# Modeling the Dynamic Reconfiguration in Smart Crisis Response Systems

Akram Seghiri[1,2][a], Faiza Belala[1] and Nabil Hameurlain[2][b]
[1]LIRE Laboratory, University of Constantine 2 - Abdelhamid Mehri, Constantine, Algeria
[2]LIUPPA Laboratory, University of Pau, Pau, France

Keywords: Smart Systems-of-Systems, Architecture Description Language, Dynamic Reconfiguration, Maude Strategy Language.

Abstract: Crises response systems form the majority of today's complex systems; they generate situations characterized by harmful consequence, low probability, and short decision time. The decision-making in these systems constitute a challenging issue and must be conscientiously supported. The aim of this paper is twofold, we derive a referential architecture for crisis response system-of-systems in one hand, the role of communication and the characteristics of constituent systems that are needed to adapt their dynamic behavior are added to the architecture standard of IEEE. In the other hand, we define a formal model on the basis of the Maude Strategy language, to deal with the dynamic reconfiguration of these system types.

## 1 INTRODUCTION

### 1.1 Context and Problematic

Crises response systems form the majority of today's complex systems; they generate situations characterized by harmful consequence, low probability, and short decision time. The crisis response in general constitutes a challenging issue and must be conscientiously supported. It involves multiple parties, each with their own autonomy and capabilities, leading to differentiations in structure, goals, strategies, and constraints for cooperation (Van Veelen et al., 2008). Therefore, a crisis response system needs to support distributed and continuous adaptation due to unpredictable changes in goals and plans for the current crisis management organization. Different approaches to adaptation and (re-)configuration have to be integrated for dynamically structuring, configuring, and re-configuring organizations to comply with actual goals and agreements on plans in crisis management conditions.

On the other hand, Systems of Systems (SoS), are a set of independently developed systems that interact to achieve a specific goal, known as an SoS mission, which neither of them can solely achieve. They are distinguished from large monolithic systems by a set of eight key 'dimensions' of SoS (Nielsen et al., 2015): autonomy, independence, distribution, evolution, emergent behavior, dynamic behavior, interdependence and interoperability. Besides these dimensions, we also observe geographic distribution as a recurrent feature for SoS (Nielsen et al., 2015). They have been classified by the level of managerial control.

This systems type is being more used in various emerging concepts and technologies, such as Internet of Things or Cyber-Physical Systems. Thus, there is a constant need for a SoS to be more reactive, to have an adaptive manner, and to be widely reconfigurable. This gave birth the concept of Smart SoS (SSoS). SSoS are particular SoS that inherit their basic characteristics, but also have the ability to be reconfigured dynamically in order to respond to external events that may affect them, and achieve an SoS mission, as a response to these multiple events. These events may happen concurrently and in parallel, and an SSoS should be able to handle this parallelism. This is done through a set of actions, which are defined as three key operations: (1) dynamic communication links between these systems (2) a combination of their functions to achieve a new function that neither of them can obtain on its own (3) a mobility on the SoS, by adding, removing or replacing its constituent systems.

[a] https://orcid.org/0000-0002-1760-1932
[b] https://orcid.org/0000-0003-3311-4146

Seghiri, A., Belala, F. and Hameurlain, N.
Modeling the Dynamic Reconfiguration in Smart Crisis Response Systems.
DOI: 10.5220/0011069300003176
In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2022)*, pages 162-173
ISBN: 978-989-758-568-5; ISSN: 2184-4895

## 1.2 Objective

A Smart Crisis Response SoS (SCRSoS) is one of the possible SSoS. It has the goal of crisis management, and regroups a set of independent systems that should cooperate to realize this goal, despite being fully autonomous, independent and heterogeneous complex systems. Thus, the SoS approach is justified by their ability to respond to a crisis that neither of them can manage on its own. Considering different crisis as external events, that have short decision time and most of the time may happen without notice, the need of smart systems that react to these events is essential. They need to be dynamically adaptive and mobile in order to have emerging behaviours. Thus, reconfiguring their components and natures is vital; they ought to interact by being linked together and combining their functions to achieve SoS missions. Systems in SCRSoS have a hierarchy in which, despite belonging to a specific SoS and realizing common goals, constituent systems still maintain their independent and managerial autonomy, and their own evolutionary nature. SCRSoS has an unpredictable nature, as decision making on how systems contribute depends on many factors:

- External events triggering different type of behaviours from concerned constituent systems.
- Behavioural constraints prohibiting some systems from interacting on specific events, or from sharing some of their functions with other specific systems.

By having these factors, we can achieve SoS missions in an autonomous way, defined as an event triggering certain interactions (links) between specific constituent systems, then they combine some of their roles in order to respond to that event, ensuring coherence by not violating behavioural constraints.
In order to design SCRSoS, and to execute and simulate its behaviours and reconfigurations, a formal model is needed. This model should be enough generic, thus it will be inspired from a reference architecture derived from the ISO/IEC/IEEE 42010 standard (ISO/IEC/IEEE, 2011) and also based on ArchSoS, an existing Architecture Description Language (ADL) dedicated to SoS software architectures. Maude is used to specify, implement and execute this model. The aim of this paper is twofold, we derive a referential architecture for crisis response system-of-systems in one hand, the role of communication and the characteristics of constituent systems that are needed to adapt their dynamic behavior are added to the architecture standard of IEEE. In the other hand, we define a formal model on the basis of the Maude Strategy language, to deal with the dynamic reconfiguration of these system types. Besides, this model is very appropriate to formally analyse some relveant properties of Smart-SoS.

## 1.3 Related Work

There have been few approaches in the literature dealing with the formal description and analysis of Smart and reconfigurable SoS. Among these works, we are interested in the study of the following three contributions, allowing to situate our own.

Authors in (Nielsen and Larsen, 2012) proposed an extension of VDM-RT formalism to enable the modeling of evolving SoS, their constituents, and communication channels that can be added and removed at runtime. It was applied on an intelligent Vehicle Monitoring SoS aiming to improve road safety.

Similarly, the work of (Oquendo, 2016) has proposed an ADL, known as SosADL and based on Π-calculus. It focused on how organizing the interactions among an SoS constituent systems to enable the emergence of SoS-wide behaviors derived from local behaviors. Also, this work has been extended and enhanced to cover two SoS application examples: (1) an urban river monitoring SoS illustrating self-organizing architectures (Oquendo and Legay, 2015), and (2) an IoV Internet of Vehicles serving as a support to describe an SoS-based exogenous approach (Oquendo, 2019).

On the other hand, in (Chaabane et al., 2019), authors adapted the standard “ISO/IEC/IEEE 42010: systems and software engineering-architecture description” and improved it to support SoS software architectural description. They evaluated their proposal with the Goal-Question-Metric (GQM), as an effectiveness check. The approach was used to model a Smart City SoS dealing with structural, behavioral and requirements aspects.

In our previous work, we have defined ArchSoS language, as an ADL dedicated to software architectural description of SoS. Its originality lies in the fact that, its syntax being textual or graphical, is inspired from Bigraphical Reactive Systems (Milner, 2009) and its operational semantics is based on rewriting theories (Meseguer, 1996). Then, obtained ArchSoS models are naturally executed and analyzed through Maude system (Clavel et al., ) (McCombs, 2003). In this paper, we propose to extend ArchSoS in order to deal explicitly with the principle of reconfiguration in SoS, as well as their formal analysis. Indeed, Smart SoS constituent systems have unpredictable behaviours, as they are constantly evolving and have to coordinate and interact to accomplish global missions depending on the events occurring.

### 1.4 Paper Organization

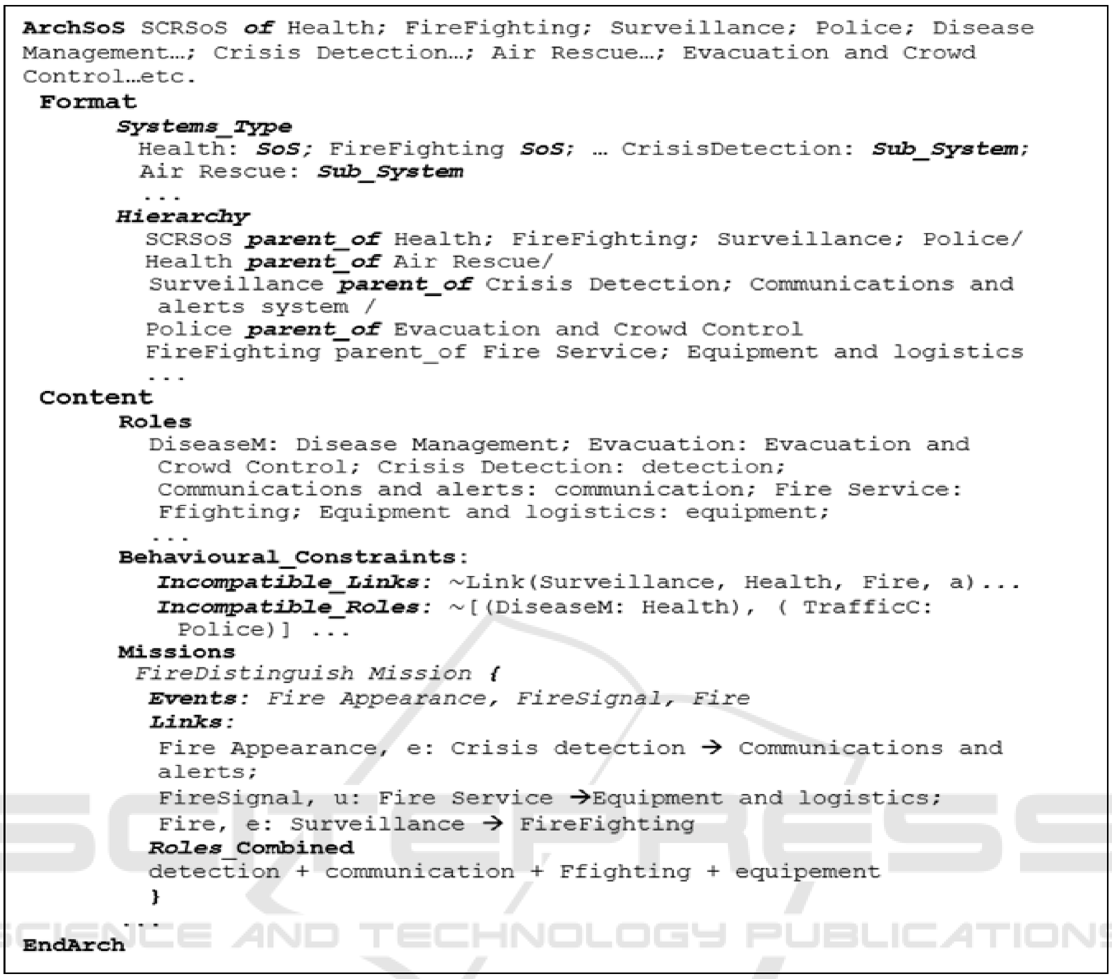
The reminder of this paper is organized as follows: Section 2 motivates our case study, which is the SCR-SoS, Section 3 defines reconfigurations in ArchSoS using a referential architecture inspired from an IEEE standard. Section 4 gives a review of Maude and its strategy language extension. Section 5 gives a formal model, centred on Maude strategies, to describe Smart SoS reconfigurations. It is executed by applying strategies concerning SCRSoS behaviours and its various events. Section 6 gives a formal analysis of some SoS relevant properties. Finally, we conclude the paper with some remarks in Section 7.

## 2 MOTIVATING EXAMPLE: SMART CRSoS

The Smart Crisis Response SoS (SCRSoS) is an SoS resulting from the collaboration of many other SoS. These latter collaborate in order to deal with bigger crisis and disasters that neither of them can maintain on its own. The main goal of having a SCRSoS is to reduce the element of surprise after having a crisis or disaster, resulting in less reaction time needed to provide immediate assistance, warnings and evacuation. By having a responsive and reconfigurable Smart SoS, several solutions to preserve human life and keep the damage to the minimum. This may be found and obtained via the cooperation of constituent systems. In this work context, SCRSoS is specified using the ArchSoS langage, more appropriate to model the structure of such complex systems and their behaviours. Figure 1 illustrates this description. We have SCRSoS and all the systems it contains, followed by two abstraction levels using:

- Format: It contains all the systems belonging to an SoS; alongside their corresponding types (SoS or atomic Sub-System) using 'Systems_Type' tag, followed by the hierarchic structure of the systems inside an SoS, which is given using the 'parent_of' tag to define a belonging relation.
- Content: It is divided into :
  - Roles: To each atomic sub-system is attached a role defining its basic and independent function. Systems may combine their roles in order to have a new role for the SoS.
  - Behavioural Constraints: A set of conditions that need to be satisfied when dealing with an SoS dynamics and evolution of constituent systems. We note that these constraints differ from one SoS to another, and they are divided into two type of constraints: (1) **Incompatible Links**: a set of constraints to indicate that some systems cannot communicate with each other. We note the communication between systems as a link, which has three possible types: a: Authority link, u: Usage link, e: data_Exchange link. The constraint has the syntax: ∼Link (Surveillance, Health, Fire, a). It states that two SoS *Surveillance* and *Health* cannot be linked with a link of type 'a' in case of a Fire event. (2) **Incompatible Roles**: states the illegal role combinations between constituent systems, it is noted: ∼[(DiseaseM: Disease Management), (Evacuation: Evacuation and crowd control)], it states that the roles *DiseaseM* and *Evacuation*, corresponding to *Disease Management* and *Evacuation and crowd control* respectively, cannot be combined together.
  - Missions: a mission is the main goal of an SoS, and a desired behaviour that is achieved by combining different roles belonging to specific linked systems, according to specific events. For instance, the *FireDistinguish* mission has three possible events, *Fire*, *FireApperance* and *Firesignal*, to each event is associated a respective link stating the systems that are linked in response to this event. The mission is achieved when SCRSoS has the specific role combination of these linked systems.

As an example of missions and events in SCRSoS, we define two possible scenarios of two different crisis occurring at the same time. SCRSoS has to be reconfigured in order to respond to the events and achieve corresponding missions: *FireDistinguish* and *HurricaneEvacuation*. The two corresponding scenarios are described in Table 1, that shows affected SoS with due to events, and specific constituent systems being reconfigured. In this example, This leads two distinct missions of SCRSoS are achieved. We notice that multiple reconfigurations are needed, to respond to external events that may happen at the same time. For instance, in order to achieve the *FireDistinguish* mission, SCRSoS needs to respond to three sequential events, which are *FireApperance*, *FireSignal* and *Fire*. The concerned constituent systems are linked depending on the event, and their roles are combined in order to produce emergent roles indicating that the mission is achieved. Through these two concurrent scenarios of SCRSoS, we tackle two main challenges that may



```
ArchSoS SCRSoS of Health; FireFighting; Surveillance; Police; Disease
Management…; Crisis Detection…; Air Rescue…; Evacuation and Crowd
Control…etc.
 Format
        Systems_Type
          Health: SoS; FireFighting SoS; … CrisisDetection: Sub_System;
          Air Rescue: Sub_System
          ...
        Hierarchy
          SCRSoS parent_of Health; FireFighting; Surveillance; Police/
          Health parent_of Air Rescue/
          Surveillance parent_of Crisis Detection; Communications and
           alerts system /
          Police parent_of Evacuation and Crowd Control
          FireFighting parent_of Fire Service; Equipment and logistics
          ...
 Content
        Roles
          DiseaseM: Disease Management; Evacuation: Evacuation and
           Crowd Control; Crisis Detection: detection;
           Communications and alerts: communication; Fire Service:
           Ffighting; Equipment and logistics: equipment;
          ...
        Behavioural_Constraints:
           Incompatible_Links: ~Link(Surveillance, Health, Fire, a)...
           Incompatible_Roles: ~[(DiseaseM: Health), ( TrafficC:
             Police)] ...
        Missions
         FireDistinguish Mission {
          Events: Fire Appearance, FireSignal, Fire
          Links:
           Fire Appearance, e: Crisis detection → Communications and
           alerts;
           FireSignal, u: Fire Service →Equipment and logistics;
           Fire, e: Surveillance → FireFighting
          Roles_Combined
          detection + communication + Ffighting + equipement
          }
      ...
EndArch
```

Figure 1: ArchSoS description of SCRSoS.

occur when having multiple events at the same time:

- SCRSoS may be reconfigured and deal with **parallel events** without affecting each other.
- SCRSoS may be reconfigured in oder to respond to a specific disaster and crisis before others, depending on their gravity or their response order, for instance *FireAppearance* event need to be dealt with before the *FireSignal* event, and the *Fire* event should follow after. Another example would be that the *Fire* event should be **prioritized** over the *Hurricane* event, because naturally a fire needs a faster response than a hurricane that takes times to strike.

## 3 MODELING SMART SoS IN ArchSoS

We adapt ArchSoS definition to the ISO/IEC/IEEE 42010 standard, as an architectural reference to describe SCRSoS structures, and constraint its behaviours. This standard has a conceptual model representing the notions of systems, components, and their environment. It contains an architectural description of a system architecture, represented by architectural views and architectural models. These models are built according to specific formalisms and notations. However, this standard needs to be adapted in order to address SoS characteristics and their challenges. Thereby, allowing the definition of SoS models in accordance to this standard.

Authors in (Chaabane et al., 2019) adapted the ISO/IEC/IEEE 42010 standard and improved it to deal with SoS by replacing the system of interest as-

Table 1: SCRSoS Scenarios.

| Event | Affected SoS | Reconfigured Systems |
|---|---|---|
| Scenario 1 : FireDistinguish Mission | | |
| Fire-Appearance | Surveillance | Crisis detection system + Communications and alerts system |
| FireSignal | FireFighting | Fire Service system + Equipment and logistics system |
| Fire | SCRSoS | Surveillance SoS + FireFighting SoS |
| Scenario 2 : HurricaneEvacuation Mission | | |
| Hurricane-Appearance | Surveillance | Crisis detection system + Communications and alerts system |
| Hurricane-Signal | Police | Evacuation and crowd control system + Traffic control system |
| Hurricane | SCRSoS | Surveillance SoS + Police SoS |

pect of the metric with a constituent systems entity, linked to a system-of-systems entity. In a similar way, we extend this standard by replacing the system of interest with a Smart SoS entity containing other ArchSoS entities and elements. We also introduce the ability to reconfigure Smart SoS architectures, and the architectural models. Figure 2 illustrates this extension (blue rectangles).

The architecture description is the key element of the standard conceptual model, it identifies three elements (1) the stakeholders, which are the most interested entities in a system, such as developers, maintainers, and mostly a designer that defines an SoS structure and constraints in ArchSoS. He may also trigger events to simulate SoS behaviours. (2) The concerns, representing the purpose of a system and its potential achievement, its implemention feasibility and its capacity to evolve. It represents the SCRSoS missions. (3) The systems of interest exhibiting this architecture, which are in our case SSoS. They are supposed to have their complexity reduced while having a high abstract software description.

Stakeholders and concerns are identified by an architecture description of an SSoS entity architecure. A SSoS is composed by of SSoS, and of other subsystems entity, delimiting atomic systems that have no constituent systems. Both SSoS and sub-systems can be linked to other SSoS or sub-systems respectively, and they may contain roles that can be combined. We note that links and role combinations are constrained by the SCRSoS behavioural constraints An architectural view is composed of architecture models, governed by model kinds. In our case, we adapt Maude models as an executable rewrite theory of SSoS. Maude's strategy language offers additional specifications and control over the Maude model execution.

SoSs are known by their evolutionary nature; either on the structural level by changing the SoS hierarchy and its actual constituent systems, or on the behavioural level by defining dynamic interactions and collaborations between them, leading to flexible reconfigurations. This evolutionary nature makes it difficult to capture SoS behaviours, and how they are conducted relatively to external events that may occur. We should specifiy how SoS missions are achieved, in a coherent and autonomous way. This is the case of Smart SoS; no central control on their evolution. Smart SoS have autonomous constituent systems that can adapt to external events, and evolve to fulfil an SoS mission.

This paper addresses this challenging issue, by investigating and focusing on SoS reconfigurations. It defines a formal approach to guide and constrain SoS behaviours. We use a Maude extension, known as the Maude strategy language (Martí-Oliet et al., 2009) (Rubio et al., 2022) to define deterministic behaviours of SoS in ArchSoS, as there is no control on how Maude rewrite theories are executed. This will ensure that the SCRSoS constituents can be reconfigured according to specific events, in order to achieve the SoS missions.

## 4 MAUDE STRATEGY LANGUAGE REVIEW

Maude (Clavel et al., ) (McCombs, 2003) is a language that implements rewriting logic, as formal semantic framework. It allows the definition of concurrent computations of complex systems, considering the following two aspects:

- The structural aspect using a mathematical set of equations defining the syntax of the system.
- The dynamic aspect using rewrite rules, that from an initial state Si, make a transition to a next state Sj, thereby changing the general behavior of the system.

The main unit of programming and specifying in Maude language is a module. We distinguish two types of modules in the standard Maude: (1) a Functional module, which uses equational logic, defining the syntactic structure of a system, (2) a System module, to define a system dynamic evolution, as an application of rewrite rules that can be conditioned. The

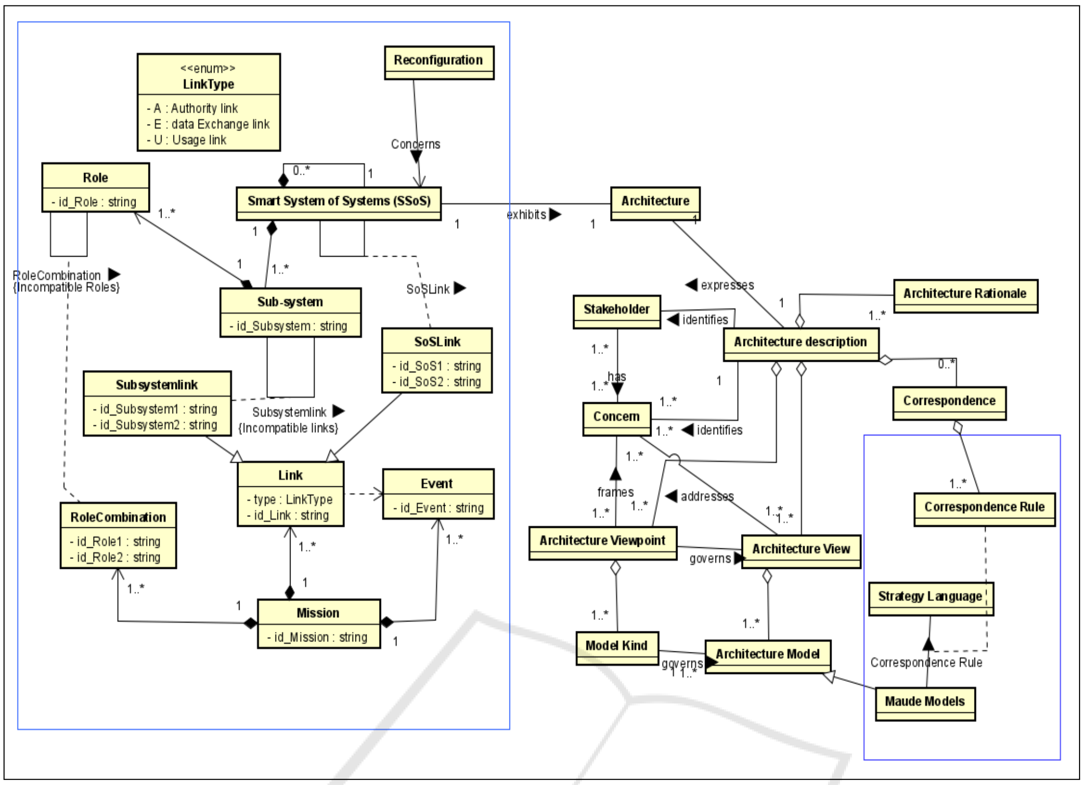


Figure 2: Extending ArchSoS according to the ISO/IEC/IEEE 42010 standard.

```
smod ArchSoSStrategies is
protecting ArchSoSBehaviours .
including STRATEGY-MODEL-CHECKER .
including ArchoSoSAnalysis .
strat FireDistinguishS @ SoS .
strat HurricaneEvacuationS @ SoS .
strat CrisisResponseS @ SoS .

  sd FireDistinguishS := ( LinkFireApperance ; AcquireRoleFireAppearance ; LinkFireSignal
  ; AcquireRoleFireSignal ; LinkFire ; AcquireRoleFire ) .

  sd HurricaneEvacuationS := ( LinkHurricaneAppearance ; AcquireRoleHurricaneApperance
  ; LinkHurricaneSignal ; AcquireRoleHurricaneSignal ; LinkHurricane ; AcquireRoleHurricane) .

  sd CrisisResponseS := ( FireDistinguishS | HurricaneEvacuationS ) .

  op initial : -> SoS .
  eq initial =  opSoS< CrisisControl , null >.P[
    opSoS< Surveillance ,  null >.P[
     opsubSyS< CAlert , communications >.L[ null ]|
     opsubSyS< CDetection , detection >.L[ null ] ]
     .L[ null ].E[ FireAppearance , HurricaneAppearance ]
  | opSoS< FireFighting , null >.P[opsubSyS< EquipementsL , equipements >.L[ null ] |
     opsubSyS< FireS , Ffighting >.L[ null ] ]
     .L[ null ].E[ FireSignal , HurricaneSignal ]
  | opSoS< Police , null >.P[opsubSyS< TrafficC ,  TManagement >.L[ null ] |
     opsubSyS< EvacuationCC , CManagement >.L[ null ] ].L[ null ].E[ null ] ]
     .L[ null ].E[ Fire , Hurricane ]  .
endsm
```

Figure 3: SCRSoS Strategy module.

usage of both these modules covers both the structural and behavioural aspects of a system.

Maude modules extensions may be defined to make its specifications more expressive. For instance, we may cite Object-Oriented Maude (Clave et al., 2000) which models systems as a multiset of entities represented as objects with unique identities, communicating with message passing. Real Time Maude (Ölveczky and Meseguer, 2007) is another Maude extension used to define real-time/embedded systems, equipped with timed rewriting for simulations.

In this paper, we are interested by Maude's strategy language that separates between rewrite rules of a Maude system module, and their rewrite guidance using strategies. This language has its own operational semantics, allowing the manipulation of strategies without the need of being familiar with the reflective concepts of Maude META-LEVEL module. It defines internal strategies as rewrite rules (Martí-Oliet et al., 2009) (Rubio et al., 2022). By using Maude's strategy language, we can associate to each Maude system module a strategy module that contains strategy expressions defining the number, type, and order of executing rewrite rules. Thus, the semantic of a system does not only depend on the system module itself, but also on its defined set of strategies. The syntax of a strategy module is given by the Figure 4, that we will detail in the following :

- Strategy module declaration is achieved with the keyword *'smod'*. It contains a set of strategies, they concern a specific Maude system module designed by the *protecting* tag. The module itself can use and import other strategy modules using the *including* tag.
- Strategy declaration is decklared by the keyword *'strat'*. It contains two arguments : the identifier of the strategy (S1, S2 in figure 4), and the sort that is affected by this strategy.
- Strategy definition is also possible using the *sd*(strategy definition) or the *csd*(conditional strategy definition) tags, the latter defines a strategy that is applied only when a condition is fulfilled. This strategy definition may have the following syntax :

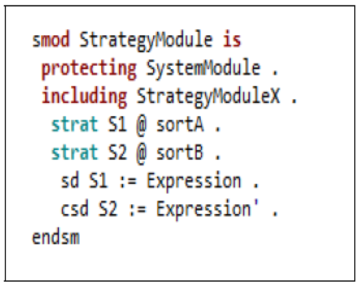
  $$sd\ strategy\_identifier := Expression$$

  $$csd\ strategy\_identifier := Expression\ if\ Condition$$

  Expressions are a set of terms defining how exactly rewrite rules are applied.

In Maude strategy language, a possible execution involving the strategy concept is done via the following command:

$$srew\ Term\ using\ S1$$



```
smod StrategyModule is
 protecting SystemModule .
 including StrategyModuleX .
  strat S1 @ sortA .
  strat S2 @ sortB .
   sd S1 := Expression .
   csd S2 := Expression' .
endsm
```

Figure 4: A Strategy Module syntax.

where Term represents a state of a system, and S1 is the used strategy to rewrite this state term.

The Maude strategy language also offers the possibility to organize the execution of strategies and their rewrite rules, giving a priority to certain ones above the others, and constraining how they should be executed according to a set of operators. The main operators are summarized in Table 2 .

We have to notice that in the operator ";", the system is constrained to run all rules of the strategy S1 before being able to run the rules of S2. If the order of the rules is inversed (noted S2; S1), then S2 is executed first and S1 follows next, if it is possible to run it after S2. This offers the possibility to force a system to have an execution pattern where critical rules should always be applied.

The operator "|" constraints the system to run (if possible) S2 even if S1 cannot be executed. However, when the rewriting occurs, S1 has a priority over S2 (the opposite way when they are inversed). The semantic of an SoS using this operator offers possibility to order the execution according to the importance of the system's behaviours.

## 5 RECONFIGURATION FORMAL MODEL FOR SSoS

In this Section, we show how we associate a strategy-based rewrite theory to define the structural and behavioural aspects of SCRSoS. We are particularly interested by specifying the reconfiguration management using a set of proposed strategies. Formally, our model using the strategy language has the following definition:

$$def_{ArchSoS} = \{\Sigma_{ArchSoS}, (E \cup A)_{ArchSoS}, S_{ArchSoS}$$

$$(R_{ArchSoS}, SM_{ArchSoS})\}$$

It describes SoS architectures expressed in ArchSoS, while specifying their semantics via a set of Maude strategy modules :

Table 2: Strategy Language Operators.

| Operator | Syntax | Functionality |
|---|---|---|
| ; | sd Strategy0 := S1, S2 | Concatenation (Associative ) |
| \| | sd Strategy0 := S1 \| S2 | Union (Associative and Commutative) |
| * | sd Strategy =: (. . . . . . )* | At least 0 iteration |
| + | sd Strategy =: (. . . . . . )+ | At least 1 iteration |
| ! | sd Strategy =: (. . . . . . )! | All iterations |

Table 3: Correspondance table between ArchSoS aspects and Maude.

| Aspect | ArchSoS entities | Maude specification |
|---|---|---|
| ArchSoS Syntax | SoS,<br>Sub-Systems<br>and events | sort SoS idSoS.<br>subsort subSyS $<$ SoS .<br>sort Event .<br><br>op opSoS $<\_,\_>.P[\_].L[\_].E[\_]$ :<br>IdSoS Role SubSyS Link Event $\rightarrow$ SoS [ctor] . op null : $\rightarrow$ SoS [ctor] .<br>op $\_\|\_$ : SoS SoS $\rightarrow$ SoS [ctor assoc comm id: null] .<br><br>sort IdSubsystem .<br>op opsubSyS $<\_,\_>.L[\_]$ :<br>IdSubsystem Role Link $\rightarrow$ subSyS [ctor] .<br>op $\_\|\_$ : subSyS subSyS $\rightarrow$ subSyS [ctor assoc comm id: null] . |
| | Roles | sort Role .<br>op ”RoleName” $\rightarrow$ Role [ctor].<br>op null : $\rightarrow$ Role [ctor] .<br>op $\_+\_$ : Role Role $\rightarrow$ Role [ctor assoc comm id: null] . |
| | Links | sorts Link LinkT.<br>op link $<\_:\_;\_\rightarrow\_>$ :<br>Event LinkT IdSoS IdSoS $\rightarrow$Link [ctor] .<br>op a : $\rightarrow$ LinkT [ctor] .<br>op u : $\rightarrow$ LinkT [ctor] .<br>op e : $\rightarrow$ LinkT [ctor] . |
| | Events | sort Event (sort inside the opSoS construct). |
| ArchSoS Semantic | Control Predicates $\phi_i$ | Maude Conditional Equations:<br><br>IsActive(SoS), CompRoles(Rolei, Rolej), NoComp(SoS), isLinked(Subsystemi), areLinked (Subsystemi, Subsystemj), PosLink(Subsystemi,Subsystemj,Event, LinkT), NoClinks(Si), CanAdd(SoS,SubSystemi) CanRemove(SoS,SubSystemi) . |
| | Action | Maude Conditioned Rewrite Rule<br>crl [rewrite-rule-name] : State$_i$ $\rightarrow$ State$_j$ if $\phi_i$ . |
| | Mission | Strategy definition S$_i$ |
| | Reconfiguration | Strategies and rewrite rules combination using operators . |
| | Verification Property | Maude LTL Property Prop$_i$ |

- $\Sigma_{ArchSoS}$ and $(E \cup A)_{ArchSoS}$ represent the ArchSoS syntax using an equational theory giving to each ArchSoS entity (SoS, sub-systems, roles, links, events), its rigorous semantics (meaning).
- The semantic $S_{ArchSoS}$ is defined by the tuple $(R_{ArchSoS}, SM_{ArchSoS})$:
  - $R_{ArchSoS}$ identifies a set of rewrite rules representing actions that may affect SSoS behaviours, and a set of predicates that constraints the application of these rewrite rules.
  - $SM_{ArchSoS}$ is a set of strategies that may be combined via operators to define ArchSoS possible reconfigurations.

Table 3 defines the correspondence between ArchSoS and its Maude specification. In the ArchSoS part, we declare all sorts and operators that serve to define ArchSoS entities. The main user-defined operator is "opSoS"; it constructs a SoS having a clear and formal syntax. The semantic part of this table (Table 3) illustrates how we associate to each behavioural element of ArchSoS, a rigorous semantic. For instance, one reconfiguration is defined by a strategy S$_i$, which containts a set of rewrite rules combined via Maude strategy language dedicated operators.

Our contribution will be more explained through the scenarios taken from the proposed case study (SCRSoS). In order to deal with both SCRoS missions simultaneously, our previous behavioural model does not define this situation, the rewriting of the corresponding SCRSoS states must be done sequentially and separately. In this present work, we use some strategies definition to tackle this problem. Figure 3 illustrates the strategy module.

- *FireDistinguishS* strategy: Is defined to achieve the *FireDistinguish* mission, it has six rewrite rules reconfigured sequentially with the ";" operator, stating that each rule needs to be executed before the next one is applied.
- *HurricaneEvacuationS* strategy: Similarly, it is proposed to achieve the *HurricaneEvacuation* mission.
- *CrisisResponseS* strategy: It represents the global reconfiguration of the SCRSoS, and it is composed of the *FireDistinguishS* strategy, followed by the *HurricaneEvacuationS* strategy using the operator "|". It states that even if the *FireDistinguish* mission is not achieved where the fire event does not happen; the*HurricaneEvacuationS* strategy still carries on and is executed (a characteristic of the union operator "|"), there is no dependency between both strategies. Besides, if both strategies can be executed when the SCRSoS strategy is applied, the *FireDistinguish* one has higher priority than the *HurricaneEvacuationS*, it is executed before the latter.

An initial state of SCRSoS is defined in Figure 3. It contains all the six events that may be used to describe both SCRSoS scenarios. Both scenarios can be executed with the command:

*srew initial using CrisisResponse*.

The execution results are shown in Figure 5. They state that both the *FireDistinguishS* strategy and the *HurricaneEvacuationS* strategy are executed simultaneously, and two distinct solutions are available, with *FireDistinguishS* being the first solution. It is worth to note that:

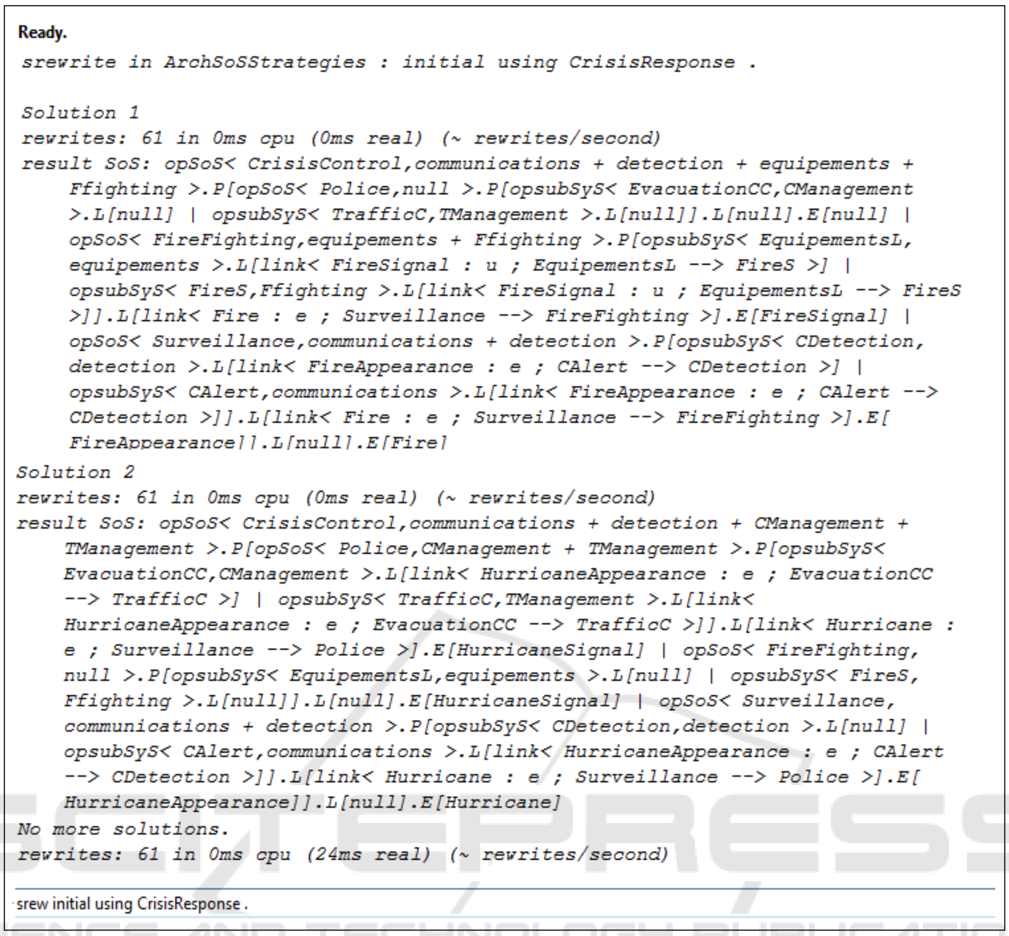
- The SCRSoS strategy execution has the ability to deal with **parallel** and concurrent events. Thus, it can respond to many crisis at the same time.
- In addition, while defining strategies in a sequential order, a **priority** is given to some scenarios over others, this is interpreted as dealing with urgent crisis first, before crisis that have more reaction time.

## 6 FORMAL ANALYSIS

Formal models of SoS specified with this ArchSoS extension have been executed and prototyped through Maude system, in order to show how we deal with dynamic reconfiguration of Smart SoS. In this section, we exploit the model-checker tool of Maude to express and analyse SoS inherent properties. We develop a new Maude module called ArchSoSAnalysis containing a set properties defined using LTL syntax (more details can be found on [Rozier, 2011]) :

- *inactivity:* Checks if an SoS role is inactive, stated by the word 'null'.
- *roles-violation:* Checks if a role combination has occurred, despite having combined roles that are not compatible.
- *links-violation:* Checks if two constituent systems are linked while being incompatible on that specific link;
- *vivacity:* It ensures that something good will always happen. It checks that every SoS on which the inactivity property holds, will always end up active:

$$eq\, vivacity = [\,] <> (inactivity \Rightarrow \sim inactivity).$$

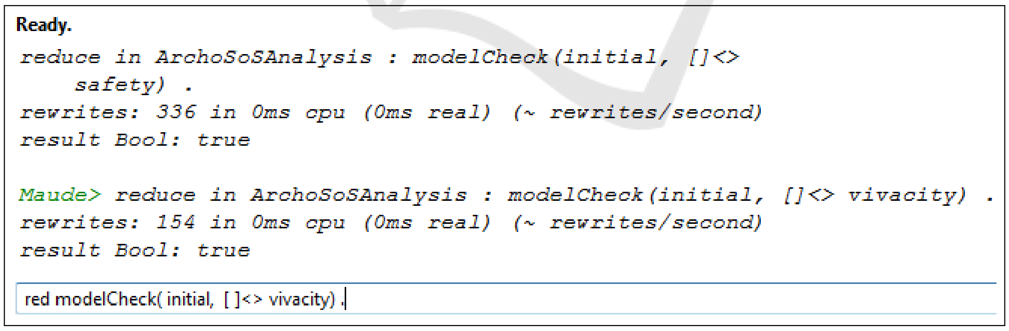


```
Ready.
srewrite in ArchSoSStrategies : initial using CrisisResponse .

Solution 1
rewrites: 61 in 0ms cpu (0ms real) (~ rewrites/second)
result SoS: opSoS< CrisisControl,communications + detection + equipements +
    Ffighting >.P[opSoS< Police,null >.P[opsubSyS< EvacuationCC,CManagement
    >.L[null] | opsubSyS< TrafficC,TManagement >.L[null]].L[null].E[null] |
    opSoS< FireFighting,equipements + Ffighting >.P[opsubSyS< EquipementsL,
    equipements >.L[link< FireSignal : u ; EquipementsL --> FireS >] |
    opsubSyS< FireS,Ffighting >.L[link< FireSignal : u ; EquipementsL --> FireS
    >]].L[link< Fire : e ; Surveillance --> FireFighting >].E[FireSignal] |
    opSoS< Surveillance,communications + detection >.P[opsubSyS< CDetection,
    detection >.L[link< FireAppearance : e ; CAlert --> CDetection >] |
    opsubSyS< CAlert,communications >.L[link< FireAppearance : e ; CAlert -->
    CDetection >]].L[link< Fire : e ; Surveillance --> FireFighting >].E[
    FireAppearance]].L[null].E[Fire]
Solution 2
rewrites: 61 in 0ms cpu (0ms real) (~ rewrites/second)
result SoS: opSoS< CrisisControl,communications + detection + CManagement +
    TManagement >.P[opSoS< Police,CManagement + TManagement >.P[opsubSyS<
    EvacuationCC,CManagement >.L[link< HurricaneAppearance : e ; EvacuationCC
    --> TrafficC >] | opsubSyS< TrafficC,TManagement >.L[link<
    HurricaneAppearance : e ; EvacuationCC --> TrafficC >]].L[link< Hurricane :
    e ; Surveillance --> Police >].E[HurricaneSignal] | opSoS< FireFighting,
    null >.P[opsubSyS< EquipementsL,equipements >.L[null] | opsubSyS< FireS,
    Ffighting >.L[null]].L[null].E[HurricaneSignal] | opSoS< Surveillance,
    communications + detection >.P[opsubSyS< CDetection,detection >.L[null] |
    opsubSyS< CAlert,communications >.L[link< HurricaneAppearance : e ; CAlert
    --> CDetection >]].L[link< Hurricane : e ; Surveillance --> Police >].E[
    HurricaneAppearance]].L[null].E[Hurricane]
No more solutions.
rewrites: 61 in 0ms cpu (24ms real) (~ rewrites/second)

srew initial using CrisisResponse .
```

Figure 5: SCRSoS strategy execution.



```
Ready.
reduce in ArchoSoSAnalysis : modelCheck(initial, []<>
    safety) .
rewrites: 336 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true

Maude> reduce in ArchoSoSAnalysis : modelCheck(initial, []<> vivacity) .
rewrites: 154 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true

red modelCheck( initial, [ ]<> vivacity) .
```

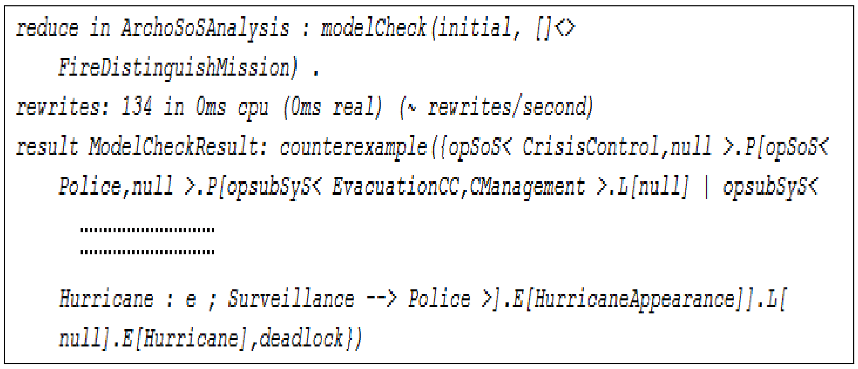
Figure 6: Safety and Vivacity model-checking results.

- *safety:* It ensures the *correct* evolution of SoS behaviours, by respecting the defined behavioural-constraints. It checks that both behavioural-constraints, for links and roles, are not violated through all SoS evolution states, it is noted as :

$$eq\, safety = [](\sim links\text{-}violation /\backslash$$
$$\sim roles\text{-}violation).$$

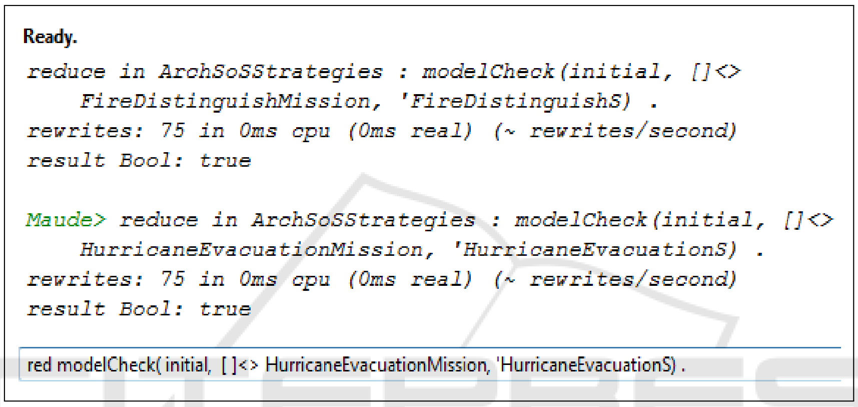
- *FireDistinguishMission:* A property that checks if SCRSoS will have a state where the roles-combination required to achieve the *FireDistinguish* mission holds.
- *HurricaneEvacuationMission:* A property that checks the presence of the roles needed for the *HurricaneEvacuation* mission on a SCRSoS evolution state .

A given Property is executed using the model-checker



```
reduce in ArchoSoSAnalysis : modelCheck(initial, []<>
    FireDistinguishMission) .
rewrites: 134 in 0ms cpu (0ms real) (~ rewrites/second)
result ModelCheckResult: counterexample({opSoS< CrisisControl,null >.P[opSoS<
    Police,null >.P[opsubSyS< EvacuationCC,CManagement >.L[null] | opsubSyS<
      ...............................
      ...............................
    Hurricane : e ; Surveillance --> Police >].E[HurricaneAppearance]].L[
    null].E[Hurricane],deadlock})
```

Figure 7: *FireDistinguishMission* property model-checking using Maude.



```
Ready.
reduce in ArchSoSStrategies : modelCheck(initial, []<>
    FireDistinguishMission, 'FireDistinguishS) .
rewrites: 75 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true

Maude> reduce in ArchSoSStrategies : modelCheck(initial, []<>
    HurricaneEvacuationMission, 'HurricaneEvacuationS) .
rewrites: 75 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true

red modelCheck( initial,  [ ]<> HurricaneEvacuationMission, 'HurricaneEvacuationS) .
```

Figure 8: SCRSoS missions' model-checking using Maude strategies.

tool via the command :

$$red\, modelCheck(initial, LTLoperators\ P)$$

Where P is the property to verify, preceded by a combination of the LTL operators $<>$ and $[]$ to define if the property needs to hold at least once in the evolution process, or to always hold for instance. Figure 6 shows the execution results of both the *vivacity* and the *safety* properties. They both return *true* indicating that they are valid through the execution of SCRSoS evolution actions.
However, when executing the *FireDistinguishMission* or the *HurricaneEvacuationMission*, both return a *counterexample* stating that both properties are not valid. This is explained by the fact of having two disctinct scenarios. Maude can only execute one scenario at once, and since both scenarios have the same initial state, thus affecting each other when executed. For instance, Figure 7 shows the counter-example from the analysis of the *FireDistinguishMission* property, it represents the execution of the*Hurricane* scenario as a situation where the property *FireDistinguishMission* never holds.

In order to deal with this problem and to validate the properties concerning SCRSoS missions, we need to verify each property by applying the strategy defined for specific missions in the model-Checking process. It will have the following syntax (Rubio et al., 2022):

$$red\, modelCheck(initial, [] <> MissionP,' MissionS)$$

where MissionP is the property that verifies the mission validation, and MissionS is the strategy to apply on the model-checking process, preceded by the symbol ’′’. Figure 8 shows the new execution results for both properties *FireDistinguishMission* and *HurricaneEvacuationMission*. Now, they return *true*, indicating that they are valid.

We notice that the SCRSoS missions' properties that couldn't be validated using standard Maude model-checking, are validated when reconfiguring the SCRSoS through specific strategies for each mission.

## 7 CONCLUSIONS

In this paper, we have extended ArchSoS, an ADL dedicated to describing SoS architectures in order to deal with Smart SoS. First, we have defined a conceptual model for SoS by adapting ArchSoS concepts to the ISO/IEC/IEEE 42010 standard. Then, we have formalized the extended version of ArchSoS using rewriting logic. Smart SoS architectures and their behaviors have been defined through a set of Maude modules allowing the formal execution and analysis of these models. Rewriting rules and predicates have been proposed to give a natural semantics to SoS behavior evolution. Dynamic reconfiguration of Smart SoS due to many possible external events was well specified using the concept of strategy provided by the Maude Strategy language. Three strategies have been defined for the considered case study SCRSoS, their execution has shown that two crisis scenarios may be applied simultaneously to achieve two SoS missions at the same time.

In addition, we have proposed a set of properties that we have checked positively using the model-checker LTL of Maude. Thus, we have shown the relevance of Maude strategies while validating some properties that remain not valid with the previous model of ArchSoS. In future, we plan to equip ArchSoS with temporal constraints in order to check non-functional properties. Other case studies are also possible to better illustrate the contributions of this language.

## REFERENCES

Chaabane, M., Rodriguez, I. B., Colomo-Palacios, R., Gaaloul, W., and Jmaiel, M. (2019). A modeling approach for systems-of-systems by adapting iso/iec/ieee 42010 standard evaluated by goal-question-metric. *Science of Computer Programming*, 184:102305.

Clave, M., Durán, F., Eker, S., Lincoln, P., Marti-Oliet, N., Meseguer, J., and Quesada, J. F. (2000). Towards maude 2.0. *Electronic Notes in Theoretical Computer Science*, 36:294–315.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcot, C. *All About Maude - A High-Performance Logical Framework : How to Specify, Program, and Verify Systems in Rewriting Logic*. Programming and Software Engineering , 4350. Springer-Verlag Berlin Heidelberg.

ISO/IEC/IEEE (2011). Systems and software engineering – architecture description. *ISO/IEC/IEEE 42010: 2011 (E)(Revision of ISO/IEC 42010: 2007 and IEEE Std 1471-2000)*, pages 1—-46.

Martí-Oliet, N., Meseguer, J., and Verdejo, A. (2009). A rewriting semantics for maude strategies. *Electronic Notes in Theoretical Computer Science*, 238(3):227–247.

McCombs, T. (2003). Maude 2.0 primer. *Department of Computer Science, University of Illinois and Urbana-Champaign, Urbana-Champaign, Ill., USA*.

Meseguer, J. (1996). Rewriting logic as a semantic framework for concurrency: a progress report. In *International Conference on Concurrency Theory*, pages 331–372. Springer.

Milner, R. (2009). *The space and motion of communicating agents*. Cambridge University Press.

Nielsen, C. B. and Larsen, P. G. (2012). Extending vdm-rt to enable the formal modelling of system of systems. In *2012 7th International Conference on System of Systems Engineering (SoSE)*, pages 457–462. IEEE.

Nielsen, C. B., Larsen, P. G., Fitzgerald, J., Woodcock, J., and Peleska, J. (2015). Systems of systems engineering: basic concepts, model-based techniques, and research directions. *ACM Computing Surveys (CSUR)*, 48(2):1–41.

Ölveczky, P. C. and Meseguer, J. (2007). Semantics and pragmatics of real-time maude. *Higher-order and symbolic computation*, 20(1-2):161–196.

Oquendo, F. (2016). π-calculus for sos: A foundation for formally describing software-intensive systems-of-systems. In *2016 11th System of Systems Engineering Conference (SoSE)*, pages 1–6. IEEE.

Oquendo, F. (2019). Architecting exogenous software-intensive systems-of-systems on the internet-of-vehicles with sosadl. *Systems Engineering*, 22(6):502–518.

Oquendo, F. and Legay, A. (2015). Formal architecture description of trustworthy systems-of-systems with sosadl. *ERCIM News*, (102).

Rubio, R., Martí-Oliet, N., Pita, I., and Verdejo, A. (2022). Model checking strategy-controlled systems in rewriting logic. *Automated Software Engineering*, 29(1):1–62.

Van Veelen, J., Van Splunter, S., Wijngaards, N., and Brazier, F. (2008). Reconfiguration management of crisis management services. In *The 15th conference of the International Emergency Management Society (TIEMS 2008)*.