# SApHESIA: An agent-based model and a criticality-based heuristic for cooperatively coupling SoSs

Valérie Camps, Stéphanie Combettes, Teddy Bouziat

# SApHESIA: an Agent-Based Model and a Criticality-Based Heuristic for Cooperatively Coupling SoSs

Valérie CAMPS, Stéphanie COMBETTES, Teddy BOUZIAT

*IRIT Lab, University of Toulouse - Paul Sabatier Toulouse France*

**Abstract**

Problems to solve nowadays have never been so complex and are continuously increasing in complexity. In this context Systems of Systems (SoS) may be a solution but the study of such systems is far from over. An SoS is a complex system characterized by the particular nature of its components: the latter, which are systems, tend to be managerially and operationally independent as well as geographically distributed. This specific characterization led to re-think research fields of classic systems engineering, such as definition, taxonomy, modelling, architecting, etc. SoS architecting focuses on the way independent components of an SoS can be dynamically structured and can autonomously and efficiently modify their interactions in order to fulfil the goal of the SoS and to cope with the high dynamics of the environment.

This paper contributes to the multi-agent and SoS research fields by proposing a new generic SoS model called SApHESIA which considers the SoS main characteristics found in the literature and extends the notion of environment and interactions between component systems. It also proposes 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 measure, local to each component, expressing its difficulties to reach its local goals. In this procedure, the SoS architecture evolves to self-adapt to the dynamics of the environment in which it is plunged, while considering the respective local goals of its components. An instantiation of SApHESIA to two distinct cases studies from different domains (logistics and exploratory missions) is done to obtain two SoSs. These two SoS are then coupled to form a new SoS at an upper level. Evaluations of these SoSs show that their cooperation make each of them more efficient.

*Email addresses:* Valerie.Camps@irit.fr (Valérie CAMPS), Stephanie.Combettes@irit.fr (Stéphanie COMBETTES), teddybouziat@gmail.com (Teddy BOUZIAT)

## 1. Introduction

Problems to solve have never been so complex and are continuously increasing in complexity. To design useful systems in this context, more and more often are used a combination and/or an aggregation of heterogeneous systems that have been independently designed but that are interdependent. Computer scientists have already begun to work on new theoretical foundations, new methodologies and engineering tools in order to face the complexity of designing this kind of systems. After 20 years of research, the study of SoS and all research fields in SoS are still widely open. In 2008, [23] 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 Systems of Systems Engineering (SoSE). How does one extend Systems Engineering (SE) concepts like analysis, control, estimation, design, modelling, controllability, observability, stability, filtering, simulation and so on*". More recently, [20] referenced 113 research fields: concept, definition, taxonomy, model, simulation, etc. 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 have to be re-thought to match with new challenges of SoSE. Furthermore, in 2015, [2] argued that even after 3000 papers and 20 years of research: "*[...] there are signs of immaturity within the research field, 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*". Current research on SoS focuses on a large variety of problems [27] to develop new methods of engineering or architecting the SoS parts.

Generic SoS models exist but have some limitations concerning the definition of the SoS environment and the interactions of the SoS parts (called Component Systems - CSs). SoS architecting consists in finding how CSs can work together and change their interactions to fulfil the SoS goals efficiently and effectively. According to [8], the challenges in architecting SoS come from the managerial and operational independence of CSs. These CSs are not designed to fulfil the global goal of the SoS but the combination of individual CSs may lead to the emergence of unexpected behaviours. The independence of the CSs leads to rethink the methodologies of architecting: contrary to SE where the architecting of a system was mostly static, in an SoS this one is dynamic.

SoS literature shows that Agent-Based Modelling and Simulation are a natural way to develop this new kind of architecture [3] [16] as they enable to describe and test new architecture dynamics by varying the behaviour of CSs in the SoS. [3] 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 "*the cooperative interaction between CSs 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 solutions exist but, to our knowledge, none is based on cooperation between CSs.

This work fits in the field of Systems of Systems (SoS) and Adaptive Multi-Agent Systems (AMAS). Section 2 offers an overview of the principles of AMAS and describes related works regarding definitions and characteristics of SoS as well as existing generic models for SoS, architecting and SoS architecting heuristics. A new generic SoS model called SApHESIA (SoS Architecting HEurIStIc based on Agents) as well as a new heuristic based on the AMAS for SoS architecting are detailed in section 3. Section 4 contains two instantiations of SApHESIA model to two distinct cases studies in order to obtain two SoSs. Each of these SoSs is then individually evaluated. These two SoSs are then a little bit modified in order to become interdependent. They are then coupled using our cooperative heuristic in order to form a new SoS at an upper level, and to evaluate our proposition to the problem of SoS architecting. Finally, we conclude and plan some future works in section 5.

## 2. Context and Related Works

This section explains the main principles of the AMAS approach as well as the issues that still remain to be solved. These issues being tightly linked with the SoS problematic, an overview of this field is then given.

### *2.1. Principles of AMAS Approach and Context*

Beyond the power of computers, the complexity of computer applications has been growing for several decades. This is due to the combination of several characteristics such as the number of interacting components, the distribution of control and processing, the presence of non-linearities between these components, the openness of the systems, as well as the dynamics of their environments. Designing systems that self-organize during their activity is a promising approach to guarantee robustness and adaptation to complex systems integrating the previous characteristics. Researchers have long been inspired by the self-organization of natural systems which leads to emerging phenomena. Currently, much research in multi-agent systems focuses on the discovery of relevant self-organization solutions guiding the agent's behaviour at the micro-level, to obtain at the macro-level the system behaviour desired by the designer.

#### *2.1.1. Principles of the AMAS Approach*

The AMAS approach [15] is used to design adaptive multi-agent systems enabling to solve complex problems that can be incompletely specified and for which an *a priori* known algorithmic solution does not exist. It is a self-organizing approach that considers the system as composed of parts (i.e. agents). It focuses on the local behaviour of each of these agents to make them adaptive (to their local environment), while ensuring that the collective behaviour emerging from interactions between agents is the expected one. Each agent has a local perception of its environment, limited reasoning skills as well as local

capacities for acting in its environment. An agent is autonomous and has a local goal to achieve.

To this end, in the AMAS approach, each agent pursuing a *perceive-decide-act* lifecycle must have a local and **cooperative** behaviour. The definition of cooperation is not a conventional one that considers resource sharing or working together. It is based on the notion of criticality, defined as the "*distance between the current situation and the local goal of the agent*" [18]. Thus, the further the agent is from its goal, the more critical it considers its current situation to be. Each agent type locally computes its criticality thanks to a domain-dependent function. As the overall function provided by the AMAS cannot always be formally defined due to the complexity of the system to be designed, the criticality does not directly depend on this overall function. An agent is cooperative if it acts to help the most critical agent (with the highest criticality) of its neighbourhood. Thus, all agents in an AMAS attempt to continually reduce the criticality of the agent that, in their neighbourhood and according to their local point of view, is the most critical (possibly itself). At the same time, they simultaneously prevent another agent from being more critical than the most critical one. If an agent is not able to help the most critical agent of its neighbourhood, it may help the immediately less critical agent. Thus, doing so, it hopes that the reduction of the criticality of this agent will lead it to help the most critical agent. Specifically, agents try to always remain in a cooperative state from their point of view allowing the overall system to converge to a cooperative state in its environment: the system is then *functionally adequate*. The presented process corresponds to a typical decision process of a generic AMAS agent. But the definition of non-cooperation situations that can occur, their detection and the actions that agents must take to resolve them, are not generic. Depending on the problem to be solved, the designer of an AMAS has (*i*) to identify the agents in the system (the entities possibly facing non-cooperative situations), then (*ii*) determine the ideal cooperation (nominal behaviour) from their local point of view and (*iii*) define their cooperative behaviour. The cooperative behaviour consists in finding the non-cooperative situations agents may face during their functioning as well as defining the actions that they must take to return to a cooperative state.

The AMAS approach has been used to solve several different problems both by their objective (problem-solving, optimization, simulation) and by the domains concerned (flood forecasting, collective robotics, ontology construction, and so on...). The encouraging obtained results have prompted us to promote the use of this approach and to develop the ADELFE methodology [26] to design adaptive systems according to the AMAS approach. ADELFE only concerns applications in which self-organization allows the emergence of a solution from the interactions between agents. It provides a tool to determine whether the use of AMAS is relevant to the problem to be solved. If this is the case, it helps the user to define the components of his system (here the agents) and guides him throughout the software development cycle (from needs analysis to system testing and validation).

*2.1.2. Context and Objectives*

Some works, notably those dedicated to user assistance, have shown that the AMAS responding to the problem is composed of several interdependent subsystems, often designed independently, each one being in charge of a partial implementation of this assistance (e.g. construction and maintenance of the user profile, definition and management of the context, etc.). A lock to be lifted is then to study the coupling of these subsystems. More precisely it is a question, in the long term, of automating the integration of several adaptive (sub)systems (built according to the AMAS approach), which can be either composed (vertical dimension), associated (horizontal dimension) or both.

This paper deals with the issue of coupling AMASs which is tightly linked with SoS problematic (autonomy of their parts, dynamic interactions between parts, complexity of the parts, heterogeneity of the parts, the important notion of the environment).

*2.2. An overview of Systems of Systems*

Even if a lot of definitions of SoS has been given from various fields ([23], [25]), no consensual definition and common characteristics of SoS is proposed. During these last ten years, research efforts have been put on SoS taxonomy and theoretical foundations in order to find a generic definition [24], [22], [9].

*2.2.1. Main Properties and Characteristics of SoS*

Maier was the first to characterize SoS in [24]. He gave five main properties distinguishing an SoS from a traditional complex system, which were accepted by researchers working on SoS ([22],[27]). These five properties are operational independence, managerial independence, geographical distribution, emergent behaviour and evolutionary development. Furthermore, Gorod [19], Sauser [9] [5] and Bjelkemyr [7] have converged on an SoS characterization that distinguishes classic systems from SoS. They propose the ABCDE characteristics meaning Autonomy, Belonging, Connectivity, Diversity and Emergence. Ballegaard Nielsen & all [6] propose a survey on model-based techniques in SoS engineering and identify challenges for research in this field. They notably notice a lack in the emergence of behaviour in modelling and simulation, and in the evolution of SoS in simulation.

*2.2.2. Existing Generic Models for SoS*

To our knowledge, few generic SoS models exist in literature. Based on set theory and ABM (Agent-Based Model), the only two following models we found enable to have formal definitions of what is a SoS. We discuss their limitations through the nine evaluation criteria we propose.

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.

*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** and *(5)* **interactions between component systems**. Finally, the following criteria concern the ability to model the SoS and the environment. *(6)* **SoS model** concerns the ability to model a SoS as an autonomous entity with its own goals. Indeed, as in a *directed* or *acknowledged* SoS, a central management exists, it has to be modeled. *(7)* **Dynamic environment model** concerns the ability to model a dynamic environment. Indeed, a SoS evolves in a dynamic environment. *(8)* **Global expressiveness** concerns the ability to express interesting SoSs. By "interesting", we mean expressive enough to be able to model concrete and relevant examples of SoS. *(9)* **Metric definitions** 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 from it.

*Generic Model of SoS Based on Set Theory.* This model, introduced in [4], uses set theory and first order logic. It has been created to model the five keys characteristics developed in [9] 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 and $E_i$ a set of non-defined elements. $A_i$, $G_i$ are sets of atomic elements. No details are given 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 and $S_1, .., S_n$ are component systems.

In this model, the characteristics, autonomy, belonging, connectivity and diversity of each system have been formally defined [4] while emergence and connectivity are not represented.

This 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.

*Agent-Based Wave Model.* In [1], 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, and $ability_i$, 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 or not to the SoS). 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 $C$, and $P$ a set of desired performances on capabilities of 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. 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 [1], 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 provided). Finally, there is no example of use of threats and national priorities factors.

Agent-Based Wave model 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

| | 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: Evaluation of existing generic SoS model

(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.
Table 1 sums up 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 (- -).

We can see that these models have a lack concerning their realism and that they do not meet some of 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.

*2.2.3. SoS Architecture, SoS Architecting and SoS Architecting Heuristics*

In the field of SoS Engineering, SoS architecture refers to the set of CSs of the SoS and their interactions. Wang & al. [28] define SoS architecture as "*an arrangement for the set of constituent systems, rules and behaviours that govern an individual system's functions. Architecture also describes how these systems' capabilities contribute to a larger goal*". An SoS evolves in a dynamic environment and its goals may change over time. Thus, the CSs of the SoS and their interactions may change. That is the reason why an SoS cannot have a static architecture: it has to adapt its own architecture during time.

In this paper, we call "SoS architecting" the process that enables to model and simulate SoS architectures that dynamically evolve over time.

SoS architecting heuristic consists in finding how CSs can work together and change their interactions to efficiently and effectively fulfil the SoS goals. Due to the dynamics of the environment and to the large set of solutions fields, finding

the optimal architecture during the time for an 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 the literature to find a satisfactory architecture for a given state of the environment. To find such an architecture, designers can use the managerial and operational independences of CSs to propose a decision strategy (such as collaboration) at the CS level. For directed and acknowledged SoS, designers use central management to coordinate and/or directly use CSs resources. For example, central management can decide to add or remove CSs during functioning.

Two most developed and documented architecting heuristics for SoS can be found in the literature. The first one is based on a collaborative SoS formation [16]: CSs ($a$) act collaboratively with each other to avoid counter-productive actions between them and ($b$) they are able to negotiate the gain and the constraints of collaboration with the SoS. To reach ($a$), this architecting heuristic uses satisficing games theory and to reach ($b$), mechanism design and Multi-Criteria Decision Analysis (MCDA) are used [16]. The second one is based on Agent-Based Wave Model ([1], [17]). It aims at proposing a generic method to find an SoS architecture satisfying constraints coming from an external environment (SoS total cost, development time, threats and so on). To achieve this goal, the method is composed of several steps that are repeated through *waves* until a satisfying architecture is found. It is based on interviews with stakeholders, fuzzy logic and a genetic algorithm. We have evaluated these two existing SoS architecting heuristics through the seven following criteria: computationality, genericity, dynamics, openness, computational cost, decentralization and cooperation. This evaluation can be found in [10].

We can retain from this analysis that both heuristics fail regarding the total decentralization of the process thus disabling to architect collaborative and virtual SoS. Moreover, they are based on a heavy process that makes difficult to consider the dynamics and the openness. Finally, cooperation between CSs is not totally represented while it is the basis for the SoS emergent behaviour. Furthermore, to our knowledge, SoS literature does not propose generic SoS models able to model a dynamic environment as well as interactions between CSs ([1], [4]).

For all these reasons, we propose a fully decentralized, open SoS architecting heuristic based on cooperation between component systems. We also propose a new SoS model called *SoS Architecting HEuriStIc based on Agents (SApHESIA)*, based on the set theory and enabling to use cooperation as an architecting heuristic. The cooperation enables agents (so, CSs) to self-organize and self-adapt to the dynamics of the environment.

This paper considers SoSs composed of interacting systems, called component systems (CSs). These CSs are autonomous and may evolve in a dynamic environment. CSs and SoS have operational and managerial independences. They may be geographically distributed and their dynamic interactions may give rise to an emergent behaviour at the SoS level. These interactions also adapt themselves over time to the evolution of the SoS and of the environ-

mental constraints. Finally, an SoS is open (CSs may join or leave it during functioning).

Next section presents the different elements of the SoS generic model we propose.

## 3. Presentation of SApHESIA

This section describes the SApHESIA model as well as the heuristic we propose to architect SoS, and briefly discusses SApHESIA tools for SoS architecting.

### *3.1. SApHESIA Model*

SApHESIA uses the multi-agent system paradigm and enables to model an SoS by focusing on its environment as well as on the interactions between its CSs. It overall consists of the SoS (made of CS and goals) as well as its environment (made of entities and rules).

#### *3.1.1. Component System Model*

A **component system** $CS$ is the smallest part of an SoS and represents an element of the second $S$ of SoS. It is defined as $CS = \{T, R, Acq, L, F, G, Cost\}$ where $T$ is the type of CS within the SoS.
$R = \{R_1, ..., R_n\}$ is the set of its resources.
$Acq = \{Acq_1, ..., Acq_q\}$ is the set of acquaintances of CS.
$L = \{L_1, ..., L_q\}$ is the set of links of CS with other component systems.
$F = \{F_1, ..., F_m\}$ is the set of its functionalities.
$G = \{G_1, ..., G_p\}$ is the set of its goals.
$Cost \in R$ is the cost of CS.
More precisely, the **type** is a string that represents a kind of a CS in the SoS.
A **resource** $R_i$ is a structure $R_i = \{type, quantity\}$ which represents the passive elements (i.e. without effector) in the SoS.
An **acquaintance** is an oriented association between two CSs ($Acq_{CS} = \{CS'\}$) meaning that the first $CS$ knows the second $CS'$.
A **link** is an oriented channel ($L_{CS} = (CS', soa)$) enabling exchanges (of communication or resources) from $CS$ towards $CS'$, where $soa \in ]0, 1]$ is the strength of this association (i.e. the quality of the channel). A link between $CS$ and $CS'$ refers to the communication channel enabling them to directly communicate while an acquaintance between $CS$ and $CS'$ refers to the knowledge (by $CS$) of the existence of $CS'$ without having the means to communicate with it.
A **functionality** $F$ is an effector that affects $(i)$ its own resources and/or the ones of other CSs as well as $(ii)$ the links between CSs. $F = \{f, t, p\}$ where $t$ is the execution time of $F$, $p \in [0, 1]$ is the performance (i.e. the probability of success) of $F$ and $f = Conditions \rightarrow Effects$ is a function of $F$. $Conditions$ is a set representing $(i)$ a certain quantity of resources or $(ii)$ the existence of a link between two CSs or $(iii)$ the existence of a CS enabling $f$ to be executed. Once $f$ is executed, $Effects$ is applied. $Effects$ is a set representing either a

certain quantity of resources or the existence of a link between two CSs affecting the SoS or the environment. The SoS designer needs to define functionalities that are so considered as inputs of the SoS.

A **goal** is a special state that CS tries to reach with a given priority. This state can be $(i)$ to own a defined quantity of a type of resources or $(ii)$ to create a link between two CSs. A goal $g \in G$ is defined as $g = \{Re\ \texttt{Op}\ Qu, Pr\}$ 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, $\texttt{Op} \in \{=, \neq, <, >, \leqslant, \geqslant\}$ is the comparison operator in order to compare $Qu$ to the current quantity of $Re$ and $Pr \in \mathbb{N}^+$ 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.
A **cost**, associated with each CS, represents its own charge and the charge for the SoS when it uses it. This charge depends on the problem. The cost indicates how much is the use of a particular CS functionality relatively to the other CSs.

As a CS may join/leave an SoS anytime, acquaintances are not static. An acquaintance indicates that a CS knows another one but we clarify that a CS does not need to know all the CSs of the SoS. Indeed, a CS knows the CSs of its neighbourhood, which themselves know other CSs of their own neighbourhood. The neighbourhood may be defined by the perception field of a CS. The perception field of a CS is all the entities (active or passive) the CS can locally observe in its environment. Even if openness has not been addressed in our experimentation this issue can be implemented using the *restricted relaxation* [14] . It is a cooperative routing protocol that is able to learn the existence of unknown agents. It consists in re-transmitting information to a neighbour who may be interested in that information (or if it is a request, likely to respond to it) without modifying the original sender. In other words, the re-transmitting agent plays the role of a simple relay: it is not an intermediary. The receiver can thus learn of the existence of a peer that is interested by (or in search of) a particular piece of information and so increases its acquaintances.

#### *3.1.2. SoS Model*

We propose to model an SoS as $SoS = \{\mathbb{S}, \mathbb{G}\}$ where $\mathbb{S}$ is the set of CSs and $\mathbb{G}$ is the set of goals that may be a subset of the goals of the CSs of $\mathbb{S}$ according to the type of management of the SoS (centralized or not).

#### *3.1.3. Environment Model*

An environment is defined as $E = \{Entities, Rules\}$
where $Entities$ is the set of the **entities** that represent active independent objects able to affect the environment or the SoS itself and $Rules$ is the set of **rules** that model exterior constraints (out of the SoS) that apply to the SoS.

An **entity** is an active independent object that does not belong to the $SoS$. It is defined as $E_i = \{T_E, R_E, Acq_E, L_E, F_E, G_E\}$ where
$T_E$ is the type of $Entities$.
$R_E = \{R_{E1}, ..., R_{Eq}\}$ is the set of **resources**,
$Acq_E = \{Acq_{E1}, ..., Acq_{Er}\}$ is the set of **acquaintances**,
$L_E = \{L_{E1}, ..., L_{Es}\}$ is the set of **links**.

$F_E = \{F_{E1}, ..., F_{Et}\}$ is the set of **functionalities**.
$G_E = \{G_{E1}, ..., G_{Eu}\}$ is the set of **goals**.
An entity can be linked to another entity or to a CS contrary to CS that can only be linked with each other.

A **rule** ($Rule = \{Conditions \rightarrow Effects\}$) models how the environment reacts, evolves and interacts with the SoS. A rule affects some the entities in the environment some CSs of the SoS that fulfil the $Conditions$.
The environment model differs from the SoS one as it represents all the entities being outside the SoS and possibly impacting it. For example, in a "military" SoS, the opponents are considered as the environment, as well as the mountains in which a vehicle may hide.

### *3.2. A Heuristic based on Criticality to Architect SoS*

This section presents our decentralized decision algorithm based on the Adaptive Multi-Agent System (AMAS) approach that uses cooperation as a social behaviour between agents [15]. We propose to model a CS as an autonomous entity (agent) that has to choose the most cooperative action (here a functionality).

#### *3.2.1. Criticality as a Metric of Cooperation*

A generic and computable definition of criticality is presented as a function taking as inputs, a subset of perceptions and its goals concerning these perceptions [13]. Perceptions enable the agent to decide the action(s) to execute (during its perceive-decide-act life-cycle) to reach its goal. The perception field of an agent is predefined by the designer and domain-dependent. Formally, the criticality $C$ of an agent $i$ at time $t$ is

$$C_i(t) = f(p_1(t), ..., p_n(t), S_{goal}) \tag{1}$$

where $\{p_1(t), ..., p_n(t)\}$ is a subset of the current perceptions of the agent $i$ and $S_{goal}$ is the final state the agent $i$ wants to reach.

To know the result of one of its actions on its own criticality, an agent needs to compute the anticipated criticality of this action. Formally, the anticipated criticality of an agent $i$ for an action $a$ is defined as:

$$CA_i(t, a) = C_i(t) + Eff_i(a) \tag{2}$$

with $C_i(t)$ the criticality of $i$ at time $t$. $Eff_i(a)$ is a function that computes the impact of the action $a$ if the agent $i$ performs $a$. The anticipated criticality can be then calculated for a sequence of actions $A = \{a_1, a_2, ..., a_n\}$ as:

$$CA_i(t, A) = C_i(t) + \Sigma_{j \in [1,n]} Eff_i(a_j). \tag{3}$$

The definition of this concept has been adapted to SApHESIA according to the resources and goals of a CS. The *current state* of a CS is represented by the perceptions of its current resources and *the state it wants to reach* is represented by its goals. A criticality for each goal $g$ in the set of goals $G$ of the CS represents

the distance of fulfilment of $g$. 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. Thus, the criticality $C_g(t)$ of the goal $g$ at time $t$ can be defined by :

$$C_g(t) = \begin{cases} (a) : 1 - \dfrac{1}{e^{\alpha \times \Delta_g(t)}} & (c) : \dfrac{atan((\alpha \times \Delta_g(t)) + \dfrac{\Pi}{2}}{\Pi} \\ (b) : \dfrac{1}{e^{\alpha \times \Delta_g(t)}} & (d) : 1 - \dfrac{atan((\alpha \times \Delta_g(t)) + \dfrac{\Pi}{2}}{\Pi} \end{cases} \tag{4}$$

(a) the goal $g$ has to be equal to a given quantity $Qu$ of $Re$ (resource).
(b) the goal $g$ has to be different from a given quantity $Qu$ of $Re$.
(c) the goal $g$ has to be smaller or equal to a given quantity $Qu$ of $Re$.
(d) the goal $g$ has to be greater or equal to a given quantity $Qu$ of $Re$.
So each goal has a criticality function, which depends on the operator of comparison ($=$, $\neq$, $<$ or $\leq$ and, $>$ or $\geq$) associated with the goal.

$\Delta_g(t)$ is the difference between the given quantity $Qu$ and the current amount of resource $Re$ of the CS. $\alpha$ is a coefficient influencing the shape of the curve.

These sigmoid functions have been chosen as their shapes correspond to the meaning of each goal. The shape of these functions can be changed according to the variation of the parameter $\alpha$. It enables to control how the goal value is important to reach or to avoid. An example for two values of $\alpha$ ($\alpha = 0.01$, $\alpha = 0.03$) is given on figure 1. The red functions represent decreasing criticality when reaching objective ($g.Op \in \{=\}$). A high $\alpha$ parameter value for the red function lead a component system to stay really critical even if it is close to its goal.

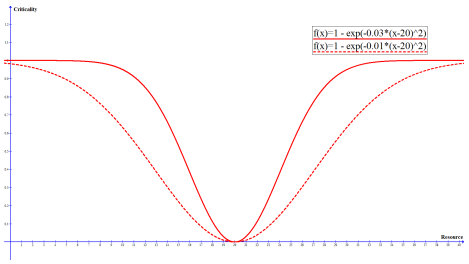


Figure 1: Example of a goal criticality: reach a value ($g.Op \in \{=\}$) with two values of $\alpha$

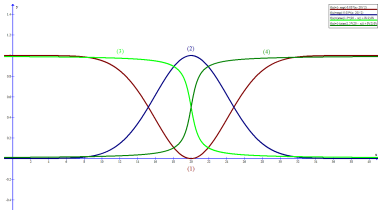

Figure 2: Examples of a goal criticality: (1) to reach a value ($g.Op \in \{=\}$); (2) to avoid a value ($g.Op \in \{\neq\}$); (3) to be equal to or greater than a value ($g.Op \in \{>, =>\}$); (4) to be equal to or less than a value ($g.Op \in \{<, =<\}$)

In figure 2 (where $\alpha = 0.03$), the blue function represents an increasing criticality when reaching an unwanted value $g.Op \in \{\neq\}$. The light green one represents a decreasing criticality when a resource is greater than a given threshold $g.Op \in \{>, =>\}$. The dark green one represents a decreasing criticality when a resource is less than a given threshold $g.Op \in \{<, =<\}$. In our evaluations (cf. section 4), alpha is equal to 1.

To calculate the criticality $C_{CS}(t)$ of a component system $CS$ 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:

$$C_{CS}(t) = \frac{\sum_{g \in G}(C_g(t) \times g.Pr)}{\sum_{g \in G}(g.Pr)} \tag{5}$$

with $g.Pr$, the priority of goal $g$ (cf. 3.1.1).

Finally, this formula of the criticality for component systems is adequate with the first definition of the criticality (1): 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.

**Algorithm 1** Cooperative Decision Algorithm of $CS_i$

**Require:** a CS $CS_i$, its functionality set $F_i$
**Require:** its acquaintance set $Acq_i$, its links set $L_i$
1: **for all** $f \in F_i$ **do**
2:  **for all** $CS_j \in L_i \cup Acq_i$ **do**
3:  $CoopTable(f)(CS_j)$ += $askAnticipatedCrit(CS_j, f)$
4:  **for all** $CS_k \in (L_i \cup Acq_i)$ **do**
5:  $CoopTable(f)(CS_j)$ += $askAnticipatedCrit(CS_j, f)$
6:  **end for**
7:  Sort $CoopTable(f)(CS_j)$
8:  **end for**
9: **end for**
10: **return** $minmaxFunc(CoopTable)$ {Choose f that minimizes the maximum of the criticalities stored in CoopTable}

*3.2.2. Cooperative Decision Algorithm for Component Systems*

We propose to model a CS as an autonomous entity (agent) having a *perceive-decide-act* life-cycle. Each agent must be cooperative with its neighborhood and, for this, it must choose the most cooperative action. That is why, the proposed decision algorithm (Algorithm 1) allows to choose the most cooperative action (here a functionality) during a $CS$'s *decision* phase. Concretely, each component system $CS_i = \{T_i, R_i, Acq_i, L_i, F_i, G_i, Cost_i\}$ has to construct its cooperative table and to choose the most cooperative functionality. Lines of the cooperative table to be built contain the anticipated criticalities for the

neighbourhood of $CS_i$ (given by $L_i \cup Acq_i$) depending on the potential execution of each functionality $f$ of $CS_i$.

Each $CS_i$ asks for the anticipated criticality of its neighborhood (in $Acq_i$ and $L_i$) for each available functionality $f \in F_i$. This request is made by the function $askAnticipatedCrit(CS_j, f)$ (line 3 of algorithm1). This function has a component system $CS_j$ and a functionality $f$ as inputs and returns the value of the anticipated criticality of $CS_j$ if $CS_i$ applies $f$ on $CS_j$. This anticipated criticality is then saved in a new line (identified by a couple $(CS_j, f)$) of the cooperative table ($CoopTable$). Then, the anticipated criticalities of other component systems (including $CS_i$) are added to this line through the function $askAnticipatedCrit(CS_j, CS_k, f)$. This function returns the anticipated criticality of $CS_k$ if $CS_i$ applies $f$ on $CS_j$ (line 5). The lines of $CoopTable$ contain the anticipated criticalities of its entire neighborhood (including $CS_i$) if $CS_i$ applies $f$ on $CS_j$. Finally, all the table lines are sorted from the highest criticalities to the lowest one (line 7). Once the cooperative table is built, $CS_i$ chooses the functionality to carry out 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 returns the functionality $f$ to apply. The reader may consult [10] for more details concerning the $minmaxFunc(CoopTable)$ function.

### *3.3. Outlines of SApHESIA Tools for SoS Architecting*

As SoS architecting tools are domain-dependent and/or not available in the public domain, we propose a set of tools in order to model and run easily SoS experiment through the SApHESIA model. This set is mainly composed of two elements :

- the *SoS and Environment Generator* that consists of a *SML Parser & Compiler* and a *GUI generator* which jointly enable the user to manually create and/or modify a SoS and/or its environment.
- the *SApHESIA core architecting* that consists of SApHESIA Engine and Viewer which respectively runs a loaded SApHESIA model and visualizes results of the tests.

Without going into the details, the GUI generator and the language called SML (SApHESIA Modeling Language) enable to translate a SApHESIA model into a computable model usable by our tools. The GUI generator also enables to change dynamically the SoS composition by adding or removing component systems or entities and also by changing the different properties of a component system (resources, links, etc.). Thanks to the viewer, user can display useful data to validate our cooperation-based SoS architecting heuristic through different scenarios.

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 elements (component system, entity, rule and so on) of the SApHESIA model. 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.

SApHESIA has been implemented in JAVA language because it is a well-maintained programming language based on the Object paradigm close to the agent paradigm. Indeed, the Object paradigm owns the concept of properties and methods enabling to represent the $Perceive - Decide - Act$ cycle of an agent. The main classes used to implement SApHESIA tools as well as details of these tools can be see in [10].

This set of tools has been used for implementing the whole of our experiments on SoS Architecting.

## 4. Instantiation of SApHESIA and Coupling Evaluation

This section presents experimentation about reuse and interdependency between existing SoSs. The aim is to evaluate that two SoSs independently built using SApHESIA can serve each other (cooperate) and then together increase their own efficiency. As we did not find any case studies with real and usable data to evaluate our propositions, we evaluate them through a case study that we developed and another one found in the literature. The first case study (CoCaRo) is a resources transportation system we have defined to validate two contributions: ($i$) the use of SApHESIA to define an agent and ($ii$) the metric of criticality enabling cooperation between agents. The second case study concerns the Missouri Toy problem, a communication transportation problem. It has been chosen because ($i$) it has already been studied in the SoS field and ($ii$) it is a good starting point for testing our heuristic concerning its robustness and its capacity to be functionally adequate. The implementation of these two case studies with the AMAS approach as well as their evaluations have already been published in [11] and [12]. The novelty of this paper concerns the transformation of both of these AMAS into SoS as well as the coupling of these two SoSs.

The first two subsections focus on the adaptation of each case study (CoCaRo and UAV) in order to be able to obtain two SoSs and then on their coupling to form a third SoS. The third subsection presents this coupling.

### *4.1. Adaptation of CoCaRo problem as a Robots SoS*

CoCaRo (Color Carrier Robot) is a system that models and simulates a resources transportation system by mobile robots.

#### *4.1.1. Description*

To stay alive, a robot (possibly red, blue or green) has to find and pick up a box (also being red, blue or green) and then to drop it in the area (called the nest) of the same colour than the box. Each robot has an initial amount of energy that it consumes at each movement. However when it drops a box in the appropriate nest, it receives a reward of energy to remain longer alive.

The value of the energy reward depends on the colours of the dropped box and of the robot: it is maximum when the robot and the dropped box both have the same colour. The instantiation of SApHESIA to this case study as well as obtained experimental results dealing with the evolutionary part of the system's behavior can be found in [11].

To be able to experiment with the coupling of these SoSs, CoCaRo needs to be modified in order to be an SoS and to have interdependencies with a second SoS. That is why, a robot is now just a resource of the robots group that will be used in a functionality to create energy. So a robot is not an active entity anymore. The robots are aggregated into colour groups, each of them being a CS. So three CSs have been defined (red, blue and green). Each CS consumes a certain rate of energy per time unit and can use boxes directly to create energy. As the aim is to study the interdependencies between SoSs, the dynamic of exchange between robots are not needed in this simulation.

The **Robots Group CS model** is defined as

$$CS_{RG} = \{T_r, R_r, Acq_r, L_r, F_r, G_r, Cost_r\} \tag{6}$$

where

- $T_r = \{Red, Green, Blue\}$ is the type set and represents respectively the red, green or blue group of robots.
- $R_r$ is the resource set defined as: $R_r = \{R_{Turn}, R_{Robot}, R_{Energy}, R_{BoxRate}, R_{EC}, R_{Box}, R_{HighValue}, R_{LowValue}\}$, where $R_{Turn}$ is used to save the current simulation step. $R_{Robot}$ represents the number of active robots in the group. $R_{Energy}$ represents the global energy level of a $CS_{RG}$. $R_{BoxRate}$ represents the rate of boxes apparition in a $CS_{RG}$, the boxes appearing randomly in the environment. $R_{EC}$ represents the energy consumption of a $CS_{RG}$. $R_{Box}$ represents the number of available boxes in a $CS_{RG}$, i.e. the number of boxes that are perceived and brought back by the $CS_{RG}$. Finally, $R_{HighValue}$ and $R_{LowValue}$ are respectively the high reward value and the low reward value when the $CS_{RG}$ uses a box (in CoCaRo, the reward depends on the colour of the dropped box and the colour of the robot).
- $Acq_r$ starts empty and does not evolve over time because each group of robots are linked to others; $Acq_r = \{\emptyset\}$;
- $L_r = \{CS_{RG}\}$ is the set of links; as the links connect all the $CS_{RG}$ (each robot can exchange boxes or information whatever their colors) and $L_r$ subsumes $Acq_r$, $Acq_r$ is empty.
- $F_r = \{F_{CreateEnergy}, F_{GiveBox}\}$ is the set of functionalities. $F_{CreateEnergy}$ represents the ability of the $CS_{RG}$ to transform boxes into energy. The $CS_{RG}$ is able to create energy if it owns at least one box and one robot ($\{R_{Box} \geqslant 1\}, \{R_{Robot} \geqslant 1\}$). $F_{GiveBox}$ represents the capacity of a $CS_{RG}$ to give boxes to another $CS_{RG}$.

- $G_r = \{G_{Box}, G_{Energy}\}$ is the set of goals. $G_{Box}$ represents the goal for a $CS_{RG}$ to own as much boxes as possible. $G_{Energy}$ represents the goal for the $CS_{RG}$ to maximize energy.
- $Cost_r$ represents how much is the use of a $CS_{RG}$. In this case study the SoS is closed (there is no entry / no exit of new $CS_{RG}$) so $Cost_r = 0$.

The **CoCaRo SoS model** is defined as

$$SoS_{CoCaRo} = \{Red, Green, Blue, G_{Box}, G_{Energy}\} \tag{7}$$

where *Red*, *Green* and *Blue* are $CS_{RG}$ (RG means Robots Group). The goals of the CoCaRo SoS ($G_{Box}$ and $G_{Energy}$) are to keep all its $CS_{RG}$ alive, which means that they must have roughly an identical criticality.

The **CoCaRo SoS environment model** does not contain any entity (because boxes are directly considered as resources). It only consists of 4 rules that simulate the boxes apparition and the energy consumption. It is defined as :

$$\begin{aligned} E_c = \{&Rule_{Apparition}, Rule_{GreenConsumption}, \\ &Rule_{BlueConsumption}, Rule_{RedConsumption}\}. \end{aligned} \tag{8}$$

The rule $Rule_{Apparition}$ enables to generate new boxes in the $CS_{RG}$. Basically, at each turn of the simulation, if a randomly generated number is lower than the $R_{BoxRate}$ resource, the resource $R_{Box}$ increases. The three other rules enable to change the consumption of energy of each $CS_{RG}$ (robots group) in order to introduce dynamics in the environment.

*4.1.2. Cooperative Behaviour*

As we previously saw in the $CS_{RG}$ model, the CoCaRo SoS goal $G_r$ consists of two goals for a $CS_{RG}$: $G_{Box} = \{R_{Box} > 0\}$ (to have as much available boxes as possible) and $G_{Energy} = \{R_{Energy} > 20^1\}$ (to have as much energy as possible). In accordance with 3.2.1, the criticalities of the two goals of a $CS_{RG}$ at time $t$ are both defined by the formula 4 (d). Furthermore, a priority of 1 was given to $G_{Energy}$ and a priority of 0.01 was given to $G_{Box}$. A low priority has been associated with the $G_{Box}$ goal so that $CS_{RG}$ do not spend their time picking up boxes.

The criticality of a $CS_{RG}$ can be then calculated with the weighted average of all its goals criticalities (formula (5)). According to SApHESIA, each $CS_{RG}$ is cooperative and pursues the cooperative decision algorithm 1 (section 3.2.2).

*4.1.3. Experimentations*

In this new version of CoCaRo, the robots are not active entities, conversely to [11]; their actions (bringing back boxes and convert boxes into energy) are

[1] 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.

Table 2: Initialization of the three $CS_{RG}$

| | *Green* | *Blue* | *Red* |
|---|---|---|---|
| $R_{Turn}$ | 0 | 0 | 0 |
| $R_{Robot}$ | 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 |

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 [11] (with active robots).

Three $CS_{RG}$ have been instantiated: *Green*, *Red*, *Blue*. Each of them has been initialized with the different values that are summed up in the table 2. In this table, $R_{LowValue} = 0.015$ and $R_{HighValue} = 0.04$. These values have been chosen arbitrarily. The idea is to begin with the *Red* $CS_{RG}$ with a high value of energy consumption and validate that the two other $CS_{RG}$ will help it by giving extra boxes.

In the scenario given in [11], in order to make comparisons, we experimented the system in 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. The aim, here, is to reproduce the hostile environment. Thus each $CS_{RG}$ has, during a fixed period of time, only boxes that are efficient for it. During the first 1000 steps of the simulation, the *Red* $CS_{RG}$ 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 (steps 1000 to 2000), the *Green* $CS_{RG}$ will have a high value of energy consumption ($R_{EC} = R_{HighValue}$) and *Red* $CS_{RG}$ will come back to a low consumption (thanks to the rule $Ru_{GreenComsumption}$). Finally, during the last 1000 steps of the simulation (steps 2000 to 3000), the *Blue* $CS_{RG}$ 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 $Ru_{BlueComsumption}$).

Obtained results can be seen in figure 3 and especially in the graphs 1 & 2. The graph 2 shows that the dynamics of the environment impacts the functioning of each $CS_{RG}$ 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 cycles 1000 and 2000 and between cycles 2000 and 3000. Secondly, the dynamics of the system CoCaRo with cooperation presented in [11] is always present. Indeed, each $CS_{RG}$ tends to balance the mean level of energy of each other. This fact can be seen with the $R_{Energy}$ and criticality graphs(graphs 1 & 2 in fig.3). Even if a $CS_{RG}$ is more constrained by the

environment, the two others, thanks to the cooperation algorithm, are able to help it. The second graph in figure 3 also shows that all the $CS_{RG}$ finish the simulation by having a very low level of energy.

### *4.2. Adaptation of UAVs as a UAVs SoS*

The Unmanned Aerial Vehicles (UAVs) case study aims at avoiding hazards while reaching targets.

#### *4.2.1. Description*

In this case study, 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 instantiation of SApHESIA to this case study as well as obtained experimental results dealing with the evolutionary part of the system's behavior can be found in [12]. In this new version of UAV SoS, the aim of the experimentation is to present the global functioning of an independent SoS using the cooperation mechanism. Conversely to [12], UAVs have to cooperatively take boxes to convert them into energy. In this adaptation, the SoS is composed of 3 CSs ($UAV_1$, $UAV_2$ and $UAV_3$) representing UAVs that are able to navigate to boxes and to convert them into energy (such as a robots in CoCaRo).

The **UAV CS model** is defined as

$$CS_{UAV} = \{T_u, R_u, Acq_u, L_u, F_u, G_u, Cost_u\} \tag{9}$$

where

- $T_u = UAV$,
- $R_u = \{R_{Energy}, R_{Box}, R_{EC}, R_{DistToBox}\}$ is the set of resources where $R_{Energy}$ represents the energy level of the $CS_{UAV}$. $R_{Box}$ represents the number of available boxes for the $CS_{UAV}$. $R_{EC}$ represents the global energy level. $R_{DistToBox}$ represents the distance of the $CS_{UAV}$ to a box.
- $Acq_u = \{\emptyset\}$ is the set of acquaintances; it is empty because each $CS_{UAV}$ is linked with all the other $CS_{UAV}$.
- $L_u$ is the set of links. It depends on the considered $CS_{UAV}$: an $CS_{UAV}$ has two links towards the two other $CS_{UAV}$.
- $F_u = \{F_{CreateEnergy}, F_{GiveBox}, F_{MoveToBox}, F_{TakeBox}\}$ is the set of functionalities, where $F_{CreateEnergy}$ represents the capacity of an $CS_{UAV}$ to convert a box into energy. $F_{GiveBox}$ represents its capacity to give a box to another $CS_{UAV}$. $F_{MoveToBox}$ represents its capacity to go to a box and decreases the resource $R_{DistToBox}$. $F_{TakeBox}$ represents the capacity to take a box when $R_{DistToBox}$ is equal to 0.

- $G_u = \{G_{Box}, G_{Energy}, G_{DistToBox}\}$ is the set of goals. $G_{Box} = \{R_{Box} > 0\}$ represents the goal for the $CS_{UAV}$ to own as much boxes as possible. $G_{Energy} = \{R_{Energy} > 0\}$ represents the goal for the $CS_{UAV}$ to maximize energy. $G_{DistToBox} = \{R_{DistToBox} == 0\}$ represents the goal to minimize the distance of the $CS_{UAV}$ to a box.
- $Cost_u$ represents how much is the use of a $CS_{UAV}$; as this case study is closed, (no CS may enter or exit the SoS), $Cost_u = 0$.

The **UAV SoS model** is defined as

$$SoS_{UAV} = \{UAV_1, UAV_2, UAV_3, G_{Box}, G_{Energy}, G_{DistToBox}\}, \quad (10)$$

where $UAV_1, UAV_2, UAV_3$ are $CS_{UAV}$. Each $CS$ represents a drone that detects, takes and converts boxes into energy. It aims at maintaining its energy level high enough to continue working; furthermore it is able to give boxes to other $CS_{UAV}$ if necessary. The goal of the $SoS_{UAV}$ (through $G_{Box}$, $G_{Energy}$ and $G_{DistToBox}$) is to keep all its $CS_{UAV}$ alive, which means that they must have roughly an identical criticality.

The **UAV SoS environment** is defined as

$$E_u = \{Boxes, Rule_{DetectBox}\} \quad (11)$$

where $Boxes$ is a set of boxes and $Rule_{DetectBox}$ is a rule explained hereafter.

An entity *box* is defined as

$$Box_i = \{T_b, R_b, Acq_b, L_b, F_b, G_b\} \quad (12)$$

with

- $T_b = Box$.
- $R_b = \{R_{Availability}\}$ represents the availability of the box for the $CS_{UAV}$.
- $Acq_b$, $L_b$, $F_b$, $G_b$ are empty sets.

The resource $R_{Availability}$ is used as a threshold to simulate the detection of a box by a $CS_{UAV}$ thanks to the rule $Rule_{DetectBox}$. If the condition part of this rule is checked, the $CS_{UAV}$ has detected this box and it saves its distance in its resource $R_{DistToBox}$.

*4.2.2. Cooperative Behaviour*

As we previously saw in the $CS_{UAV}$ model, the UAV SoS goal $G_r$ consists of three goals for a $CS_{UAV}$: $G_{Box} = \{R_{Box} > 0\}$ (to own as much boxes as possible), $G_{Energy} = \{R_{Energy} > 0\}$ (to have as much energy as possible) and $G_{DistToBox} = \{R_{DistToBox} == 0\}$ (to minimize the distance of the $CS_{UAV}$ to a box). In accordance with 3.2.1, the criticalities of the two first goals of a $CS_{UAV}$ at time $t$ are both defined by the formula 4 (d). The criticality of the

Table 3: Initialization of the three $CS_{UAV}$

| | $UAV1$ | $UAV2$ | $UAV3$ |
|---|---|---|---|
| $R_{Energy}$ | 60 | 60 | 60 |
| $R_{Box}$ | 0 | 0 | 0 |
| $R_{EC}$ | 0.08 | 0.015 | 0.015 |
| $R_{DistToBox}$ | 0 | 0 | 0 |
| $R_{DistToRG}$ | 0 | 0 | 0 |

third goal is defined by the formula 4 (a). Furthermore, a priority of 10 was given to $G_{Energy}$, a priority of 1 was given to $G_{Box}$ and a priority of 0.1 was given to $G_{DistToBox}$.

The criticality of a $CS_{UAV}$ can be then calculated with the weighted average of all its goals criticalities (formula (5)). According to SApHESIA, each $CS_{UAV}$ is cooperative and pursues the cooperative decision algorithm 1 (section 3.2.2).

*4.2.3. Experimentations*

The aim of this UAV SoS experimentation is to present the global functioning of an independent SoS using the cooperation approach we propose. UAVs can cooperate with each other by using the $F_{GiveBox}$ functionality. For the UAV SoS, three $CS_{UAV}$ have been instantiated: $UAV_1$, $UAV_2$ and $UAV_3$.

Each of the $CS_{UAV}$ has been initialized with the different values that are summed up in the table 3. The values have been chosen arbitrarily. $UAV_1$ is less effective than others because $R_{EC}$ (representing the energy consumption) is the highest. This lack of effectiveness enables to add some diversities in the SoS and to check the relevance of the cooperation mechanism carried out through the algorithm 1 within this SoS.

Obtained results can be seen in figure 3 and especially in the graphs 3 & 4. The level of energy of $UAV1$ (graph 4) decreases quicker than 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. Nevertheless, 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.

*4.3. Coupling two SoSs*

This subsection is devoted to the coupling of CoCaRo SoS and UAVs SoS. The aim is to study how it is possible to make two interdependent SoSs coop-

erate. To create dependencies in terms of functionalities, the two previous SoSs need to be slightly modified.

*4.3.1. Description*

The two SoSs to be coupled are close to those described in the previous subsections. We still have

- 3 robots groups ($CS_{RG}$) instantiated in $SoS_{RG} = \{Red, Green, Blue\}$
- 3 UAVs ($CS_{UAV}$) instantiated in $SoS_{UAV} = \{UAV_1, UAV_2, UAV_3\}$.

The $CS_{UAV}$ have been modified to create interdependencies (dependencies in both ways) between $SoS_{RG}$ and $SoS_{UAV}$. Thus, a $CS_{UAV}$ is still defined by formula given in equation 9. Only $R_u$ and $F_u$ are impacted in the following way (other elements of the set are unchanged):

- $R_u = \{R_{Energy}, R_{Box}, R_{EC}, R_{DistToBox}, R_{DistToRG}\}$
- $F_u = \{F_{CreateEnergy}, F_{GiveBox}, F_{MoveToBox}, F_{TakeBox}, F_{MoveToRG}, F_{Drop}\}$.

$R_{DistToRG}$ has been added to represent the distance of one $CS_{UAV}$ from a $CS_{RG}$. Thanks to the two new functionalities $F_{MoveToRG}$ and $F_{Drop}$, the $CS_{UAV}$ is now able to share boxes with a $CS_{RG}$ :

- $F_{MoveToRG}$ enables a $CS_{UAV}$ to go to a $CS_{RG}$ by decreasing the resource $R_{DistToRG}$. When a $CS_{RG}$ gets a box from a $CS_{UAV}$, it receives extra $R_{Energy}$.
- $F_{Drop}$ enables a $CS_{UAV}$ to give a box to a $CS_{RG}$ when $R_{DistToRG}$ is equal to 0. The use of $F_{Drop}$ (giving the possibility to drop a box for the $CS_{RG}$) also rewards the $CS_{UAV}$ by giving extra $R_{Energy}$.

The **environment of the CoCaRo and UAV SoSs** is composed of the union of the two previous environments given in equations 8 and 11:

$$E = \{Rule_{Apparition}, Rule_{GreenConsumption}, Rule_{BlueConsumption}, Rule_{RedConsumption}, Rule_{DetectBox}, Boxes\} \quad (13)$$

*4.3.2. Cooperative Behaviour*

Each CS tries to reach its objectives (i.e. to have more energy) by carrying boxes. The computation of the criticalities remains unchanged (see 4.1 and 4.2). Basically, the CS criticality varies according to the CS energy level (the lower is its energy, the higher is its criticality). As all the CSs have to be cooperative, each of them has to help the most critical of its neighbourhood. This cooperation can sometimes lead the CSs to exchange boxes. Thanks to the anticipated criticality (described in the algorithm 1), a CS can choose locally the most favourable action to perform from a cooperative point of view. The simultaneity of all the cooperative local behaviours of the different CSs, according to the AMAS approach, must provide the collective function that is "functionally adequate" in this environment.

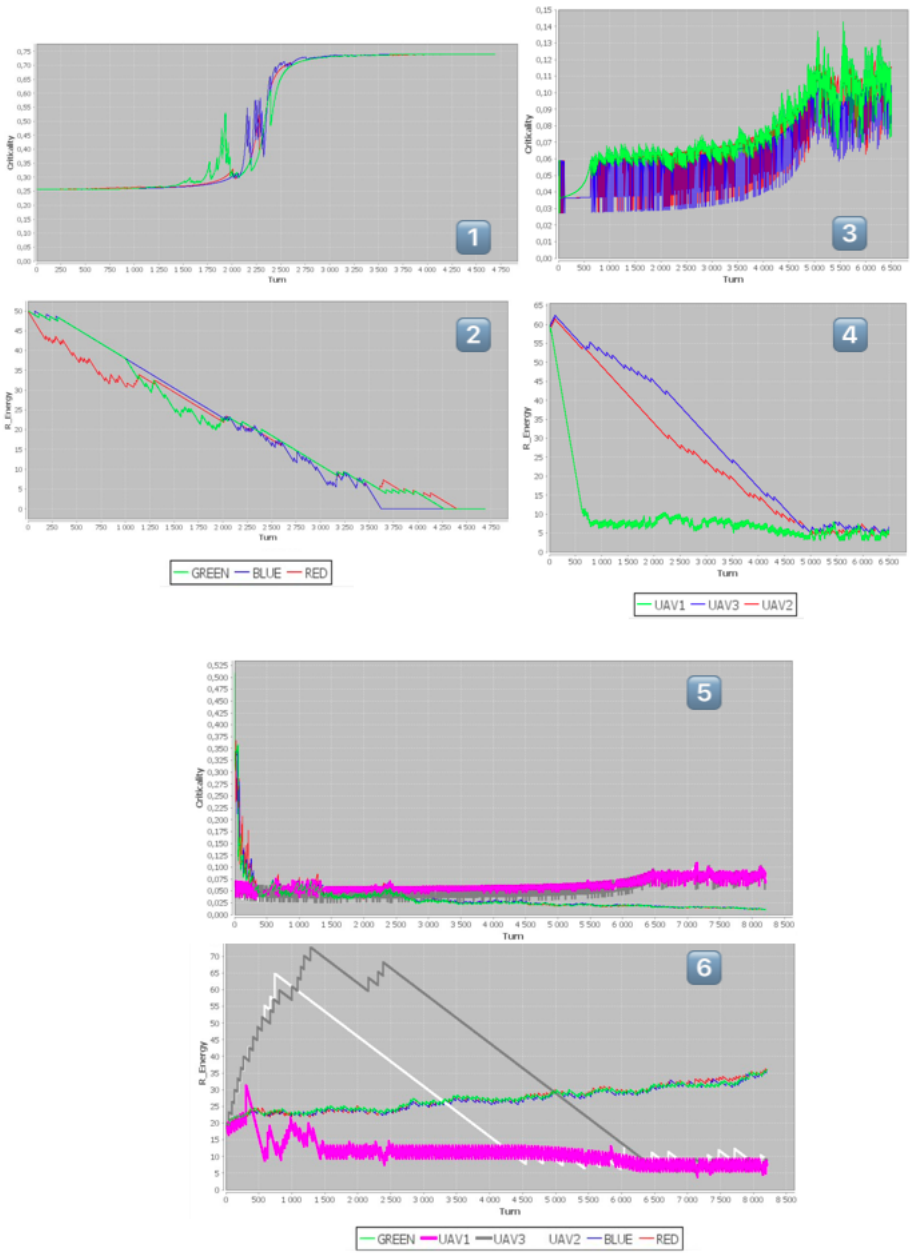


Figure 3: Results of two SoSs coupling with SApHESIA

*4.3.3. Results of the SoSs Coupling*

In order to evaluate the coupling of CoCaRo SoS and UAVs SoS, a simulation has been done. Initialization of the CSs are summarized in tables 2 and 3. Obtained results have been analyzed from the viewpoint of $R_{Energy}$ and the criticality (fig.3 (graphs 5 & 6)).

The first observed result is that no CS dies during the simulation of 8000 cycles. More specifically, the $R_{Energy}$ tends to be balanced between each different $CS_{RG}$ (fig. 3, graph 6) but this resource is slightly increasing after the cycle 1000. The criticality of the $CS_{RG}$ at the beginning of the simulation is high (criticality resource on bottom curve). This is due to the fact that, at the beginning of the simulation, no agent knows what to be cooperative means ; it needs to learn, as it goes along, what is its most cooperative action it can take for its neighbourhood (including itself). This learning begins to take effect around cycle 500. The $CS_{UAV}$ begin to drop (with the functionality $F_{Drop}$) extra boxes to the $CS_{RG}$ enabling to these latter to decrease their criticality. Thank to this cooperation and conversely to the graph 2 of fig. 3, the three $CS_{RG}$ are able to function during the 8000 cycles of the simulation.

Our experiments show that the CSs tend to find naturally the best way to solve the constraints of the environment. Indeed, the $CS_{UAV}$ bring back boxes to the $CS_{RG}$ and they increase their energy level because giving boxes to the $CS_{RG}$ rewards them with extra energy.

Furthermore, as $UAV_1$ and *Red* consume more energy than the other CSs (cf. tables 2, 3), they are the less efficient (they have a higher criticality than the others). Despite this, their respective level of energy is equivalent to that of the CS of their respective SoS because they are the most helped (by the others) thanks to the CSs cooperative behavior.

From a criticality point of view we can notice on fig. 3 that the criticality of the coupled SoSs (graph 5) is lower than the criticalities of each individual SoS (graphs 1 and 3). Furthermore this criticality stabilizes at low values. From an energy point of view we can notice on fig. 3 that the energy of the coupled SoSs (graph 6) is greater than the energies of each individual SoS (graphs 2 and 4) and it tends to increase for the CoCaRo SoS. So the two SoSs have naturally found their interdependency and the way to solve them.

However, we have to keep in mind that, if we want to solve problems involving different SoSs built with the AMAS approach, each being independently designed, issues may appear, especially when comparing the criticality of each SoS[2]. Indeed, the criticality of an SoS $A$ can have a different meaning than the criticality of an SoS $B$ as the value scale of the criticality in one SoS may be different of the second one. Indeed a criticality 0.7 does not necessarily have the same meaning in term of difficulty for $A$ and for $B$. For example, if $A$ criticality varies between 0 and 1 and the one of B between 0 and 100, a criticality of 0.7 does not have the same meaning in term of difficulty for the two. Indeed from $A$ point of view, it is very critical whereas from $B$ point of view, it is not.

[2]Every criticality used here has been manually normalized (between 0 and 1).

## 5. Conclusion and Perspectives

This paper presents a new SoS generic model enabling to model the Maier's criteria. 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 the active entities and rules that affect the interactions of the CSs. The AMAS approach can be an interesting paradigm to implement a new SoS architecting heuristic that respects the self-government principles. That is why we have proposed an SoS architecting heuristic implemented by an algorithm using cooperation as a social behaviour to enable CS self-organization. Cooperation is implemented through the concept of criticality that we have formalized to propose a reusable metric beyond the scope of the SoSs. As a bottom-up approach, our heuristic is decentralized (no central management entity needed) and enables to architect virtual and collaborative SoSs.

As we did not find any case studies with real and/or usable data to validate our work, we have evaluated our propositions through two experiments: one found in the literature and the other one made by ourselves. This experimentation relates to the capacity of our approach to reuse and combine existing SoSs that are independently designed but interdependent. The results show that the notions of cooperation and of criticality between two SoSs enable their combination in a natural way.

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 field. Concerning short-terms perspectives, testing our approach with more CSs will enable to validate that our heuristic can scale-up. To this aim, for a deeply evaluation, we could use the metrics given by [21]. A mid-terms perspectives could be to reify every CS of an SoS into an AMAS agent in order to transform an SoS into an AMAS. In this way, the CS 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. A long-term perspectives could be to collaborate with SoS experts to define more complex examples with real data and real case studies.

## References


[1] P. Acheson, L. Pape, C. Dagli, N. Kilicay-Ergin, J. Columbi, and K. Haris. Understanding system of systems development using an agent- based wave model. *Procedia Computer Science*, 12:21 – 30, 2012. Complex Adaptive Systems 2012.

[2] J. Axelsson. A systematic mapping of the research literature on system-of-systems engineering. In *10th System of Systems Engineering Conference (SoSE)*, pages 18–23, May 2015.

[3] C. Azani. *An Open Systems Approach to System of Systems Engineering*, pages 21–43. John Wiley and Sons, Inc., 2008.

[4] W. Baldwin and B. Sauser. Modeling the characteristics of system of systems. *2009 IEEE International Conference on System of Systems Engineering (SoSE)*, 2009.

[5] W. C. Baldwin, B. J. Sauser, and J. Boardman. Revisiting the "meaning of of" as a theory for collaborative system of systems. *IEEE Systems Journal*, 11(99):2215 – 2226, 2015.

[6] C. Ballegaard Nielsen, P. Gorm Larsen, J. Fitzgerald, J. Woodcock, and P. J. Systems of systems engineering: Basic concepts, model-based techniques, and research directions. *ACM Comput. Surv*, 48(2):41, 2015.

[7] M. Bjelkemyr, D. Semere, and B. Lindberg. An engineering systems perspective on system of systems methodology. In *2007 1st Annual IEEE Systems Conference*, pages 1–7, April 2007.

[8] BKCASE Editorial Board. Guide to the Systems Engineering Body of Knowledge. *Guide to the Systems Engineering Body of Knowledge (SEBoK)*, page 945, 2014.

[9] J. Boardman and B. Sauser. System of systems - the meaning of of. In *2006 IEEE/SMC International Conference on System of Systems Engineering*, April 2006.

[10] T. Bouziat. *A Cooperative Architecting Procedure for Systems Of Systems based on Self-adaptive Multi-Agent Systems.* Phd thesis, Université de Toulouse, Toulouse, France, November 2017.

[11] T. Bouziat, V. Camps, and S. Combettes. Cooperation in adaptive multiagent systems through system of systems modeling. In *2nd Global Conference on Artificial Intelligence (GCAI), Germany*, pages 214–226, 2016.

[12] T. Bouziat, V. Camps, and S. Combettes. A cooperative sos architecting approach based on adaptive multi-agent systems. In *6th IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems, SESoS@ICSE 2018, Gothenburg, Sweden, May 29, 2018*, pages 8–16, 2018.

[13] T. Bouziat, S. Combettes, V. Camps, and P. Glize. 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, October 2014.

[14] V. Camps. Application of a self-organizing method based on cooperation to information retrieval . In *Young researcher paper to European Conference on Artificial Intelligence (ECAI-98), Brighton, England, 23/08/98-28/08/98*, august 1998.

[15] D. Capera, J. George, M. Gleizes, and P. Glize. The AMAS theory for complex problem solving based on self-organizing cooperative agents. *Proceedings of WETICE*, pages 383–388, January 2003.

[16] M. DiMario, J. Boardman, and B. Sauser. System of Systems Collaborative Formation. *Systems Journal*, 3(3):360–368, 2009.

[17] L. Edward Pape II. *A domain independent method to assess system of system meta-architectures using domain specific fuzzy information.* PhD thesis, Missouri University of Science and Technology, 2016.

[18] T. Esteoule. *Adaptive Multi-Agent Systems for Wind Power Forecasting.* Phd thesis, Université Toulouse III- Paul Sabatier, France, December 2019.

[19] A. Gorod, B. Sauser, and J. Boardman. System-of-systems engineering management: A review of modern history and a path forward. *IEEE Systems Journal*, 2(4):484–499, December 2008.

[20] M. Henshaw, C. Siemieniuch, M. Sinclair, V. Barot, S. Henson, C. Ncube, S. Lim, H. Dogan, M. Jamshidi, and D. Delaurentis. The Systems of Systems Engineering Strategic Research Agenda Systems of Systems Engineering. 2013.

[21] N. Herbst, S. Becker, S. Kounev, H. Koziolek, M. Maggio, A. Milenkoski, and E. Smirni. *Metrics and Benchmarks for Self-aware Computing Systems.* Springer International Publishing, 2017.

[22] M. Jamshidi. System of systems engineering - new challenges for the 21st century. *IEEE Aerospace and Electronic Systems Magazine*, 23(5):4–19, May 2008.

[23] M. Jamshidi. *Systems of Systems Engineering: Principles and Applications.* CRC Press, 2008.

[24] M. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284, 1998.

[25] D. of Defense, O. of the Deputy Under Secretary of Defense for Acquisition, S. Technology, and S. Engineering. Systems engineering guide for systems of systems, version 1.0. Technical report, ODUSD(A&T)SSE, Washington, DC, USA, 2008.

[26] G. Picard. Agent Model Instantiation to Collective Robotics in ADELFE. In *Fifth International Workshop on ESAW'04, France*, pages 209–221. Springer Verlag, LNCA 3451, octobre 2004.

[27] S. Selberg and M. Austin. Toward an evolutionary system of systems architecture. *18th Annual International Symposium of the International Council on Systems Engineering, INCOSE 2008*, 4:2394–2407, 2008.

[28] R. Wang, S. Agarwal, and C. H. Dagli. 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, 2014.