A Cooperative SoS Architecting Approach Based On Adaptive Multi-Agent Systems

Teddy BOUZIAT
Valérie CAMPS
Stéphanie COMBETTES
teddybouziat@gmail.com
Valerie.Camps@irit.fr
Stephanie.Combettes@irit.fr
IRIT Lab, University of Toulouse - Paul Sabatier
Toulouse, France

ABSTRACT

This paper focuses on Systems of Systems (SoS) modeling and architecting. SoS architecting deals with the way that 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 an evolving environment.

In this context we defined a new model called SApHESIA (SoS Architecting HEuriStIc based on Agents) focusing on the environment and its dynamics. We also proposed a cooperative heuristic to control interactions exchanges between components. These contributions are then instantiated to a case study and evaluated through two scenarii. Obtained results are finally discussed and some perspectives are given.

1 INTRODUCTION

Since the World War II, researchers tend to develop methodologies and tools to build and control the development of more and more complex systems and projects.

This is the apparition of the System of Systems (SoS) concept, that spreads in civil domain such as crisis management or logistic systems. 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 System Engineering 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 paper deals with two SoS research areas: SoS modeling and SoS architecting. To achieve the first point, we propose a new model called SApHESIA (SoS Architecting HEuriStIc based on Agents) taking into account the environment in which the SOS is involved. This is one of the originalities of SApHESIA. We have used set theory and agent paradigm to define this model that takes into account the characteristics of SoS. Secondly, we present a new SoS architecting proposition 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 system and its goals. In this proposition, the SoS architecture evolves during 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. We also propose a management and visualization tool to observe and act on the SoS during its functioning both at the overall level and at the component systems level. The actions can concern links between component systems (which can be dynamically added or deleted), parameters, etc. Finally we instantiate this model on two scenarii (military domain) in order to validate the efficiency, the functionally adequacy and the robustness of our SoS architecting proposition.

The paper has the following structure: at first main properties and characteristics of SoS are briefly discussed, and we propose our definition of SoS. After this, the SApHESIA principles are described, followed by experimentations and results analysis.

2 SYSTEM OF SYSTEMS

Concepts of SoS and SoSE (SoS Engineering) appeared when System Engineering (SE) focused on how to design a group of more or less independent complex systems working together to fulfill a higher goal.

Authors have given a lot of definitions from various areas ([15], [20]) but unfortunately they propose neither a consensual definition of SoS nor common characteristics of SoS. Research efforts have then been put on SoS taxonomy and theoretical foundations during these last ten years in order to find a generic definition [18], [14], [6].

DOI: 10.1145/3194754.3194756

2.1 Main Properties and Characteristics of SoS

Maier in [18] was the first to characterize SoS; he gave five main properties accepted by researchers working on SoS ([14],[21]), that distinguishes a SoS from a traditional complex system. The **operational independence** means that a component system removed from the SoS continues to operate independently. The **managerial independence** means that a component system takes its own decisions concerning what it has to do. The **geographical distribution** means that in a SoS, exchanges between component systems only concern information diffusion. An **emergent behavior** at the SoS level is the production of an overall behavior that was not implemented (and so not predicted) in its component systems. Finally, an **evolutionary development** is a dynamic development that takes care and integrates the changes (structure, use etc.) that occur within the SoS at runtime.

Some convergences between the different characterizations of Gorod [13], Sauser [6] [4] and Bjelkemyr [5] exist. Thus they propose the ABCDE characteristics that distinguish classic systems from SoS. **Autonomy** is derived from the *operational and managerial independence* and refers to the independence of the component systems from each other and from the SoS itself. Conversely to classical systems, component systems of SoS are autonomous. **Belonging** is a balance between *autonomy* and the loss of a part of the independence for the benefit of the SoS goals. It is also the ability to give/accept assistance to/from others component systems to contribute to the goal(s) of component systems. Conversely to classical systems, in SoS, component systems have the choice to belong or not to a SoS by negotiating and evaluating their interests in order to fulfill their own purposes. **Connectivity** is the capability to connect component systems as they need. In SoS, component systems manage their connectivity that evolves during time. **Diversity** refers to the notion of variety in a system, which is a direct reference to the Law of Requisite Variety of Ashby [2]. The latter states that to maintain its stability, a system must have a level of variety at least equals to the variety of the environment (the outside of the system) it evolves in. This variety enables to cope with the changes that occur in the environment. In SoS, this characteristic is essential because of the environment dynamics, and it can be increased for example, by adding new component systems into the SoS. **Emergence** means the apparition of phenomena produced by a system, this apparition not only being the result of the simple sum of its parts. [16] considers the emergence as being *the movement from low-level rules to higher-level sophistication.* In SoS, the idea is to create conditions where both foreseen and unforeseen behaviors appear to cope with the environment dynamics.

2.2 Our Proposition of SOS Definition

We propose, here, the following **definition of SoS** taking into account the addressed characteristics : *a SoS is composed of interacting systems, called component systems, that are autonomous and may evolve in a dynamic environment. A SoS may 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 operational and managerial independences. They may be geographically distributed and their dynamic interactions can give rise to an emergent behavior at the SoS level. They adapt themselves to the evolution of the SoS during time and to the constraints of their environment. SoS are open (component systems can join or leave the SoS during functioning). We propose **nine criteria to evaluate existing generic SoS models** and to show their limitations. The five first ones correspond to the Maier's criteria (heterogeneity, managerial independence, geographical distribution, operational independence, interactions between component systems) while the four other ones concern the SoS and the environment models: **modeling of SoS** (ability to model one SoS as an autonomous entity with its own goals), **dynamic environment modeling** (ability to model the dynamic environment in which the SoS evolves), **global expressiveness** (ability to express problem in order to model the SoS), and **metric definition** (ability to define useful metrics, e.g. the global cost or the performance of the SoS).

We only found two SoS models in the literature: (*i*) a model based on set theory [3] and (*ii*) a based-wave model [1]. Both have a lack concerning their realism and fail concerning our evaluation criteria. The model based on set theory [3] seems hard to use for studying real cases of SoS because of its lack of expressiveness. Furthermore, the interactions between component systems in both models are not used, and the environment is not addressed or not dynamic. However as [12] explains, the autonomous adaptation of a system is always relative to the environment in which it is immersed. The adaptation process "is not in the system, nor in its environment, but between them"; a system and its environment influence each other through a coupling, very close to the notion of *structural coupling* defined by [19]. In other words, the system acts in the environment to achieve what it was designed for. The environment being open, there is no guarantee that :

- the actions undertaken by the system have the desired effects,
- nor that an entity in the environment has acted in contradiction with the purpose of the system during the decision-making process,
- nor that the state of the environment has changed since its last perception.

Then the "response" of the environment to the action of the system constitutes a pressure, a constraint that the system has to take into account in order to adjust, to learn the proper behavior that guarantees the expected overall functionality of the system.

This analysis (a deeper one can be found in [7]) leads us to propose a new generic SoS model to fill these lacks.

3 OVERVIEW OF SAPHESIA PRINCIPLES

SApHESIA (SoS Architecting HEuriStIc based on Agents) enables to model a SoS by focusing on its environment as well as on the interactions between its Component Systems (CS) while using the multi-agent system paradigm. It consists of the SoS (made of CS and goals) as well as its environment (made of entities and rules).

3.1 Component System Model

A **component system** CS is the smallest part of a SoS and represents an element of the second S of SoS. It is defined as $CS = \{T, R, Aq, L, F, G, Cost\}$ where

- T is the type of CS within the SoS,

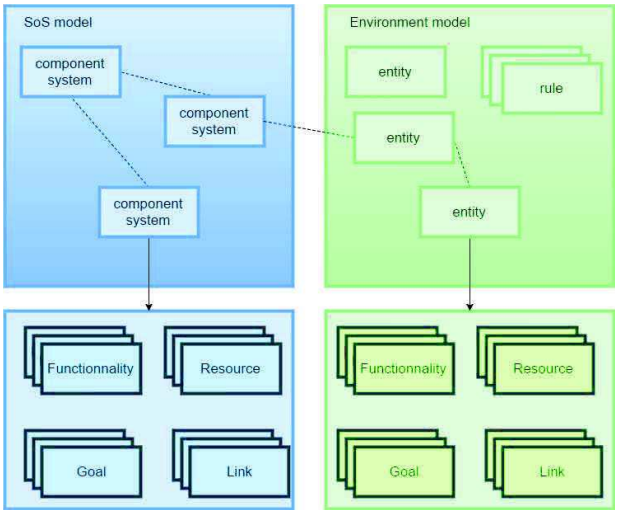

Figure 1: Components Overview of SApHESIA Model

- $R = \{R_1, ..., R_n\}$ is the set of its resources,
- $Aq = \{Aq_1, ..., Aq_q\}$ is the set of acquaintances of CS,
- $L = \{L_1, ..., L_q\}$ is the set of links of CS with others component systems,
- $F = \{F_1, ..., F_m\}$ is the set of its functionalities,
- $G = \{G_1, ..., G_p\}$ is the set of its goals, and
- $Cost \in R$ is the cost of CS.

More precisely, the **type** is a string that represents a kind of component systems in the SoS.

A **resource** R_i is a structure $R_i = \{type : String, quantity : Float\}$ which represents the passive elements (i.e. which has no effector) in the SoS.

An **acquaintance** is an oriented association between two component systems ($aq_{CS} = \{CS'\}$) 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 **functionality** $\mathcal{F}$ is an effector that can affect (*i*) its own resources and/or the ones of other CS of the SoS as well as (*ii*) the links between CS. $\mathcal{F} = \{f, t, p\}$ where t is the execution time of $\mathcal{F}$, $p \in [0, 1]$ is the performance (i.e. the probability of success) of $\mathcal{F}$ and $f = Conditions \rightarrow Effects$ is a function of $\mathcal{F}$ where $Conditions$ and $Effects$ are sets that represent respectively the conditions for f to be executed (e.g. a certain quantity of resources or the existence of a link between two CS or the existence of a CS) and the effects applied on the SoS or the environment once f is executed (e.g. a certain quantity of resources or the existence of a link between two CS). In the SApHESIA model, the functionalities need to be defined by the SoS designer, so they are considered as system inputs. Thus, different configurations of the SoS can be tested by the designer and they can help him to do tuning in order to decision-making values.

A **goal** is the special state that CS tries to reach with a given priority. This state can be *i*) to own a certain quantity of a type of resource or *ii*) the existence of a link between one CS and another one.

A **cost** is associated with each CS, representing its charge and the charge for the SoS when it uses it. This charge depends on the problem.

3.2 SoS Model

We propose to model a SoS as $SoS = \{\mathcal{S}, \mathcal{G}\}$ where

- $\mathcal{S}$ is the set of component systems and
- $\mathcal{G}$ is the set of goals, which may be a subset of the goals of the component systems of $\mathcal{S}$ according to the type of the degree of central management of the SoS.

3.3 Environment Model

An environment is defined as $\mathcal{E} = \{Entities, Rules\}$ where

- $Entities$ is a set of **entities** that represent active independent objects which are able to affect the environment or the SoS itself and
- $Rules$ is a set of **rules** that model exterior constraints applying to the SoS.

An **entity** is an active independent object that is not a part of the SoS. It is defined as a set of sets : $E_i = \{T, R, Aq, L, F, G\}$ where

- T is the type of $Entities$,
- $R = \{R_1, ..., R_n\}$ is a set of **resources**,
- $Aq = \{Aq_1, ..., Aq_q\}$ is a set of **acquaintance**,
- $L = \{L_1, ..., L_q\}$ is a set of **links**,
- $F = \{F_1, ..., F_m\}$ is a set of **functionalities**, and
- $G = \{G_1, ..., G_p\}$ is a set of **goals**.

These six elements are similar to those of the CS presented in 3.1. Nevertheless 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 all the entities in the environment and/or all CS in the SoS that fulfill the $Conditions$.

Figure 1 sums up the relationships between the elements previously described that compose SApHESIA model.

3.4 Criticality as Heuristic for Architecting SoS

Criticality is a concept used in the AMAS (Adaptive Multi-Agent Systems) approach in order to make the agents cooperate and the system self-organize. The AMAS approach [9] enables to solve problems for which an *a priori* solution does not exist. More concretely, this approach allows to design complex systems whose the overall function is not implemented in the parts (agents) of the system. It focuses on the design of multi-agent system (called Adaptive Multi-Agent Systems) that uses self-organization to make the collective function emerge, and to make the agents adapt themselves to the environment changes. In others words, the behaviors of each agent will lead to change the organization (or architecture) of the multi-agent system and so to produce a new collective function.

To give rise to this new overall function, agents uses the concept of cooperation between each other and their environment. The cooperation of an agent is the social attitude that makes an agent help other agents (itself included) to fulfill its goals. Thus an agent has to choose the action that is the most helpful for the others and for it. The best cooperative action is chosen according to the current difficulty of agents through a metric called the "criticality".

The AMAS approach instantiates the notion of an agent's criticality to the problem to be solved. Criticality is the distance (possibly temporal, spatial or even logic distance) between the current situation of the agent and its local goal; the more the agent is far from its goal, the more it considers that its current situation is critical [17]. It then defines the cooperative local treatments it must apply to reduce the criticality of the most critical agent (possibly itself), avoiding that this process leads another agent to become more critical.

We presented in [8] a generic and computable definition of criticality as a function taking as inputs, a subset of perceptions and its goals concerning these perceptions. The criticality C of an agent i at time t is

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

where $P(t) = \{p_1(t), ..., p_n(t)\}$ is the entire set of perceptions of the agent i and S_{goal} is a subset of P. To know the result of one of its actions on its own criticality an agent needs to know the anticipated criticality of this action. Formally, the anticipated criticality of an agent i for an action a is defined as

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

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 calculated for a sequence of actions $A = \{a_1, a_2, ..., a_n\}$ as

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

3.5 Instantiation of Criticality in SApHESIA

As we want to use the concept of criticality in order to propose a generic cooperative decision algorithm for architecting SoS, we propose a generic formulation of criticality for a CS based on the SApHESIA model.

We first adapt the definition of this concept to SApHESIA according to the resources and goals of a CS. Thus, the *current state* of a CS is represented by the perceptions of its current resources and *the state it tries to reach* is represented by its goals. We define a criticality for each goal g in the set of goals G of the component system CS. This criticality has to represent the distance of fulfillment of g. The criticality $C_g(t)$ of the goal g at time t can be defined by:

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

(1) the goal g has to be equal to a given quantity Qu of Re (resource),
(2) the goal g has to be different from a given quantity Qu of Re,
(3) the goal g has to be smaller or equal to a given quantity Qu of Re,
(4) the goal g has to be greater or equal to a given quantity Qu of Re,
where $\Delta_g(t)$ is the difference between the given quantity Qu and the current amount of resource Re of CS and α is a coefficient influencing the shape of the curve. For example, if the goal g concerning a resource Re has to be greater (resp. smaller) than a given threshold quantity Qu, then the criticality of g has to be *high* if Re is smaller (resp. greater) than Qu, and *small* if the Re is greater (resp. smaller) than Qu. These sigmoid functions have been chosen as their shapes correspond to the meaning of each goal. Nevertheless one issue is that the criticality of different CS of a SoS can be calculated in various ways [17]. For example a criticality may belong to $I_1 = [\beta, \gamma]$ whereas another one to $I_2 = [\beta', \gamma']$. To solve this point a normalization of criticalities has to be done from I_1 to I_2 and vice versa; a proposition is given in [7].

3.6 Cooperative Decision Algorithm for Component Systems

A component system is modeled as an autonomous entity (agent) having a $Perceive - Decide - Act$ cycle. The proposed decision algorithm 1 is implemented during its *Decision* phase in order to choose the most cooperative action (here a functionality).

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

Algorithm 1: Cooperative Decision Algorithm of CS_i

Basically each component system $CS_i = \{T_i, R_i, Aq_i, L_i, F_i, G_i, Cost_i\}$ has to construct the cooperative table and to choose the most cooperative functionality.

The Cooperative Table Construction: This table contains the anticipated criticalities lists of CS_i as well as its neighborhood for all the functionalities it can apply. To construct it, CS_i asks for the anticipated criticality of its neighborhood (using Aq_i and L_i) for each functionality $f \in F_i$ with the function $askAnticipatedCrit(CS_j, f)$ (line 4 of algorithm 1) with CS_j and a functionality f as inputs. This function returns the value of the anticipated criticality of CS_j if CS_i applies f on CS_j. This anticipated criticality is saved in a new line (identified by a couple (CS_j, f)) of the cooperative table $CoopTable$. The anticipated criticalities of other component systems (including CS_i) are added to this line through the function

$askAnticipatedCrit(CS_j, CS_k, f)$ that returns the anticipated criticality of CS_k if CS_i applies f on CS_j (line 6). Finally, all the lines are sorted from the highest criticality to the lowest one (line 8).

The Most Cooperative Functionality Choice : Once $CoopTable$ built, CS_i chooses the most cooperative functionality in terms of criticality ($minmaxFunc$) to minimize the maximum of neighborhood criticality. Indeed, from an AMAS point of view, each CS tries to "help" its neighborhood by choosing the functionality that, in the worst case, causes the minimum raise of criticality. This cooperative behavior applied by all the CSs can be seen as a totally decentralized and cooperative heuristic only based on a local comparison of anticipated criticalities.

4 PROPOSED TOOLS, EXPERIMENTATIONS AND RESULTS

After a brief description of the managing tool implemented in order to architect SoS with SApHESIA, this section presents an instantiation of SApHESIA model with the Missouri Toy Problem [11], the scenarii and metrics and then discusses the obtained results.

4.1 Overview of SApHESIA Managing Tool

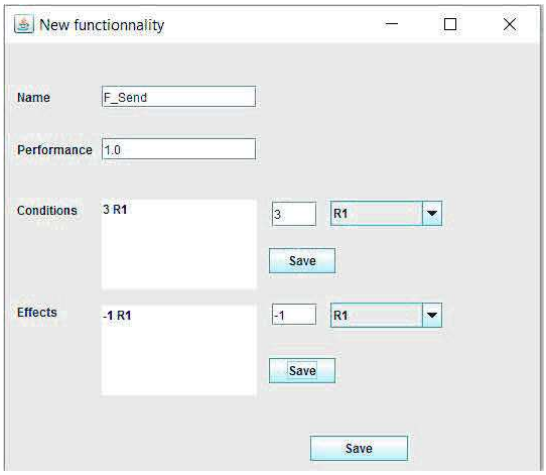

Figure 2: A Functionality Creation

The SApHESIA managing tool enables to study SoS architecting by simulating interactions between CSs in a dynamic environment. It is made of two components:

- the **SoS and Environment Generator** that generates the simulation from the inputs describing the SOS and its environment. It enables (*i*) to add or remove CSs and entities during the simulation, (*ii*) to change all the different properties of a CS (resources, links, goals, etc.). Thus the effects of these changes (on the SoS or on the environment) may be seen at runtime, without re-launching the simulation to update these effects. The use of SApHESIA Modeling Language (SML) based on XML files containing the description of a predefined SoS and environment in a SML Parser & Compiler, enables to facilitate the GUI generator when a large SoS is created from scratch. Figure 3 shows the CSs creation and figure 2 shows an example of functionality creation.
- the **Core Simulator**, composed of an *engine* and a *viewer*, which enables to run the simulation scenario defined by the *SoS and Environment Generator*. For that, the *engine* uses the components (SoS and environment) loaded with the SML Parser & Compiler; The *viewer* automatically shows in a window the data (such as resources, criticality etc.) chosen to be displayed while the simulation is running. The state of CSs and the links between them as well as entities can then be visualized.

4.2 Instantiation of SApHESIA to the Missouri Toy Problem

The Missouri Toy Problem is a SoS architecting scenario initially presented by [10] and then extended by [11]. Its goal is to *relay commands and ISR data from a ground station to a carrier battle group via Unmanned Aerial Vehicles (UAVs) and satellites A or B.* The ground station cannot communicate directly with the carrier battle group. Modeled as a SoS, the types of CSs and their functionality are (*i*) the ground station (*Ground* type in SApHESIA) whose functionality is to *send a signal* to satellites and UAVs; (*ii*) the UAV (UAV type) whose functionality is to *send a signal*; (*iii*) the satellite A (Sat_A type) whose functionality is to *send a signal* to UAVs and other satellites A (not to satellites B); (*iv*) the satellite B (Sat_B type) whose functionality is to *send a signal* to UAVs and other satellites B, (not to satellite A) and (*v*) the carrier battle group (*Carrier* type) whose functionality is to *receive a signal* from a component system but not from the ground station.
Constraints : *Ground* cannot exchange with *Carrier* and Sat_A cannot exchange with Sat_B and vice versa.
Resources are the signals (received and sent). The functionality F_{Send} represents the sending of a signal; it consumes one *Signal* of the sender and generates one to the receiver. Perturbations in the environment can be events such as cyber-attacks, weather issues, jamming of communication links or the unavailability of CSs.

4.3 Evaluation Criteria and Associated Metrics

This section defines the three evaluation criteria and the associated metrics we propose for the evaluation.

Functional Adequation: a decentralized architecting heuristic leads to a *SoS* that is functionally adequate i.e. it achieves the expected overall function (from the viewpoint of an external observer knowing the SoS purpose). The metric we propose for functional adequacy is the Total Transmitted Signals $TTS = \frac{GS}{RS}$ where GS is the number of *Signal* resources generated by the *Ground* and RS is the number of *Signal* effectively received by the *Carrier*.

Efficiency: SApHESIA model enables to define a performance of a functionality used by a CS. The efficiency of the *SoS* is related to the use of the most efficient CSs, i.e. the ones having the best performance. This efficiency is evaluated through a metric called *Cost* representing the *global cost of the SoS* and which is inversely proportional to the efficiency of the *SoS*. Indeed the more efficient

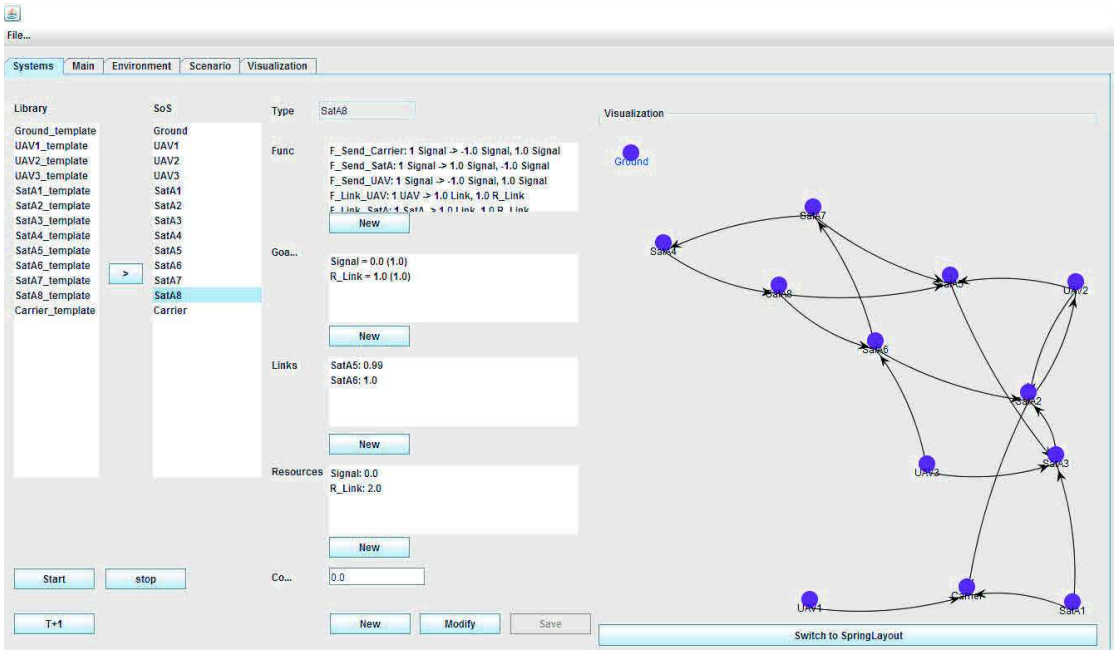

Figure 3: Managing Tool of SApHESIA

the SoS is, the lower the cost is. To compute $Cost$, the F_{Send} functionality is changed by adding a new resource called $R_{CostSend}$ that saves how many times the CS sends a signal. The cost metric is $Cost = \Sigma_{S_i \in \mathcal{S}}(S_i.Cost + S_i.R(R_{CostSend}))$.
Finally, the minimum cost, $MinCost$, for transmitting all the signals between $Ground$ and $Carrier$ for a given scenario represents the most efficient architecture. Thus, $MinCost$ will be calculated for the scenario and compared with the current cost of the architecture. The functionality F_{Send} of each CS increments the resource $R_{CostSend}$. Formally, $CostRatio = \frac{Cost}{MinCost}$.

Robustness: we defined it as *the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions.* To evaluate robustness of our SoS architecting heuristic, some events representing failures of CSs occur during the scenario. A failure disables the functionalities of a CS by preventing it to receive/send signals or link with other CSs. To cope with these failures, the *SoS* has to adapt by finding new CSs in order to achieve the SoS goal (to transmit the signal). Finally, the evolution of TTS evaluates the robustness as it represents the failure a CS that will prevent the SoS to be functionally adequate. The transmission of signals to the $Carrier$ is then momently interrupted and TTS does not evolve anymore. To summarize, the TTS metric evaluates the functional adequacy, the $CostRatio$ evaluates the efficiency and the TTS evolution evaluates the robustness.

4.4 Scenarii Description

This section describes the scenarii by giving the available CSs for the *SoS*, the initial values of the different CSs resources and the description of the scenario through the events that will be triggered.

Available Component Systems and Initialization Values: In these scenarii, 21 CSs of different types compose the SApHESIA model : 1 $Ground$ (g), 5 UAV ($u_i, i \in [\,1, 5\,]$), 8 Sat_A ($a_i, i \in [\,1, 8\,]$), 6 $SatB$ ($b_i, i \in [\,1, 6\,]$), and 1 $Carrier$ (c).

Each type of CSs is agentified according to the AMAS approach (i.e. all the agents must be cooperative) and initialized with the parameters given in table 1. The SApHESIA engine is initialized with a *reinforcement link* and a *destruction link* both equal to 0.1 (these values have been experimentally determined). The reinforcement link represents the strength of the link between two connected CSs. Until the 2000 cycle the $Rule_{Signal}$ rule generates $Signal$ with a rate of 0.09, so the total number of signals is equal to 180 (0.09×2000).

Minimum Cost Calculation. The table 2 gives the performances of the F_{Send} functionality for each CS. For example, the performance of the F_{Send} of UAV u_1 to Satellite a_1 is 0.5. This table enables to calculate the most efficient path i.e. the one composed of CSs with the highest performances that is: $Ground \rightarrow u_3 \rightarrow a_3 \rightarrow Carrier$. The value of $R_{CostSend}$ for each CS is given in table 1 enabling to compute $MinCost = 180 \times \Sigma_{S_i \in Opti}(S_i.Cost + S_i.R(R_{CostSend}))$ with $Opti = \{Ground, u_3, a_3\}$. In our example, $MinCost = 1080 = 180 \times (Ground.Cost + u_3.Cost + a_3.Cost + Ground.R(R_{CostSend}) + u_3.R(R_{CostSend}) + a_3.R(R_{CostSend}))$.

Scenario 1: Functional Adequacy, Efficiency and Robustness Testing. In this scenario, all CSs are running from cycle 0 to cycle 2000. At cycle 2000, CS a_3 breaks down so that CSs are no longer able to link or send signal to a_3. We added this event to test the robustness of the *SoS* i.e. that the CSs may find a new path to send signals to $Carrier$. The SoS should then find the path

Table 1: Resources Initialization and Cost of each Component System

	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

	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 2: Performance p_{Send} of Functionality F_{Send}

	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																					

$Ground \rightarrow u_1 \rightarrow a_1 \rightarrow Carrier$ (given as being now the best path in the table 2). Simultaneously, from the cycle 2000, the signal generation is interrupted, in order to compare the number of generated signals and the one of received signals (through the metric TTS) at the cycle 7000 (the simulation end).

Scenario 2: Deeper Robustness Testing. This scenario aims at deeply testing the SoS robustness through the generation of several failure events. A CS failure occurs every 100 cycles in order to study how the SoS adapts. As the performance of CSs on F_{Send} is different, the order of failures influences the SoS functioning. Indeed, if all the CSs having a good performance fail first, the SoS will have more difficulty to overcome the failures. We defined 3 sequences where the position of the failures on the most efficient CSs ($OptCS = a_1, a_3, u_1, u_3$) are different :

$s_{easy} = \{u_2, u_4, u_5, a_2, a_4, a_5, a_6, a_7, a_8, b_1, b_2, b_3, b_4, b_5, b_6, \underline{OptCS}\}$

$s_{medium} = \{u_2, u_4, u_5, a_2, a_4, a_5, a_6, \underline{OptCS}, a_7, a_8, b_1, b_2, b_3, \underline{b_4}, b_5, b_6\}$

$s_{hard} = \{\underline{OptCS}, u_2, u_4, u_5, a_2, a_4, a_5, \underline{a_6}, a_7, a_8, b_1, b_2, b_3, b_4, b_5, b_6\}$

Ground and *Carrier* are not concerned; as they are respectively the source and the destination of signals, their failures immediately stop the SoS functioning. The TTS metric highlights the inability of the SoS to send a *Signal* from *Ground* to *Carrier*.

4.5 Results Discussion

Figures 5 show the simulation results of Scenario 1. The curves *CostRatio*, *Signal*, *Criticality* and *TTS* respectively show one aspect of the evolution of the different CSs. The curve *F_Send_UAV1&3* shows how many times on the last 500 cycles the *Ground* has used F_{Send} with u_1 and u_3. The curve *F_Send* shows how many times this functionality has been used and with which CS.

At the beginning of the simulation, the number of transmitted signals TTS increases, meaning that the *SoS* finds its functional adequacy. Before cycle 2000, Figure F_{Send} shows that *Ground* uses u_3 more than u_1 to send signal; the most efficient path is used. After cycle 2000, TTS is always increasing, showing that the *SoS* is robust as it is still running even if a failure (of a_3) has occurred. TTS is highly increasing after turn 2000 because the generation of signals is over. *CostRatio* increases even more because the failure of a_3 leads u_3 and u_1 to find alternative paths being less efficient than $Ground \rightarrow u_1 \rightarrow a_1 \rightarrow Carrier$ which is the current most efficient path. Curve F_{Send} shows that finally u_1 is preferred to

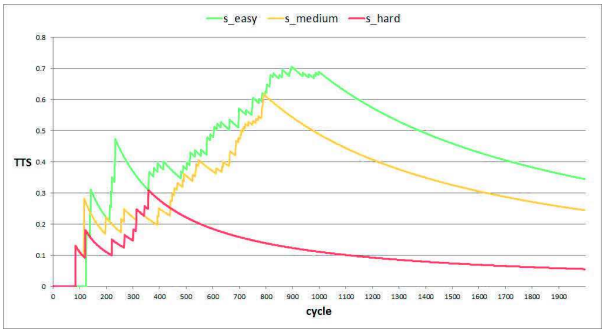

Figure 4: Scenario 2: Evolution of TTS for the 3 Sequences

u_3 because of this new most efficient path. The F_Send_UAV1&3 curve confirms that the most used CSs are *Ground*, u_1, u_3, a_1 and a_3. $CostRatio < 1.4$ means that even with failures, the SoS finds efficient alternative paths. This first scenario shows that the *SoS*, through the cooperative behavior of its CS, can find architectures that are compliant with our evaluation metrics.

Figure 4 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. First, these results show that the order of failures has an important impact on the global functioning of the SoS as the functionalities of the CS have different performances.

The TTS evolution shows that the SoS is no more able to send *Signal* at cycle 1000 for s_{easy}, at cycle 800 for s_{medium} and at cycle 400 for s_{hard}, a failure appearing every 100 cycles. Thus, 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} on a total of 19 CS. This scenario illustrates the robustness of the SoS driven by our cooperative heuristic : even with several CS failures, the SoS (thanks to the ability of its CS to find new neighbors, to self-organize), is able to find alternative architectures to continue the signals sending. More details and results can be found in [7].

5 CONCLUSION AND PERSPECTIVES

This paper deals with the modeling and architecting of SoS. We have presented SApHESIA (SoS Architecting HEuriStIc based on Agents) formalism, which takes into account the characteristics of SoS, and especially the SoS environment, its dynamics and the self-organization of its CSs. We also proposed a new SoS architecting process based on the Adaptive Multi-Agent System (AMAS) approach that advocates full cooperation between all the CSs of the SoS through the concept of criticality. This criticality is a metric representing the difficulty of an agent (CS) to reach its goal. In this proposition, the SoS architecture evolves during time to self-adapt to its environment dynamics, while taking into account the respective local goals of its components. We have presented the tools used to manage and to visualize data of the studied SoS model. Finally this model has been instantiated with two scenarii in order to evaluate the efficiency, the functional adequacy and the robustness of the proposed SoS architecting.

One perspective is to more investigate on the evaluation of SApHESIA, especially by trying to compare it with other approaches (until now, we did not find any case studies enabling such comparisons) and another perspective is to investigate on criticality in order to coupling interdependent AMAS (having different criticality scales) which have been independently designed.

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] W. R. Ashby. *An introduction to cybernetics*. New York,J. Wiley, 1956.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] D. Capera, J. George, M. Gleizes, and P. Glize. 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, 2003.

[10] D. A. DeLaurentis, K. Marais, N. Davendralingam, S. Yeob Han, P. Uday, Z. Fang, and C. Guariniello. Assessing the Impact of Development Disruptions and Dependencies in Analysis of Alternatives of System-of-Systems. TechReport SERC-2012-TR-035, Purdue Uninersity, december 2012.

[11] 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.

[12] P. Glize. *L'adaptation des systèmes à fonctionnalité émergente par auto-organisation coopérative*. HDR, 2001.

[13] 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.

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

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

[16] S. Johnson. *Emergence: the connected lives of ants, brains, cities, and software*. Scribner, 2001.

[17] S. Lemouzy. *Systèmes interactifs auto-adaptatifs par systèmes multi-agents auto-organisateurs : application à la personnalisation de l'accès à l'information*. PhD thesis, December 2011.

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

[19] H. Maturana and F. Varela. *Autopoiesis and cognition: The realization of the living*. Springer, 1980.

[20] D. of Defense. System of systems - systems engineering guide: Considerations for systems engineering in a system of systems environment. Techreport, Department of Defense, 2006.

[21] 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.

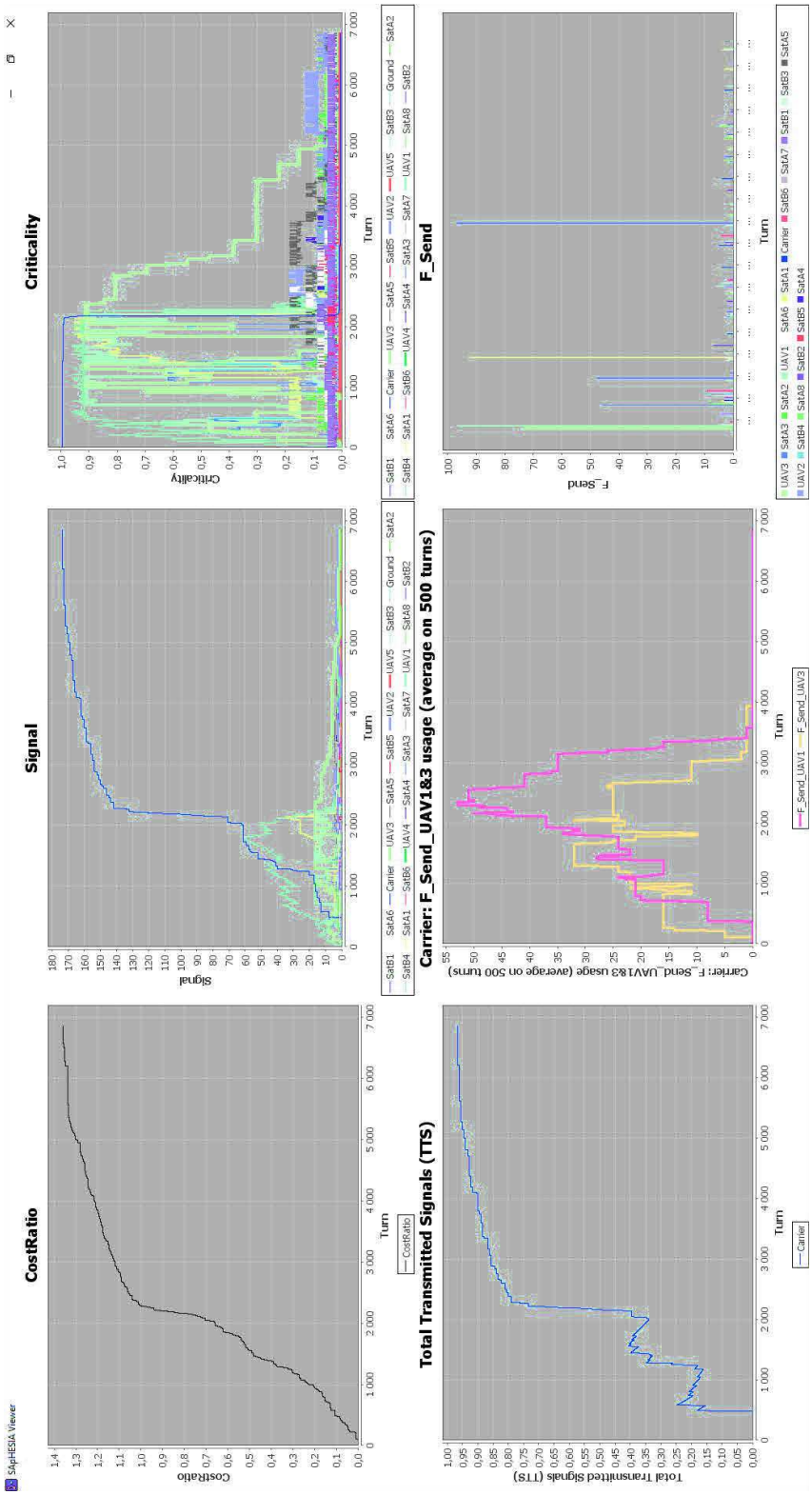

Figure 5: Scenario 1: Simulation Results