, F_{A_1}, G_{A_1}, Cost_{A_1}\}$$

With:

- $T_{A_1} = A_1$
- $R_{A_1} = \{\{RawMaterial, 0\}, \{MachinedProduct, 0\}, \{R_{On}, 1\}\}$

- $Acq_{A_1} = \emptyset$
- $L_{A_1} = \{A_2\}$
- $F_{A_1} = \{F_{ProductMachined}, F_{SendMachined}\}$
- $G_{A_1} = \{\{MachinedProduct > 0, 1\}, G_{DistordCriticality}\}\}$
- $Cost_{A_1} = 1$

With:

- $F_{ProductMachined} =$
  $\{\{RawMaterial \geqslant 1\} \rightarrow \{\{RawMaterial \mathrel{+}= -1\}, \{MachinedProduct \mathrel{+}= 1\}\}, t_m, p_m\}$
- $F_{SendMachined} =$
  $\{\{MachinedProduct \geqslant 1\} \rightarrow \{\{MachinedProduct \mathrel{+}= -1\}, \{SyS.R(MachinedProduct) \mathrel{+}= 1\}\}, t_s, p_s\}$

$$A_2 = \{T_{A_2}, R_{A_2}, Acq_{A_2}, L_{A_2}, F_{A_2}, G_{A_2}, Cost_{A_2}\}$$

With:

- $T_{A_2} = A_2$
- $R_{A_2} = \{\{MachinedProduct, 0\}, \{PaintedProduct, 0\}, \{R_{On}, 1\}\}$
- $Acq_{A_2} = \emptyset$
- $L_{A_2} = \{A_2', A_3\}$
- $F_{A_2} = \{F_{SendPainted}, F_{ProductPainted}\}$
- $G_{A_2} = \{\{MachinedProduct > 0, 1\}, \{PaintedProduct > 0, 1\}, G_{DistordCriticality}\}\}$
- $Cost_{A_2} = 1$

With:

- $F_{ProductPainted} =$
  $\{\{MachinedProduct \geqslant 1\} \rightarrow \{\{MachinedProduct \mathrel{+}= -1\}, \{PaintedProduct \mathrel{+}= 1\}\}, t_p, p_p\}$
- $F_{SendPainted} =$
  $\{\{PaintedProduct \geqslant 1\} \rightarrow \{\{PaintedProduct \mathrel{+}= -1\}, \{SyS.R(PaintedProduct) \mathrel{+}= 1\}\}, t_s, p_s\}$

$$A_3 = \{T_{A_3}, R_{A_3}, Acq_{A_3}, L_{A_3}, F_{A_3}, G_{A_3}, Cost_{A_3}\}$$

With:

- $T_{A_3} = A_3$
- $R_{A_3} = \{\{PaintedProduct, 0\}, \{FinalProduct, 0\}, \{R_{On}, 1\}\}$
- $Acq_{A_3} = \emptyset$
- $L_{A_3} = \{A_1, B_3\}$
- $F_{A_3} = \{F_{ProductFinal}, F_{SendFinal}\}$
- $G_{A_3} = \{\{PaintedProduct > 0, 1\}, \{FinalProduct == 5, 1\}, G_{DistordCriticality}\}\}$
- $Cost_{A_3} = 1$

With:

- $F_{ProductFinal} =$
  $\{\{PaintedProduct \geqslant 1\} \rightarrow \{\{PaintedProduct \mathrel{+}= -1\}, \{FinalProduct \mathrel{+}= 1\}\}, t_p, p_p\}$
- $F_{SendFinal} =$
  $\{\{FinalProduct \geqslant 1\} \rightarrow \{\{FinalProduct \mathrel{+}= -1\}, \{SyS.R(FinalProduct) \mathrel{+}= 1\}\}, t_f, p_f\}$

$$B_1 = \{T_{B_1}, R_{B_1}, Acq_{B_1}, L_{B_1}, F_{B_1}, G_{B_1}, Cost_{B_1}\}$$

With:

- $T_{B_1} = B_1$
- $R_{B_1} = \{\{RawMaterial, 0\}, \{MachinedProduct, 0\}, \{R_{On}, 0\}\}$
- $Acq_{B_1} = \emptyset$
- $L_{B_1} = \{B_2\}$
- $F_{B_1} = \{F_{ProductMachined}, F_{SendMachined}\}$
- $G_{B_1} = \{\{RawMaterial > 0, 1\}, G_{DistordCriticality}\}\}$
- $Cost_{B_1} = 1$

$$B_2 = \{T_{B_2}, R_{B_2}, Acq_{B_2}, L_{B_2}, F_{B_2}, G_{B_2}, Cost_{B_2}\}$$

With:

- $T_{B_2} = B_2$
- $R_{B_2} = \{\{MachinedProduct, 0\}, \{PaintedProduct, 0\}, \{R_{On}, 0\}\}$
- $Acq_{B_2} = \emptyset$
- $L_{B_2} = \{B_3\}$
- $F_{B_2} = \{F_{SendPainted}, F_{ProductPainted}\}$
- $G_{B_2} = \{\{MachinedProduct > 0, 1\}, \{PaintedProduct > 0, 1\}, G_{DistordCriticality}\}\}$
- $Cost_{B_2} = 1$

$$B_3 = \{T_{B_3}, R_{B_3}, Acq_{B_3}, L_{B_3}, F_{B_3}, G_{B_3}, Cost_{B_3}\}$$

With:

- $T_{B_3} = B_3$
- $R_{B_3} = \{\{PaintedProduct, 0\}, \{FinalProduct, 0\}, \{R_{On}, 0\}\}$
- $Acq_{B_3} = \emptyset$
- $L_{B_3} = \{B_1\}$
- $F_{B_3} = \{F_{SendFinal}, F_{ProductFinal}\}$
- $G_{B_3} = \{\{PaintedProduct > 0, 1\}, \{FinalProduct == 10, 1\}, G_{DistordCriticality}\}\}$
- $Cost_{B_3} = 1$

$$Ru_{Raw} = \{\emptyset\} \rightarrow \{A_1.R(RawMaterial) += 0.05\}, \{B_1.R(RawMaterial) += 0.05\}$$

$$Ru_{FinalProduct} = \{\emptyset\} \rightarrow \{A_3.R(FinalProduct) -= 0.1\}, \{B_3.R(FinalProduct) -= 0.1\}$$

The values of generation and consumption (0.05 and 0.1) in these two rules have been chosen arbitrarily.

11.5 Results

Figure 11.4 shows the different curves given by 1000 cycles of simulations. The **Criticality** curves shows that the criticality of component systems of SoS_A is always lower than SoS_B. A consequence is that the component A_2 gives all its *PaintedProduct* to B_2 and A_3 gives all its *FinalProduct* to B_3 through the different functionalities F_{Send} (cf. F_{Send} curve). Then, from cycle 0 to 300, A_3 cannot fulfill its goal of producing 5 units of *FinalProduct* (cf. **FinalProduct** curve). But, around cycle 100, the coefficient concerning the SoS_B is decreasing for A_2 and for A_3 (cf. **Coeff from SoS A to SoS B** curve). As a result, the criticality of B_2 and B_3 decreases until cycle 400. Then, A_2 starts giving *PaintedProduct* to A_3 and A_3 stops giving *FinalProduct* to B_3, enabling A_3 to fulfill its goals and then the SoS_A to function with the presence of the SoS_B.

11.6 Discussion

At the beginning of the simulation, A_2 has a lower criticality than B_2. Then, as soon as A_2 has produced one *PaintedResource*, A_2 gives it to B_2 thanks to $F_{SendPainted}$ because its cooperative algorithm informs it that the criticality of B_2 will decrease if it gives *PaintedResource* to B_2 (because B_2 owns the goal $\{PaintedProduct > 0, 1\}$). But, each time $F_{SendPainted}$ is used by A_2, the criticality adaptation algorithm increments the $pastInteractions[B_2.getSoS()]$ (line 8 of the algorithm). Moreover, as A_2 gives all its *PaintedProduct*, its goal $\{PaintedProduct > 0, 1\}$ is not fulfilled. Then, the FI variable (computed thanks to the for loop from the line 4 to 6), containing the distance between the current state and the goals of A_2, is high. As a result of the increment of the $pastInteractions[B_2.getSoS()]$ and a high value of FI, the **adaptive coefficient** α_{SoS_B/A_2} (computed in line 10) becomes to decrease, then the **adapted criticality of** A_2 **for** B_2 $C_{B_2/A_2}(t) = \alpha_{SoS_B/A_2} \times C_B(t)$ becomes to decrease (cf. the curve **Coeff from SoS A to SoS B**.

α_{SoS_B/A_2} decreases over time until stabilizing around cycle 300. The coefficient is stabilizing because the value of αA_{2SoS_B} decreases until $C_{A_2}(t) > C_{B_2/A_2}(t)$ (cf. the curve **Adapted criticality from** A_2). Then, A_2 becomes more critical than B_2 (even if $C_{B_2}(t) > C_{A_2}(t)$), enabling A_2 to help itself by keeping *PaintedProduct* and then giving them to A_3, enabling A_3 to produce *FinalProduct* (cf. the curve **FinalProduct**).

Once A_3 starts to have *FinalProduct* (around cycle 100), the same dynamic appears between A_3 and B_3: A_3 begins to give *FinalProduct* to B_3, decreases its α_{SoS_B/A_3} and finally produces more *FinalProduct* around cycle 400.

We also notice that the two adaptive coefficients α_{SoS_B/A_2} of A_2 and α_{SoS_B/A_3} of A_3 converge to the same value.

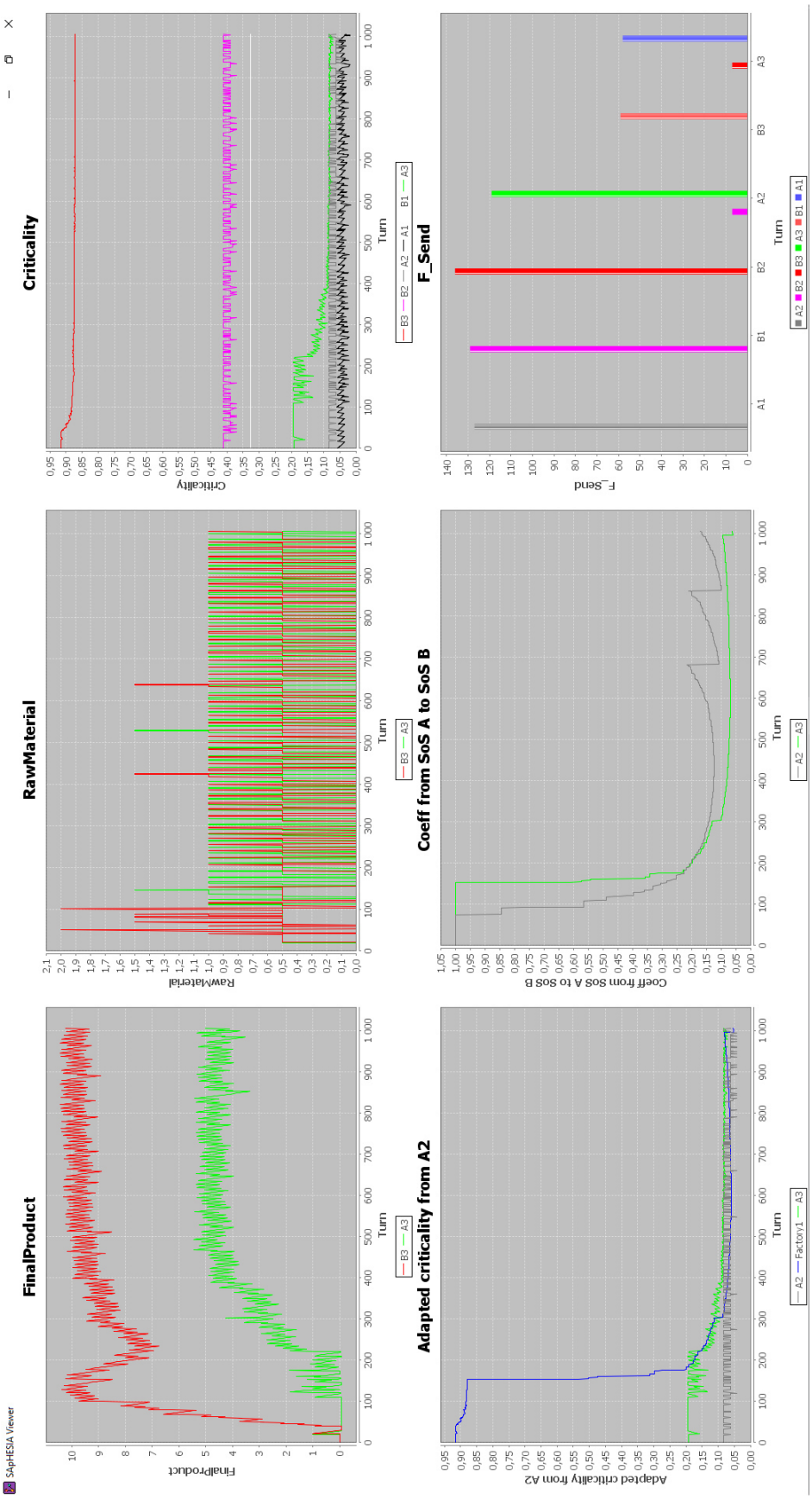


Figure 11.4 – Results of the experiment with the criticality adaptation

11.7 Proof of the Uniqueness of the Normalization

Our experimentation shows that the coefficients α_{SoS_B/A_2} of A_2 and α_{SoS_B/A_3} of A_3 converge to the same value. It also shows that learning these coefficients takes time and requires that A_2 and A_3 exchange with component systems of SoS_B, thus reducing the efficiency of both A_2 and A_3. That is why we propose to prove that these coefficients α_{SoS_B/A_i} are the same for all the component systems of SoS_A communicating with component systems of SoS_B. In other terms, we are going to prove that the coefficient of normalization from SoS_A for SoS_B is unique. This proof is composed of 3 parts:

- the first is dedicated to show that the decision of a component system is invariant with the range of criticalities of the SoS;
- the second is the fact that non-normalization of the different ranges between several SoS prevent the cooperation between them;
- the third is the proof of the uniqueness of the coefficient of criticality normalization between two SoS.

The first part of the proof is to show that the decision of a component system S from a SoS is invariant when multiplying its criticality range by a constant.
Let $[0, {C^{MAX}}_{SoS}]$ be the interval of the criticality of SoS. As the criticality is a metric superior to zero, then ${C^{MAX}}_{SoS} > 0$. As the decision is based on the leximin of the cooperation table computed with algorithm 5.1 (see chapter 5 for more details) the decision of the component system S is based on the **order of the criticalities of its neighborhood**. If the whole table is multiplied by the same constant, the order is not changed, thus the decision of S is not changed. We can then conclude that the decision of a component system is invariant with the criticality range of the SoS.

Secondly, the non-normalization of the different criticality ranges between SoS prevents them to cooperate.
As an example, let A be a component system of SoS_A and let B be a component system of SoS_B. To lighten the notation we note C_A the criticality of A (instead of $C_A(t)$, even if C_A depends of the time). Let $[0, {C^{MAX}}_{SoS_A}], [0, {C^{MAX}}_{SoS_B}]$ be respectively the intervals of criticalities of SoS_A and SoS_B. As, in the general case ${C^{MAX}}_{SoS_A} \neq {C^{MAX}}_{SoS_B}$, let suppose that ${C^{MAX}}_{SoS_A} < {C^{MAX}}_{SoS_B}$. Let $\{v_{A_1}, ..., v_{A_n}\} \in SoS_A$ be the neighborhood of A in SoS_A and $\{v_{B_1}, ..., v_{B_m}\} \in SoS_B$ the neighborhood of A but in SoS_B. As ${C^{MAX}}_{SoS_A} < {C^{MAX}}_{SoS_B}$, it can happen that (as we have seen in the previous experimentation) the order of criticalities are the following $\{C_{v_{A_1}} < ... < C_{v_{A_n}} < C_{v_{B_1}} < ... < C_{v_{B_m}}\}$. In this case, A cannot help its own neighborhood but always helps the component systems of B. Then the SoS_A cannot properly cooperate with SoS_B when the ranges are not normalized.

Now we are going to prove that the coefficient of normalization is unique. To always be able to compare the criticality of A in SoS_A with the one of B of SoS_B, the adapted criticality of B for A (noted $C_{B/A} = \alpha_{SoS_B/A} \times C_B$ with ${C^{MAX}}_{B/A}$ its maximum) has to be normalized such as :
${C^{MAX}}_{SoS_A} = {C^{MAX}}_{B/A}$

$\Leftrightarrow C^{MAX}{}_{SoS_A} = \alpha_{SoS_B/A} \times C^{MAX}{}_{SoS_B}$
$\Leftrightarrow \alpha_{SoS_B/A} = \frac{C^{MAX}{}_{SoS_A}}{C^{MAX}{}_{SoS_B}}$

Then, the only value possible is $\alpha_{SoS_B/A} = \frac{C^{MAX}{}_{SoS_A}}{C^{MAX}{}_{SoS_B}}$.

In this way, all the criticality values of SoS_B are comparable with the ones coming from SoS_A. Furthermore, as the criticalities of SoS_B are only multiplied by a constant value ($\alpha_{SoS_B/A} = \frac{C^{MAX}{}_{SoS_A}}{C^{MAX}{}_{SoS_B}}$) it does not change the order of the criticalities and then does not change the decision of the component system. So, the value of the coefficient for adapting the criticality from a SoS to another is unique. The advantage of this uniqueness is the possibility to share a learned coefficient between the component systems of the same SoS.

11.8 Conclusion

This final experiment chapter presents an extension of our cooperative algorithm that is a new algorithm that enables component systems to adapt their criticality when working with other SoS. Indeed, the criticality of a SoS may be computed in a different way than the criticality of another SoS because they have been independently designed. This experiment shows that in this case, our algorithm of criticality adaptation enables two SoS to work in the same environment together by the self-normalization of the criticality of their component systems. Moreover, we prove that the coefficient of normalization can be shared between the component systems of a same SoS to save time on the learning phase of the coefficient. As previously said, this problematic also appears in the AMAS approach. This proof is applicable to the AMAS approach by replacing a component system by an agent and a SoS by an AMAS.

Conclusion

The System of Systems research area is a widely open one. As a new and trans-domain research area, a lot of efforts have been made to find the generic characteristics and definitions of the notion of SoS. It is common to define a SoS as a subclass of a complex system where the component systems (the parts of the SoS) have specific characteristics such as managerial and operational independence. As a complex system, a SoS evolves in a dynamic environment where unpredictable events may occur and be harmful for the SoS. To cope with this problematic, SoS has to self-architect in a dynamic manner.

Contributions

After summarizing literature on generic SoS characteristics and definitions, we have proposed a working definition that takes into account the most common generic accepted characteristics of SoS as well as the notion of dynamic environment. Finally two generic SoS models are presented with a focus on their limitations concerning the interactions between component systems, the environment model and their global expressiveness. That is why the first contribution of this thesis is a new SoS generic model suiting our working definition of a SoS. It enables to model the Maier's criteria such as the managerial and the operational independence of the components and their geographical distribution. The notions of environment and interactions between component systems have been extended from existing SoS models. The dynamics of the environment is taken into account through active entities and the rules that have effects on interactions of component systems.

We presented two SoS architecting heuristics and showed that they do not respect some of the core principles proposed by [Azani, 2008] such as the open interface and the self-government principles. As the simulation of SoS is also an important field of research, we have explained the main paradigms and tools used to simulate SoS such as DEVS and ABS. We also presented the AMAS approach, used to create multi-agents systems where the overall functionality emerges from the self-organization of its parts driven by their cooperation. The AMAS approach can be an interesting paradigm to implement a new SoS architecting heuristic that respects the open interface and the self-government principles. That is why we have proposed a SoS architecting heuristic implemented through an algorithm that uses cooperation as a social behavior to enable self-organization of the component systems of the SoS. The cooperation is implemented through the concept of criticality that we have

formalized to propose a reusable metric beyond the scope of the SoS. As a bottom-up approach, our heuristic is decentralized (does not need a central management entity) and enables to architect virtual and collaborative SoS. It is also open, (i.e., component systems can be added/removed at runtime) enabling to respect the open interface principle of SoS.

The third contribution concerns architecting tools to simulate generic SoS. We propose SML, an XML language defined to easily declare SoS through XML files. Furthermore, visualization tools have been developed to analyze resources and criticality evolution during experiments.

As we have searched but did not find any case studies with real and usable data to evaluate our work, we have evaluated our propositions through four experimentations found in literature or made by ourselves. The first experimentation concerns the Missouri Toy problem that is a communication transportation problem. It has been chosen because it has been already studied and it is a good starting point for testing our heuristic concerning its robustness and its capacity to be functionally adequate. The results show that our heuristic is robust, functionally adequate and open.

The second experimentation concerns CoCaRo, a resources transportation system we have defined to validate two contributions. The first relates to the metric of criticality enabling cooperation between agents. The second one uses SApHESIA to define an agent (which is a component system) representing an overall AMAS in order to enable to several AMAS to work together and improve their performances. The results show that the criticality enables cooperation between agents and improves the global performance of the system and its robustness. Moreover, the use of SApHESIA also improves the performance of the system.

The third experimentation concerns the instantiation of a UAV fleet within a hostile environment. It enables the comparison of our heuristic with satisficing games regarding to the overall performance of the system. The results of our experiments show that our approach is competitive with satisficing games in term of performance. Furthermore, experiments with numerous component systems validate our complexity analysis by showing that the computational cost of our approach is linear with the number of component systems.

The fourth experimentation relates to the capacity of our approach to reuse and combine existing SoS that are independently designed but interdependent. The results show that the notion of criticality between two SoS enables in a natural way their combination.

The final experiment proposes an improvement of our heuristic: component systems from different SoS may have to adapt their criticalities if these latter do not have the same scale if they detect issues concerning the comparison of the criticality. The results show that our algorithm of criticality adaptation enables the coupling of two SoS even if their respective criticalities have different scales.

Finally, all of the results show that our approach is a generic SoS architecting heuristic that respects the set of principles for efficient SoS Architecting defined by [Azani, 2008] such as the self-government and open interface principles. Its total decentralization enables to find alternative architectures for virtual and collaborative SoS.

Perspectives

Even if our work showed interesting results, it is only a tiny part, a modest contribution with regards to what it remains to do in the vast SoS research area.

Concerning short-terms perspectives, testing our approach with more component systems will enable to validate that our heuristic can scale-up. Because of the number, the complexity and the heterogeneity of the systems involved, domains such as smart cities and factories of the future could be future case studies. Furthermore, implementing other heuristics than the cooperation in SApHESIA would enable to compare our results for the same experiments.

Concerning mid-terms perspectives, we think that this work on the SoS field could be extended to the AMAS approach. It would be interesting to reify every component system of a SoS into an AMAS agent in order to transform a SoS into an AMAS. In this way, the component system model could be a basis to a formal model of an AMAS agent. Indeed, the AMAS approach has been instantiated in several domains and is mature to be generalized and formalized. It could enable to propose a generic algorithm of cooperation such as the one we propose in this thesis.

Concerning long-term perspectives, to collaborate with SoS experts to define more complex examples with real data and real case studies is possible. Thanks to the simplicity and the ability of our heuristic to find architecture solution with loops and non-linearity, our heuristic could be the first step of a more global process of SoS design. The use of our SApHESIA tools could enable SoS designers to do high level simulations of SoS and then proposing first solutions of architecture.

Abbreviations

ABS Agent-Based Simulation

AMAS Adaptive Multi-Agent System

DoD Department of Defense

FIE Fuzzy Inference Engine

FIS Fuzzy Inference System

INCOSE International Council on Systems Engineering

KPA Key Performance Attribute

NCS Non-Cooperative Situation

SAHS SoS Architecting Heuristic Simulation

SApHESIA SoS Architecting HEuriStIc based on Agents

SAR Search And Rescue

SE Systems Engineering

SML SApHESIA Modeling Language

SoS System of Systems

SoSE Systems of Systems Engineering

UAV Unmanned Aerial Vehicle

List of Figures

List of Tables

Part IV

Annexes

Annex 1

This annex proposes examples of SML. The SML is a XML language used to declare SApHESIA models in XML files that are used as inputs of our simulation tools. The syntax of the SML is formally described in the chapter 6.

System

As an example, let's take Sat_{A3}, a component system of type Sat_A of the Missouri toy model. In this example, the generic SApHESIA model for Sat_{A3} is:

$$Sat_{A3} = \{T_{Sat_{A3}} R_{Sat_{A3}}, Acq_{Sat_{A3}}, L_{Sat_{A3}}, F_{Sat_{A3}}, G_{Sat_{A3}}, Cost_{Sat_{A3}}\}$$

With:

- $T_{Sat_{A3}} = SAT_A$
- $R_{Sat_{A3}} = \{\{Signal, 20\}\}$
- $Acq_{Sat_{A3}} = \{UAV_1, UAV_3, Carrier\}$
- $L_{Sat_{A3}} = \{UAV_3, 1\}$
- $F_{Sat_{A3}} = \{F_{LinkSatA}, F_{Send}\}$
- $G_{Sat_{A3}} = \{G_{Link}, G_{Send}\}$
- $Cost_{Sat_{A3}} = 100$

Then, its translation in SML is the following:

```
<System>
        <Name> SatA3 </Name>
        <Type> SatA </Type>

        <Resource>
                <Type> Signal </Type>
                <Quantity> 20 </Quantity>
```

```
        </Resource>

        <Acquaintance>
                <To> UAV3 </To>
                <To> UAV1 </To>
                <To> Carrier </To>
        </Acquaintance>

        <Link>
                <Intensity> 1 </Intensity>
                <To> UAV3 </To>
        </Link>

        <Functionality  > F_LinkSatA </Functionality>
        <Functionality  > F_Send </Functionality>

        <Goal>
                <Object> Signal </Object>
                <Type> EQ </Type>
                <Value> 0 </Value>
                <Priority> 1 </Priority>
        </Goal>

        <Goal>
                <Object> Link </Object>
                <Type> EQ </Type>
                <Value> 2 </Value>
                <Priority> 1 </Priority>
        </Goal>

        <Cost> 100 </Cost>
</System>
```

Functionality

As a example, we propose the functionality F_{Send} used in the Missouri Toy problem:

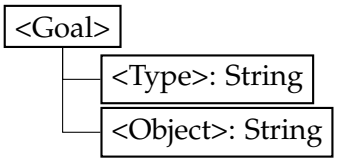
$F_{Send} = \{\{Signal \geqslant 1\} \rightarrow$
$\{Signal \mathrel{+}= -1\}, \{R_{CostSend} \mathrel{+}= 1\}, \{Signal \mathrel{+}= -1\}, \{SyS, Signal \mathrel{+}= 1\}, t_{Send}, p_{Send}\}$

The functionality F_{Send} consumes one resource *Signal* and create one resource *Signal* to a component system of type UAV, $SatA$, $SatB$, $Carrier$, it has a probability of 0.7 to success and takes one unit of time to proceed.

```
<Functionality  >
        <Name> F_Send </Name>
```

```
        <Condition>
                <Type> Resource </Type>
                <Name> Signal </Name>
                <Quantity> 1 </Quantity>
                <ResourceOp> == </ResourceOp>
        </Condition>
        <Effect>
                <Type> Resource </Type>
                <Name> Signal </Name>
                <Quantity> -1 </Quantity>
        </Effect>
        <Effect>
                <Type> Resource </Type>
                <Name> Signal </Name>
                <Quantity> 1 </Quantity>
                <To> UAV, SatA, SatB, Carrier </To>
        </Effect>
        <Time> 1 </Time>
        <Performance> 0.7 </Performance>
</Functionality>
```

Goals



For example, if a goal of is to get zero resource of type *Signal*, it has to be declared as:

```
<Goal>
        <Object> Signal </Object>
        <Type> EQ </Type>
        <Value> 0 </Value>
        <Priority> 1 </Priority>
</Goal>
```

Rule

As an example, hereafter a rule generating the resource *Signal* to the component system *Ground* when this one has its resource R_{On} superior to 0:

```
<Rule>
```

```
        <Condition>
                'Ground.R_On' SUP 0
        </Condition>

        <Effect>
                'Ground.Signal' EQ 'Ground.Signal' + 0.09
        </Effect>
</Rule>
```

SML file of Missouri Toy Model

The SML file for the simulation of the Toy Problem presented in chapter 7. This file contains:

- two functionalities $F_{LinkSatA}$ and F_{UnLink};
- four component systems are also presented: *Carrier*, *Ground*, UAV_1 and Sat_{A1}
- two rules $Rule_{Turn}$ and $Rule_{GenerateSignal}$.

```
<SoS>
 <Functionality  >
  <Name> F_Send </Name>
  <Performance> PerfSend </Performance>
    <Time> 1</Time>
    <Name> Signal </Name>
        <Quantity> 1 </Quantity>
      <Type> Resource </Type>
    </Condition>
    <Effect>
      <Name> Signal </Name>
        <Quantity> -1 </Quantity>
      <Type> Resource </Type>
    </Effect>
    <Effect>
      <Name> Signal </Name>
      <Quantity> 1 </Quantity>
      <Type> Resource </Type>
      <To> SyS </To>
    </Effect>
  </Functionality>

  <Functionality  >
    <Name> F_UnLink </Name>
    <Performance> 1 </Performance>
    <Time> 1 </Time>
```

```
    <Condition>
      <Name> SyS </Name>
      <Quantity> 1 </Quantity>
      <Type> Link </Type>
    </Condition>
    <Effect>
      <Name> Link </Name>
      <Quantity> -1 </Quantity>
      <Type> Link </Type>
    </Effect>
    <Effect>
      <Name> R_Link </Name>
      <Quantity> -1 </Quantity>
      <Type> Resource </Type>
    </Effect>
    <Effect>
      <Name> R_LinkResource </Name>
      <Quantity> 1 </Quantity>
      <Type> Resource </Type>
      <To> SyS </To>
    </Effect>
  </Functionality>

  <Functionality  >
    <Name> F_LinkSatA</Name>
    <Performance> 1 </Performance>
    <Time> 1 </Time>
    <Condition>
      <Name> SatA </Name>
      <Quantity> 1 </Quantity>
      <Type> System </Type>
    </Condition>
    <Effect>
      <Name> Link </Name>
        <Quantity> 1 </Quantity>
      <Type> Link </Type>
    </Effect>
    <Effect>
      <Name> R_Link </Name>
        <Quantity> 1 </Quantity>
      <Type> Resource </Type>
    </Effect>
    <Effect>
      <Name> R_LinkResource </Name>
```

```
        <Quantity> -1 </Quantity>
      <Type> Resource </Type>
      <To> SatA </To>
    </Effect>
  </Functionality>

  <System>
    <Name> Carrier </Name>
    <Type> Carrier </Type>
    <Cost> 1 </Cost>
    <Resource>
      <Type> Signal </Type>
      <Quantity> 0 </Quantity>
    </Resource>
    <Resource>
      <Type> R_Cost </Type>
      <Quantity> 0 </Quantity>
    </Resource>
    <Resource>
      <Type> R_LinkResource </Type>
      <Quantity> 5 </Quantity>
    </Resource>
    <Goal>
      <Object> Signal </Object>
      <Type> SUP </Type>
      <Value> 0 </Value>
      <Priority> 1 </Priority>
    </Goal>
  </System>

  <System>
    <Name> Ground </Name>
    <Type> Ground </Type>
    <Id>1  </Id>
    <Id>1  </Id>
    <Functionality  > F_Send </Functionality>
    <Functionality  > F_LinkUAV </Functionality>
    <Functionality  > F_LinkSatA </Functionality>
    <Functionality  > F_LinkSatB </Functionality>
    <Functionality  > F_UnLink </Functionality>
    <Resource>
      <Type> Signal </Type>
      <Quantity> 0 </Quantity>
    </Resource>
```

```
    <Resource>
      <Type> R_CostSend </Type>
      <Quantity> 1 </Quantity>
    </Resource>
    <Resource>
      <Type> PerfSend </Type>
      <Quantity> 0.1 </Quantity>
    </Resource>
    <Resource>
      <Type> PerfSendUAV1 </Type>
      <Quantity> 0.7 </Quantity>
    </Resource>
    <Resource>
      <Type> PerfSendUAV3 </Type>
      <Quantity> 0.7 </Quantity>
    </Resource>
    <Resource>
      <Type> Turn </Type>
      <Quantity> 0 </Quantity>
    </Resource>
    <Resource>
      <Type> R_GenerationRate </Type>
      <Quantity> 0.3 </Quantity>
    </Resource>
    <Resource>
      <Type> R_FinalTurn </Type>
      <Quantity> 2000 </Quantity>
    </Resource>
    <Goal>
      <Object> Signal </Object>
      <Type> EQ </Type>
      <Value> 0 </Value>
      <Priority> 1 </Priority>
    </Goal>
    <Goal>
      <Object> Link </Object>
      <Type> SUP </Type>
      <Value> 0 </Value>
      <Priority> 1 </Priority>
    </Goal>
  </System>

  <System>
    <Name> UAV1 </Name>
```

```
    <Type> UAV </Type>
    <Cost> 1 </Cost>
    <Functionality  > F_Send </Functionality>
    <Functionality  > F_LinkUAV </Functionality>
    <Functionality  > F_LinkSatA </Functionality>
    <Functionality  > F_LinkSatB </Functionality>
    <Functionality  > F_LinkCarrier </Functionality>
    <Functionality  > F_UnLink </Functionality>
    <Resource>
      <Type> Signal </Type>
      <Quantity> 0 </Quantity>
    </Resource>
    <Resource>
      <Type> R_CostSend </Type>
      <Quantity> 1 </Quantity>
    </Resource>
    <Resource>
      <Type> R_LinkResource </Type>
      <Quantity> 5 </Quantity>
    </Resource>
    <Resource>
      <Type> PerfSend </Type>
      <Quantity> 0.1 </Quantity>
    </Resource>
    <Resource>
      <Type> PerfSendA1 </Type>
      <Quantity> 0.5 </Quantity>
    </Resource>
    <Goal>
      <Object> Signal </Object>
      <Type> EQ </Type>
      <Value> 0 </Value>
      <Priority> 1 </Priority>
    </Goal>
    <Goal>
      <Object> R_Link </Object>
      <Type> SUP </Type>
      <Value> 0 </Value>
      <Priority> 1 </Priority>
    </Goal>
  </System>

  <System>
    <Name> SatA1 </Name>
```

```
    <Type> SatA </Type>
    <Cost> 1 </Cost>
    <Functionality  > F_Send </Functionality>
    <Functionality  > F_LinkUAV </Functionality>
    <Functionality  > F_LinkSatA </Functionality>
    <Functionality  > F_LinkCarrier </Functionality>
    <Functionality  > F_UnLink </Functionality>
    <Resource>
      <Type> Signal </Type>
      <Quantity> 0 </Quantity>
    </Resource>
    <Resource>
      <Type> R_CostSend </Type>
      <Quantity> 1 </Quantity>
    </Resource>
    <Resource>
      <Type> R_LinkResource </Type>
      <Quantity> 5 </Quantity>
    </Resource>
    <Resource>
      <Type> PerfSend </Type>
      <Quantity> 0.1 </Quantity>
    </Resource>
    <Resource>
      <Type> PerfSendCarrier </Type>
      <Quantity> 1 </Quantity>
    </Resource>
    <Goal>
      <Object> Signal </Object>
      <Type> EQ </Type>
      <Value> 0 </Value>
      <Priority> 1 </Priority>
    </Goal>
    <Goal>
      <Object> R_Link </Object>
      <Type> SUP </Type>
      <Value> 0 </Value>
      <Priority> 1 </Priority>
    </Goal>
  </System>



  <Environment>
```

```
  <Rule>
    <Condition>
      'Ground.Turn' INF 'Ground.R_FinalTurn'
     </Condition>

    <Effect>
      'Ground.Signal' = 'Ground.Signal' + 'Ground.R_GenerationRate'
    </Effect>
  </Rule>

  <Rule>
    <Condition>
      True
    </Condition>
    <Effect>
      'Ground.Turn' = 'Ground.Turn' + 1
    </Effect>
  </Rule>



  </Environment>

</SoS>
```

Annex B

This annex presents the curves concerning the different functions of criticalities for a goal g:

$$g = \{Re \circ\!\!\mathrm{p}\ Qu, Pr\}$$

where:

- Re is a type of resource;
- $Qu \in \mathbb{R}$ is the quantity of resource of type Re the component system wants to own;
- $\circ\!\!\mathrm{p} \in \{=, \neq, <, >, \leqslant, \geqslant\}$ is the comparison operator in order to compare Qu to the current quantity of Re;
- $Pr \in \mathbb{N}^+$ is the priority of the goal enabling to model the relative importance of a goal. The higher the priority is, the more important the goal is.

As a reminder, the following functions are used for the calculation of the criticality of the goal g at time t $C_g(t)$:

$$C_g(t) = \begin{cases} (1) : 1 - \dfrac{1}{e^{\alpha \times \Delta_g(t)}} \\ (2) : \dfrac{1}{e^{\alpha \times \Delta_g(t)}} \\ (3) : 1 - \dfrac{atan((\alpha \times \Delta_g(t)) + \dfrac{\Pi}{2}}{\Pi} \\ (4) : \dfrac{atan((\alpha \times \Delta_g(t)) + \dfrac{\Pi}{2}}{\Pi} \end{cases}$$

(1): if $g.Op \in \{=\}$: the goal g is to reach a given quantity Qu of resource Re;
(2): if $g.Op \in \{\neq\}$: the goal g is to be as far as possible from a given quantity Qu for the resource Re;
(3): if $g.Op \in \{<, =<\}$: the goal g is to be inferior or equal to a given quantity Qu for the resource Re;
(4): if $g.Op \in \{>, =>\}$ the goal g is to be superior or equal to a given quantity Qu for the resource Re .
With :

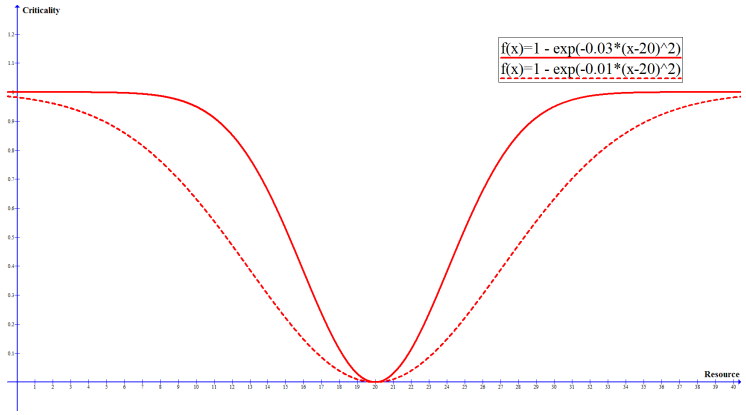


Figure 11.5 – Example of a goal criticality: reach a value ($g.Op \in \{=\}$)

- $\Delta_g(t) = g.Qu - S.R(g.Re)(t)$ is the difference between the given quantity Qu and the current amount of resource Re of S;
- $S.R(g.Re)(t)$ is the current amount of resource Re of component system S at time t;
- α is a coefficient value that influences the shape of the curve.

Figures 11.5,11.6, 11.7 and 11.8 present four examples of goal criticality functions. Numerical values are the following:

- $g.Qu = 20$;
- $\alpha = 0.01$ for the red dotted curve (figure 11.5), $\alpha = 0.03$ for the others.

The shape of these functions can be changed according to the variation of the parameter α. It enables to control how the goal value is important to reach or to avoid. For example, the fact to choose a higher α parameter for the red function will lead to a component system which tends to stay really critical even if it is close to its goal. The variation of this parameter in the goal criticality function is not studied in this document. Nevertheless, an example for two values of α is given on figure 11.5. The red functions (figure 11.5) represent decreasing criticality when reaching objective ($g.Op \in \{=\}$). The blue function (figure 11.6) represents increasing criticality when reaching an unwanted value $g.Op \in \{\neq\}$. The light green one (figure 11.7) represents decreasing criticality when a resource being lower than a given threshold $g.Op \in \{>, =>\}$. The dark green one (figure 11.8) represents decreasing criticality when a resource being greater than a given threshold $g.Op \in \{>, =>\}$.

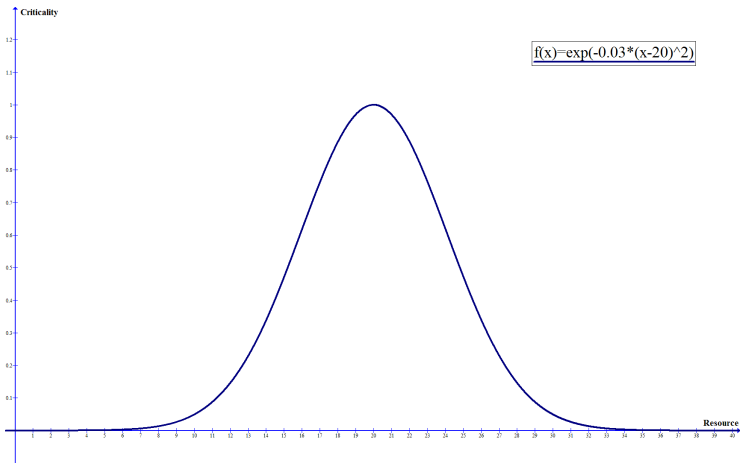


Figure 11.6 – Example of a goal criticality: avoid a value ($g.Op \in \{\neq\}$)

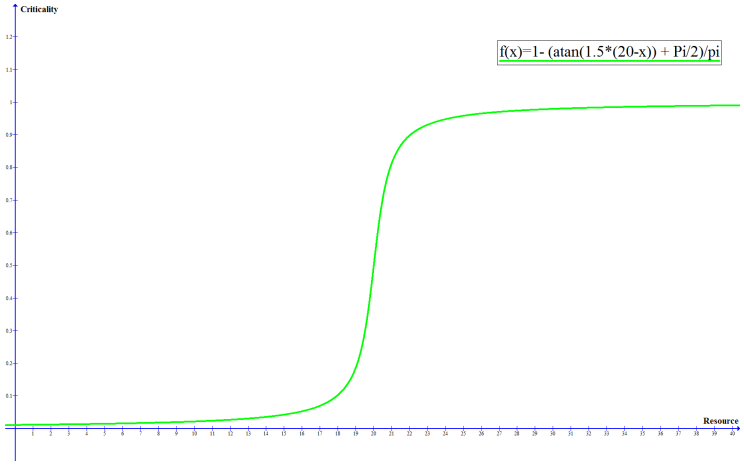


Figure 11.7 – Example of a goal criticality: to be inferior or equal to a value ($g.Op \in \{<, =<\}$)

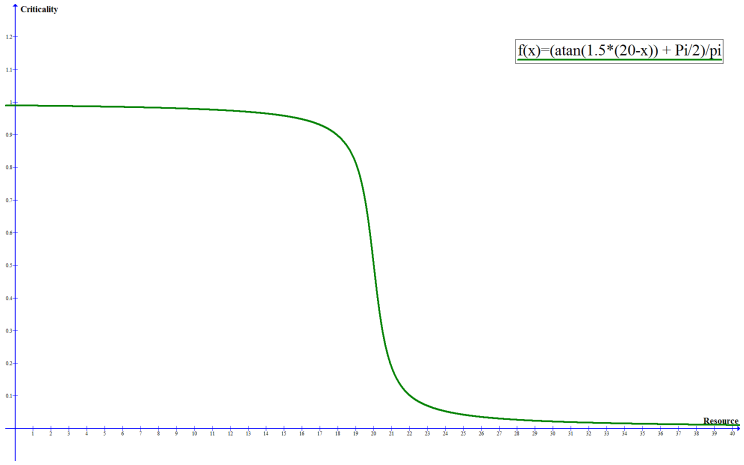


Figure 11.8 – Example of a goal criticality: Be superior or equal to a value ($g.Op \in \{>, =>\}$)

Bibliography

[Acheson et al., 2012] Acheson, P., Pape, L., Dagli, C., Kilicay-Ergin, N., Columbi, J., and Haris, K. (2012). Understanding system of systems development using an agent- based wave model. *Procedia Computer Science*, 12:21 – 30.

[Archibald et al., 2006a] Archibald, J. K., Hill, J. C., Johnson, F. R., and Stirling, W. C. (2006a). Satisficing negotiations. *Trans. Sys. Man Cyber Part C*, 36(1):4–18.

[Archibald et al., 2006b] Archibald, J. K., Hill, J. C., Johnson, F. R., and Stirling, W. C. (2006b). Satisficing negotiations. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 36(1):4–18.

[Ashby, 1956] Ashby, W. R. (1956). *An introduction to cybernetics.* London, UK : Chpaman & Hall.

[Axelsson, 2015] Axelsson, J. (2015). A systematic mapping of the research literature on system-of-systems engineering. In *10th System of Systems Engineering Conference (SoSE)*, pages 18–23.

[Azani, 2008] Azani, C. (2008). *An Open Systems Approach to System of Systems Engineering*, pages 21–43. John Wiley and Sons, Inc.

[Baldwin and Sauser, 2009] Baldwin, W. and Sauser, B. (2009). Modeling the characteristics of system of systems. *IEEE International Conference on System of Systems Engineering (SoSE)*.

[Baldwin et al., 2015] Baldwin, W. C., Sauser, B. J., and Boardman, J. (2015). Revisiting the meaning of of as a theory for collaborative system of systems. *IEEE Systems Journal*, PP(99):1–12.

[Banks et al., 2005] Banks, J., Carson, J., Nelson, B., and D.M., N. (2005). *Discrete-event System Simulation (Fourth Edition)*. Prentice-Hall international series in industrial and systems engineering. Prentice Hall.

[Behdani, 2012] Behdani, B. (2012). Evaluation of paradigms for modeling supply chains as complex socio-technical systems. In *Proceedings of the Winter Simulation Conference*, WSC '12, pages 413:1–413:15. Winter Simulation Conference.

[Benites Gonçalves, 2016] Benites Gonçalves, M. (2016). *Supporting architectural design of acknowledged Software-intensive Systems- of-Systems*. Phd thesis, Université de Bretagne Sud.

[Bessadi et al., 2016] Bessadi, A. K., Jamont, J.-P., Occello, M., Ben Yelles, C. B., and Koudil, M. (2016). About cooperation of multiagent collective products: An approach in the context of cyber-physical systems. In *2016 IEEE RIVF International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, pages 19–24, Hanoi, Vietnam. IEEE.

[Bjelkemyr et al., 2007] Bjelkemyr, M., Semere, D., and Lindberg, B. (2007). An engineering systems perspective on system of systems methodology. In *Systems Conference, 2007 1st Annual IEEE*, pages 1–7.

[BKCASE Editorial Board, 2014] BKCASE Editorial Board (2014). Guide to the Systems Engineering Body of Knowledge. *Guide to the Systems Engineering Body of Knowledge (SEBoK)*, page 945.

[Boardman and Sauser, 2006] Boardman, J. and Sauser, B. (2006). System of systems - the meaning of of. In *2006 IEEE/SMC International Conference on System of Systems Engineering*, pages 118–123.

[Boes, 2014] Boes, J. (2014). *Apprentissage du contrôle de systèmes complexes par l'auto-organisation coopérative d'un système multi-agent*. Phd thesis, Université de Toulouse, Toulouse, France.

[Bonabeau, 2002] Bonabeau, E. (2002). Agent-based modeling: methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(suppl. 3):7280–7287.

[Boulding, 1956] Boulding, K. E. (1956). General systems theory—the skeleton of science. *Management Science*, 2(3):197–208.

[Bouziat et al., 2016] Bouziat, T., Camps, V., and Combettes, S. (2016). Cooperation in Adaptive Multi-Agent Systems through System of Systems modeling. In *Global Conference on Artificial Intelligence (GCAI)*, volume 41, pages 214–226.

[Bouziat et al., 2017] Bouziat, T., Combettes, S., Camps, V., and Boes, J. (2017). Cooperative Multi-Agent Approach for Computational Systems of Systems Architecting. In *International Conference on Agents and Artificial Intelligence (ICAART)*. SciTePress.

[Bouziat et al., 2014] Bouziat, T., Combettes, S., Camps, V., and Glize, P. (2014). La criticité comme moteur de la coopération dans les systèmes multi-agents adaptatifs. In *Journées Francophones sur les Systèmes Multi-Agents (JFSMA)*, pages 149–158. Cepadues Editions.

[Caffall and Michael, 2009] Caffall, D. and Michael, J. (2009). System of Systems Collaborative Formation. *Systems Journal*, 3(3):385–401.

[Camps, 1998] Camps, V. (1998). *Vers une théorie de l'auto-organisation dans les systèmes multi-agents basée sur la coopération : application à la recherche d'information dans un système d'information répartie*. Phd thesis, Université Paul Sabatier, Toulouse, France.

[Capera, 2005] Capera, D. (2005). *Systèmes multi-agents adaptatifs pour la résolution de problèmes: Application à la conception de mécanismes*. Phd thesis, Université Paul Sabatier, Toulouse, France.

[Capera et al., 2003] Capera, D., George, J., Gleizes, M., and Glize, P. (2003). The AMAS theory for complex problem solving based on self-organizing cooperative agents. *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE*, 2003-January:383–388.

[CARLOCK and FENTON, 2001] 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.

[Ceccarelli et al., 2015] Ceccarelli, A., Mori, M., Lollini, P., and Bondavalli, A. (2015). Introducing meta-requirements for describing system of systems. In *IEEE 16th International Symposium on High Assurance Systems Engineering*, pages 150–157.

[Crossley and Professor, 2004] Crossley, W. A. and Professor, A. (2004). System of systems: An introduction of purdue university schools of engineering's signature area," paper presented at the engineering systems symposium.

[Dagli and Acheson, 2012] Dagli, C. H. and Acheson, P. (2012). Fuzzy assessor using type 1 and type 2 fuzzy sets. *Procedia Computer Science*, 8:159 – 164.

[Dahmann et al., 2009] Dahmann, D. J., Baldwin, K. J., and Rebovich, G. (2009). Systems of systems and net-centric enterprise systems. *7th Annual Conference on SyStems Engineering Research*.

[De Wolf and Holvoet, 2005] De Wolf, T. and Holvoet, T. (2005). *Emergence Versus Self-Organisation: Different Concepts but Promising When Combined*, pages 1–15. Springer, Berlin, Heidelberg.

[DeLaurentis et al., 2012] DeLaurentis, D. A., Marais, K., Davendralingam, N., Yeob Han, S., Uday, P., Fang, Z., and Guariniello, C. (2012). Assessing the Impact of Development Disruptions and Dependencies in Analysis of Alternatives of System-of-Systems. TechReport SERC-2012-TR-035, Purdue University, West Lafayette,IN.

[Di Marzo Serugendo et al., 2011] Di Marzo Serugendo, G., Gleizes, M.-P., and Karageorgos, A. (2011). *Self-organising Software: From Natural to Artificial Adaptation*. Springer Publishing Company, Incorporated, 1st edition.

[DoD, 2006] DoD (2006). System of systems systems engineering guide: Considerations for systems engineering in a system of systems environment. Techreport, Department of Defense.

[Drogoul et al., 2013] Drogoul, A., Amouroux, E., Caillou, P., Gaudou, B., Grignard, A., Marilleau, N., Taillandier, P., Vavasseur, M., Vo, D. A., and Zucker, J.-D. (2013). Gama: multi-level and complex environment for agent-based models and simulations (demonstration). In *International Joint Conference on Autonomous Agents and Multiagent Systems - AAMAS 2013*, pages 1361–1362. IFAAMAS.

[Edward Pape II, 2016] Edward Pape II, L. (2016). *A domain independent method to assess system of system meta-architectures using domain specific fuzzy information*. PhD thesis, Missouri University of Science and Technology.

[Ferber, 1995] Ferber, J. (1995). *Les Systèmes multi-agents: vers une intelligence collective*. I.I.A. Informatique intelligence artificielle. InterEditions.

[Georgé, 2004] Georgé, J.-P. (2004). L'émergence. Rapport de recherche 2003-12-R, IRIT, Université Paul Sabatier, Toulouse.

[Gilbert, 2008] Gilbert, N. (2008). *Agent-Based Models*. Sage Publications, London.

[Glize, 2001] Glize, P. (2001). *L'Adaptation des Systemes a Fonctionnalite Emergente par Auto-Organisation Cooperative*. Habilitation à diriger des recherches, Université Paul Sabatier, Toulouse, France.

[Gonçalves et al., 2014] Gonçalves, M. B., Cavalcante, E., Batista, T., Oquendo, F., and Nakagawa, E. Y. (2014). Towards a conceptual model for software-intensive system-of-systems. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1605–1610.

[Gorod et al., 2008] Gorod, A., Sauser, B., and Boardman, J. (2008). System-of-systems engineering management: A review of modern history and a path forward. *IEEE Systems Journal*, 2(4):484–499.

[Guivarch, 2014] Guivarch, V. (2014). *Prise en compte de la dynamique du contexte pour les systèmes ambiants par systèmes multi-agents adaptatifs*. Phd thesis, Université de Toulouse, Toulouse, France.

[Henshaw et al., 2013] Henshaw, M., Siemieniuch, C., Sinclair, M., Barot, V., Henson, S., Ncube, C., Lim, S., Dogan, H., Jamshidi, M., and Delaurentis, D. (2013). The Systems of Systems Engineering Strategic Research Agenda Systems of Systems Engineering. (2).

[Hill et al., 2005] Hill, J. C., Johnson, F. R., Archibald, J. K., Frost, R. L., and Stirling, W. C. (2005). A cooperative multi-agent approach to free flight. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 1083–1090. ACM.

[Hu et al., 2005] Hu, X., Zeigler, B. P., and Mittal, S. (2005). Variable structure in devs component-based modeling and simulation. *Simulation*, 81(2):91–102.

[IEEE, 2010] IEEE (2010). Systems and software engineering – vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, pages 1–418.

[Jackson, 2003] Jackson, M. O. (2003). *Mechanism Theory*. Oxford UK.

[Jamshidi, 2008a] Jamshidi, M. (2008a). System of systems engineering - new challenges for the 21st century. *IEEE Aerospace and Electronic Systems Magazine*, 23(5):4–19.

[Jamshidi, 2008b] Jamshidi, M. (2008b). *Systems of Systems Engineering: Principles and Applications*. CRC Press.

[Johnson, 2001] Johnson, S. (2001). *Emergence: the connected lives of ants, brains, cities, and software*. Scribner.

[Kaddoum, 2011] Kaddoum, E. (2011). *Optimisation sous contraintes de problèmes distribués par auto-organisation coopérative*. Phd thesis, Université de Toulouse, Toulouse, France.

[Lemouzy, 2011] Lemouzy, S. (2011). *Systèmes interactifs auto-adaptatifs par systèmes multi-agents auto-organisateurs : application à la personnalisation de l'accès à l'information*. Phd thesis, Université de Toulouse, Toulouse, France.

[Lukasik, 1998] Lukasik, S. J. (1998). Systems, systems of systems, and the education of engineers. *Artificial intelligence for engineering design analysis and manufacturing*, 12(1):55–60.

[Maier, 1998] Maier, M. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284.

[Manthorpe, 1996] Manthorpe, W. H. J. (1996). The emerging joint system of systems: A systems engineering challenge and opportunity for apl. *John Hopkins APL Tech Dig*, pages 305–310.

[Mittal et al., 2014] Mittal, S., Doyle, M. J., and Portrey, A. M. (2014). *Human in the Loop in System of Systems (SoS) Modeling and Simulation*, pages 415–451. John Wiley & Sons, Inc.

[Mittal and Luis Risco Matin, 2013] Mittal, S. and Luis Risco Matin, J. (2013). *Netcentric System of Systems Engineering with DEVS Unified Process)*. CRC press.

[Morin, 1977] Morin, E. (1977). *La méthode: La Nature de la nature (t.1)*. Editions du Seuil, Paris.

[Mour et al., 2013] Mour, A., Kenley, C. R., Davendralingam, N., and DeLaurentis, D. (2013). Agent-based modeling for systems of systems. *INCOSE International Symposium*, 23(1):973–987.

[Mrissa et al., 2015] Mrissa, M., Médini, L., Jamont, J.-P., Le Sommer, N., and Laplace, J. (2015). An Avatar Architecture for the Web of Things. *Internet Computing, IEEE*, pages 30–38.

[Myerson, 1983] Myerson, R. (1983). Mechanism design by an informed principal. *Econometrica*, 51(6):1767–97.

[Paredis et al., 2013] Paredis, C. J., Bishop, C., Bodner, D., Acheson, P., Dagli, C., and Kilicay-Ergin, N. (2013). Model based systems engineering for system of systems using agent-based modeling. *Procedia Computer Science*, 16:11 – 19.

[Paz et al., 2016] Paz, J. F. D., Bajo, J., Rodríguez, S., Villarrubia, G., and Corchado, J. M. (2016). Intelligent system for lighting control in smart cities. *Information Sciences*, 372:241–255.

[Picard, 2004] Picard, G. (2004). *Methodology for developing adaptive multi-agent systems and designing software with emergent functionality*. Phd thesis, Université Paul Sabatier Toulouse III.

[Rincon et al., 2016] Rincon, J. A., Bajo, J., Fernandez, A., Julian, V., and Carrascosa, C. (2016). Using emotions for the development of human-agent societies. *Frontiers of Information Technology & Electronic Engineering*, 17(4):325–337.

[R.S, 2000] R.S, P. (2000). Systems of systems integration (sosi) - a smart way of acquiring army c412ws systems. In *Proceedings of the Summer Computer Simulation Conference*, pages 134–139.

[Russell and Norvig, 2002] Russell, S. J. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, second edition.

[Saaty, 1980] Saaty, T. (1980). *The Analytic Hierarchy Process, Planning, Piority Setting, Resource Allocation*. McGraw-Hill, New york.

[Selberg and Austin, 2008] Selberg, S. and Austin, M. (2008). Toward an evolutionary system of systems architecture. *18th Annual International Symposium of the International Council on Systems Engineering, INCOSE 2008*, 4(1):2394–2407.

[Simon, 1956] Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological Review*, 63(2):129–138.

[Squazzoni, 2012] Squazzoni, F. (2012). *What is Agent-Based Computational Sociology all About?*, pages 1–32. John Wiley & Sons, Ltd.

[Stirling, 2005] Stirling, W. (2005). Social utility functions - part i: theory. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):522–532.

[Stirling and Frost, 2005] Stirling, W. and Frost, R. (2005). Social utility functions - part ii: applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):533–543.

[Tainter, 1988] Tainter, J. A. (1988). *The Collapse of Complex Societies*. Cambridge University Press.

[Wang et al., 2014] Wang, R., Agarwal, S., and Dagli, C. H. (2014). Executable system of systems architecture using opm in conjunction with colored petri net: A module for flexible intelligent and learning architectures for system of systems. *INCOSE International Symposium*, 24(s1):581–596.

[Weyns et al., 2005] Weyns, D., Van Dyke Parunak, H., Michel, F., Holvoet, T., and Ferber, J. (2005). Environments for multiagent systems state-of-the-art and research challenges. In *Proceedings of the First International Conference on Environments for Multi-Agent Systems*, E4MAS'04, pages 1–47, Berlin, Heidelberg. Springer-Verlag.

[Zeigler et al., 2000] Zeigler, B., Praehofer, H., and Kim, T. (2000). *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.