A Formal Language for Modelling and Verifying Systems-of-Systems Software Architectures

Akram Seghiri, Faiza Belala, Nabil Hameurlain

1 INTRODUCTION

System of systems (SoS) (Maier, 1996) are an emerging class of complex, distributed and independent systems that are cooperating and interacting to accomplish specific goals, known as SoS missions, which neither of them can achieve nor obtain on its own. Various challenges have to be considered when modelling and specifying a SoS, as for instance: 1) **Missions:** they represent the main goal of a SoS and its desired behaviour that it needs to attend. Several constituent systems may combine their roles to achieve a certain mission. 2) **Interactions:** A mission of a SoS can only be achieved by a cooperation between its constituent systems. 3) **Hierarchy**: A SoS is constituted with a set of sub-systems. Each one may also be a SoS. These constituents are, autonomous and operating independently. Besides, they may operate to achieve the SoS mission. 4) **Evolution:** The constituent systems are constantly changing their interactions and role combinations, yielding to dynamic behaviours of a SoS. 5) **Dynamic Controls:** A SoS dynamic evolution should be controlled, to prohibit undesired behaviours that may occur in the SoS and in its constituents.

In order to overcome these challenges and to capture the dynamic nature of a SoS, we have defined in a previous work ArchSoS (Seghiri et al., 2018), a formal Architectural Description Language (ADL). Its main purpose is to model hierarchical structures of these systems and their reconfigurations, as well as the ability to manage potential cooperation between their constituent systems.

In this paper, we extend ArchSoS by a syntax-driven description, that has the ability to model SoS constituents, and the events affecting them, as well as their behavioural constraints.

Particularly, the main objective of this paper is twofold. On one hand, a formal and graphical syntax, inspired by Bigraphic Reactive Systems (BRS) principle (Milner, 2001) is given to ArchSoS. This will offer a syntax-driven SoS description, dealing with the hierarchy concepts and interactions between systems. On the other hand, a guided rewriting-based operational semantic is associated to ArchSoS. Rewrite theories (Meseguer, 1996) seem to be appropriate for defining a SoS semantic. ArchSoS specifications are implemented using Maude language (Clavel et al., 2007) to execute and simulate them, while analyzing their pertinent properties.

A Bigraphic Reactive System (BRS) is a formalism for describing and modelling computational systems. It contains a *Bigraph* model, representing the structural aspect of a system, and a set of *Reaction Rules* to describe how bigraphical components may be reconfigured, defining the semantic of a system. A Bigraph is defined by a set of nodes and a set of links between those nodes, known as edges. The nodes have the ability to be nested inside one another. A Bigraph may also have algebraic notations, which are equivalent to graphical ones. A reaction rule has the form $R \rightarrow R'$, where R is called the *redex* (Bigraph before the state transition), and R' is the *reactum* (Bigraph after the state transition). Thus, in a transition of a Bigraph using a specified rule, the *redex* is the part of the Bigraph that is matched, and will be replaced by the matching part of the *reactum*.

We have chosen BRS as a base for modeling SoS systems since it allows a formal and visual technique for designing SoS software architectures, it provides enough expressive ability to represent not only SoS components and elements, but also their dynamic specification through the use of reaction rules.

On the other hand, Maude which is an implementation language based on Rewriting logic, is chosen as a formal semantic framework for ArchSoS. It uses a mathematical notations represented by a rewriting theory = $(\Sigma, E, R,$ where the signature (Σ, E) describes its static structure, while the rewriting rules R describe its behaviour and evolution, our choice of using rewrite theories is justified by their ability of reasoning on concurrent systems, representing their states and their different transitions. Maude emphasizes *simplicity, expressiveness and performance* (Clavel et al., 2007), and offers a particular syntax using a set of sorts and operations to define concurrent computations between systems, allowing the execution of its formal specifications and formal verification.

The rest of this paper is organized as follows: Section 2 discusses the Related Work. Section 3 presents an overview of our approach. Section 4 presents ArchSoS and its concrete syntax, illustrated through a realistic example. In Section 5, an operational semantic for ArchSoS is given to define SoS evolutions. Section 6 is devoted to explain how to take benefit from the Maude tool to implement ArchSoS and to formally analyze some of ArchSoS qualitative properties using Linear Temporal Logic (LTL). Finally, Section 7 discusses future work and concludes the paper.

2 RELATED WORK

Table 1 provides a summary of various approaches dealing with SoS modelling in the context of software engineering. Only researchers using formal models in one of the stages of the SoS Engineering process are identified. They are classified according to the above-mentioned challenges. The focus concerns the hierarchical aspect of a SoS structure, the mobility of its links, its execution semantics and its formal analysis.

In the literature, some researchers have opted for the use of model-based approaches to specify and design SoS; Systems Modeling Language (SysML) has been used in, (Dahmann et al., 2007), (Lane and Bohn, 2013), (Bryans et al., 2014) and (Hause, 2014), to represent SoS characteristics, diagrams are used to specify system's structures and relations between them. SysML offers a good representation of a SoS hierarchical structure, alongside the links between constituent systems. It is more used to deal with modelling and static aspects only; the evolution is implemented using programming languages and is limited to system restructuration.

Formal techniques, such as Bigraphs, were alternatively used in (Stary and Wachholder, 2016) and (Gassara et al.,2017) to model SoS. (Gassara et al., 2017) adapted a multi-scale modeling methodology that was applied to a smart buildings SoS. In these approaches, the structure of a system is defined as a concrete Bigraph, while dynamics are represented by reaction (transition) rules. On the other hand, authors in (Nielsen and Larsen, 2012) proposed a formal approach based on the extension of an object-oriented technique, the Vienna Development Method Real-Time (VDM-RT). This extension allows the initial architecture of a SoS to be changed during run-time through dynamic reconfiguration operations. Authors in (Woodcock et al., 2012) proposed a formal modeling language called *CML* (Compass Modelling language), based on Circus, focusing on the geographical distribution and the topology aspects of SoS. In (Derhamy et al., 2019), authors presented a graph model to build SoS architectures by adapting SOA theories and its query approach, SoS architectures are built. Constituent systems are composed to form SoS in a decentralized manner.

Table 1. SoS Modelling Approaches

	SoS Syntax Description			**SoS Semantic Description**		**Formal Execution and Verification**
Existing Approaches Based on :	***Hierarchy***	***Links***	***Missions***	***Evolution***	***Behavioural Constraints***	***Implementation***
SYSML (Dahmann et al., 2007), (Lane and Bohn, 2013), (Bryans et al., 2014), (Hause, 2014)	++	++	-	+	-	-
Bigraphs (Stary and Wachholder, 2016)	+++	++	+	+	-	-
(Gassara et al.,2017)	+++	++	-	+	-	-
VDM-RT (Nielsen and Larsen, 2012)	+	++	-	+++	-	+
CML (Woodcock et al., 2012)	-	+	-	+	-	-
SOA (Derhamy et al., 2019)	+	++	+	+	-	-
CPN (Akhtar et al., 2019)	+	+	-	++	+	+
Ontologies (Nilsson et al., 2020)	+	++	-	+	-	-
ADL						
π-Calculus for SoS (Oquendo and Legay, 2015), (Oquendo, 2016)	-	+++	++	++	++	-
SoSADL – DEVS (Neto, 2016)	+	+++	+	++	-	-
Sos-ADL (Silva et al., 2020)	-	+++	+++	+	++	+
Multi-labeled graphs ADL (Chaabane et al., 2019)	++	+++	-	++	-	-

$^{+++}$: ***Highly Supported,*** $^{++}$: ***Supported,*** $^{+}$: ***Partially Supported,*** **-** : ***Not Supported***

Authors in (Akhtar et al., 2019) proposed a formal architecture approach based on Colored Petri Nets (CPN) as a formal modelling tool. They specify some behavioural properties concerning a system's safety, and verify them using Labelled Transition System (LTS) as a model-checking tool. The approach was applied to a Smart Flood Monitoring SoS. Similarly, authors in (Nilsson et al., 2020) specified an architectural framework based on an ontology that uses the Object Process Methodology (OPM) ISO 19450. They model SoS-related concepts and relations along them to streamline collaboration among involved SoS stakeholders.

Some dedicated approaches were based on ADLs to deal with few challenges on the design level of SoS, reducing their complexity (Guessi, 2015). For instance, (Neto, 2016) gave DEVS-based notations and functions as a simulation model for SoS behaviours. In the same thought, (Oquendo 2016) used the "π-Calculus for SoS" (Oquendo and Legay, 2015) to describe an ADL for SoS, focusing on the control

of interactions between systems through the usage of mediators and bindings. It has been extended through (Silva, E et al., 2020) by introducing a verification method using the mission modeling language mKAOS and the DynBLTL formalism to verify mission-related properties.

Authors in (Chaabane et al, 2019) extended the ISO/IEC/IEEE 42010 standard (ISO/IEC/IEEE, 2011) to express SoS characteristics by proposing an ADL based on multi-labeled graphs where Goal-Question-Metric (GQM) was used to evaluate the effectiveness of the proposed models.

By analyzing the previous related works, it is worth to notice that the hierarchical aspect is well defined especially in modelling approaches, such as SySML and Bigraphs. Unfortunately, a formal verification of SoS is not considered. It is also pointed that only few of these existing approaches deal with the semantical behaviour description, particularly no attention is paid to formalize behavioural constraints of SoS, nor their model execution and formal analysis.

3 APPROACH PRINCIPLE

For more details about the most useful concepts of the BRS and Maude language, involved in the process of defining ArchSoS, the reader may see (Milner, 2001) and (Clavel et al., 2007).

The process of defining ArchSoS is divided into two steps:

First, we give ArchSoS a formal syntax, inspired from BRS concepts which we extend to be able to represent the explicit roles of each sub-systems, as well as specific link points. Thus, SoS models in ArchSoS may be represented graphically or algebraically.

In the second step, we define an operational semantic for ArchSoS, decorated by behavioural constraints controlling a system evolution. SoS may evolve using a Strategy, which is a state evolution of a SoS using a specific action, and controlled by a specific predicate stating whether the application of the action is allowed. ArchSoS semantic is implemented in Maude language, offering enough expressiveness and simulation with its executable rewriting logic. Maude is equipped with a Model-Checker, allowing the analysis of qualitative properties regarding a SoS behaviour and evolutions, as for instance, the mission consistency of a given SoS.

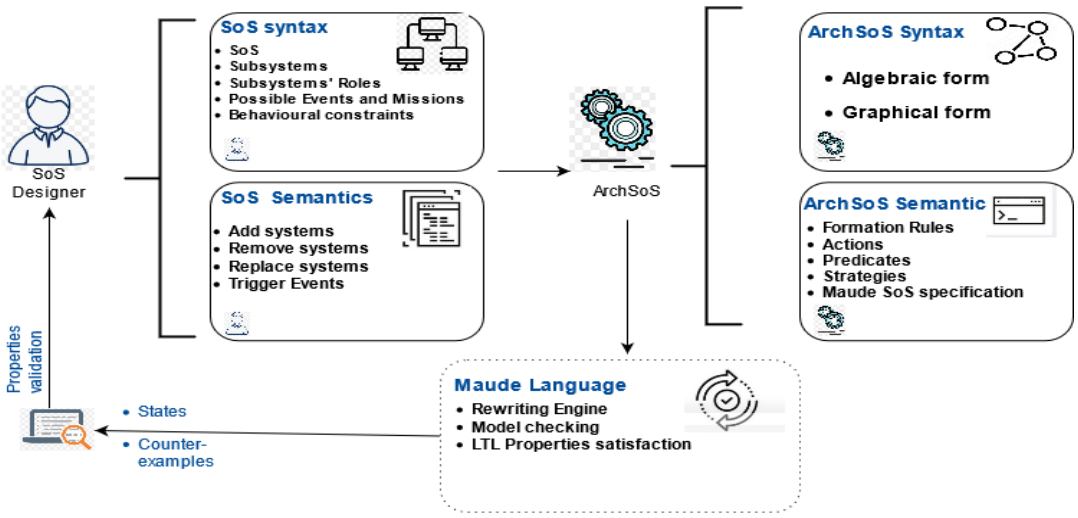

Figure 1. ArchSoS definition process.

Figure 1 sums up the principle of our ArchSoS formal description language. A designer initializes a SoS description by defining a hierarchical structure of a SoS, its constituents, their roles, the possible events/missions, and a set of behavioural constraints. This hierarchical structure of SoS is specified using both algebraic graphical forms inspired from Bigraph notations. A designer may simulate by adding, removing and replacing constituent systems, and may trigger events that occur in a SoS. ArchSoS semantic description is equipped with a set of actions affecting SoS evolution inspired from BRS reactive rules. For instance, constituent systems may be linked, according to a specific event triggered by the system designer, and their roles may be combined to achieve a new SoS role, thus achieving SoS missions. In order to guide the application of these actions, they are equipped with a set of predicates expressing application condition, creating a strategy rule that reduces the non-determinism in SoS behaviour evolution. Maude language enables the specification of a formal syntax as well as a semantic for SoS. Through its rewriting engine, the designer may simulate this specification and analyze it. ArchSoS properties are specified using LTL temporal logic and their satisfaction is achieved using Maude's inbuilt Model-Checker.

4 ARCHSOS CONCRETE SYNTAX

ArchSoS syntax is inspired from Bigraphs, allowing both an algebraic and a graphical description of its concepts. In this paper, we extend Bigraphs by making the ports' notation explicit: using the operator {…} that identifies the ports names in the algebraic and graphical forms.

4.1 Algebraic Description

A SoS is defined as a node of a SoS sort, nesting different systems: either other SoS or sub-systems. Sub-systems are also nodes of sub-system sorts that are nested inside a SoS node. Inner names represent events that may occur, affecting the constituent systems of a SoS. To each SoS or sub-system is attached at least two different ports: **1) R port(s)**: describes the role of a system, sub-systems have pre-defined roles describing their functionalities, while SoS roles are explicit only when they are linked to at least two other sub-system (or sub-SoS) ports, thus expressing an emergent behavior for a SoS that is only active when it emerges. **2) L port(s):** may be attached to an event and to another L port via an hyper-edge that indicates the nature of this link: 'e' hyper-edge: data-exchange, 'a' hype-edge: authority, 'u' hyper-edge: usage link. It allows a system to be linked to another system according to a specific event,

Definition 1. *The algebraic syntax of a SoS in ArchSoS is recursively defined using Backus Naur form (*McCracken, 2003) by:

```
<SoS> ::= <Events> <id_SoS> "{" <id_Role> "," <id_Link> "}" "." "(" <SoS>  "|"
      <SoS> ")"|<Events> <id_SoS> "{" <id_Role> "," <id_Link> "}" "." "(" <SoS>
      ")" |<Events> <id_SoS> "{" <id_Role> "," <id_Link> "}" "."<Sub_systems> ")"

<Sub_systems> ::= <id_Subsystem> "{" <Roles> "," <id_link> "}" | <id_Subsystem> "{"
      <Roles> "," <id_link> "}" "|" <sub_systems>

<Roles> ::= <id_role> | <id_role> "," <Roles>
<Event> ::= "\" <id_event> | "\" <id_event>  <Event>
<id_event> ::= String <id_SoS> ::= String <id_Link> ::= String
<id_Role> ::= String <id_Subsystem> ::= String
```

We give the following notation of a SoS as an example:

$$\backslash Ei\ SoS\{R,\ L\}.(Si.\{Ri,\ Rj,\ Li\}|Sj.\{Rk,\ Rl,\ Lj\})$$

This indicates that a SoS (of name *SoS*), having the Role *R* and the link *L* (inside the operator *"{…}"*), contains two sub-systems *Si* and *Sj* via the nesting operator *"."*. They are separated via the juxtaposition operator *"|"*. They have the roles *Ri, Rj* and *Rk, Rl* respectively. *L, Li* and *Lj*

are the link ports of the SoS, and the sub-systems `Si` and `Sj` respectively. The SoS has an unlinked event `"Ei"`.

4.2 Graphical Description

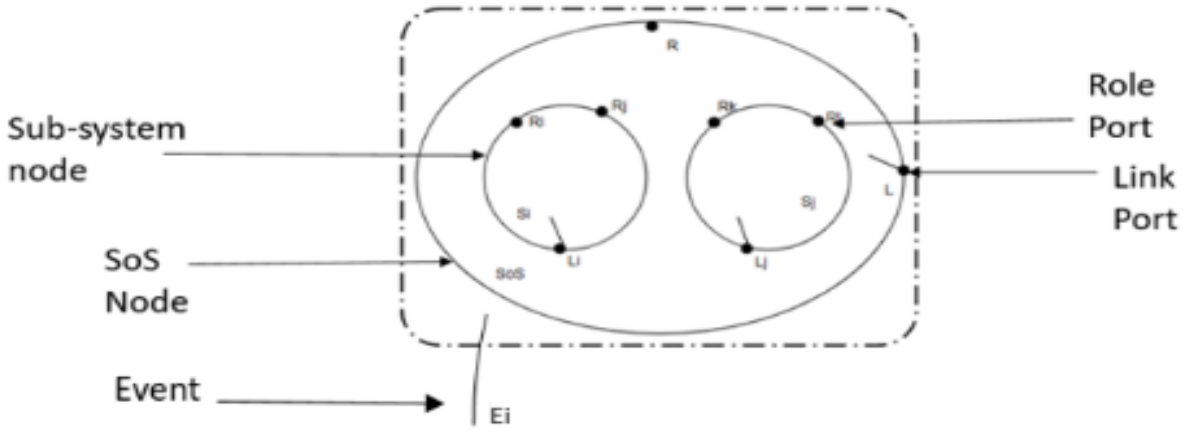

Figure 2 illustrates the graphical elements of ArchSoS syntax description, illustrated on the previous example noted: `\Ei SoS{R, L}.(Si.{Ri, Rj, Li}|Sj.{Rk, Rl, Lj})`. The SoS node in this example has two nested nodes Si and Sj, representing atomic sub-systems, but may as well nest other SoS nodes. This graphical description allows a better view of a SoS constituents and their hierarchy.

4.3 Behavioural constraints

ArchSoS is extended with a set of behavioural constraints, prohibiting certain illegal interconnections between systems. They represent various conditions that need to be satisfied when dealing with a SoS dynamics and evolution of its constituent systems. We may note that these constrains are defined by a SoS designer, as they differ from one SoS to another. They are divided into two type of constraints: (1) *Incompatible Links*: This indicates that some systems cannot communicate with each other. The constraint has the syntax: ∼Link (*Si*, *Sj*, a), it states that two systems *Si* and *Sj* cannot be linked with a link of type 'a'. (2) *Incompatible Roles*: represent the illegal role combinations between constituent systems, it is noted as: ∼ [(*Ri*: *Si*),(*Rj*: *Sj*)], it states that the roles *Ri* and *Rj*, corresponding to systems Si and Sj respectively, cannot be combined together.

4.4 Application Example

The Crisis Response System of System (CRSoS) is chosen as a case study to validate and illustrate our approach. It contains four different SoS, which are: Health, Firefighting, Surveillance and the Police SoS, equipped with their own sub-systems (For instance, Crisis detection system is a sub-system of the surveillance SoS). Each of them is, on one hand, operating for its own goals and purposes, and on the other hand interacting and communicating with other systems to achieve a common goal, thus creating new behaviours. CRSoS has many missions, but in this paper, we present only one, which is the *Fire-Distinguish* Mission. We have three possible events: Fire, FireSignal, and FireAppearance. Figure 3 illustrates the graphical description of CRSoS.

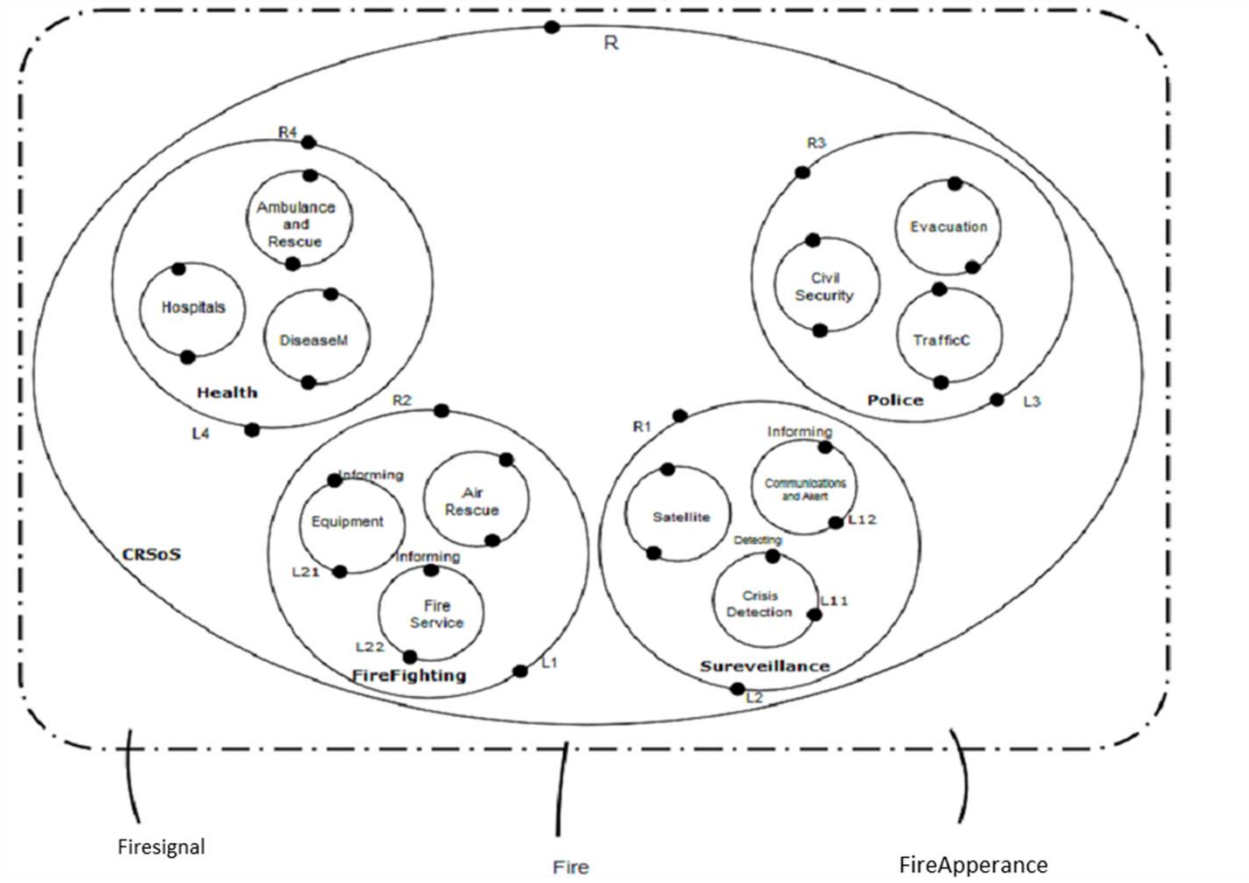

Figure 3. CRSoS Graphical description

Behavioural constraints for CRSoS system are shown in Table 2. Some roles cannot be combined and specific systems cannot be linked together with a specific link type. For instance, *~Link(Surveillance,*

Health, a) indicates that the Surveillance sub-system cannot have an authority over the health sub-system in CRSoS, when a fire event occurs.

Table 2. CRSoS Behavioural constraints

Constraint	Conditions
Incompatible Links	∼Link(Surveillance, Health, Fire, a) ∼Link(Surveillance, FireFighting, Fire, a) ∼Link(Surveillance, Health, Fire, u)
Incompatible Roles	∼ [(Equipment : Equipment and logistics), (detection : Crisis Detection)] ∼ [(DiseaseM : DiseaseManagement),(TrafficC: TrafficC)]

5 ARCHSOS OPERATIONAL SEMANTICS

In this section, we describe SoS behaviour modelling in ArchSoS. We define a SoS state and the appropriate semantics to specify its transition from one state to another.

Definition 2. *The state of a SoS* `ST` *is defined as the tuple:*

`ST = (id_SoS, E)`, *where:* `id_SoS` *is the identification of a SoS,* `E` *is an event that affects a SoS and its constituent systems.*

For example, a state for CRSoS affected by a *Fire* event is defined as `ST = (CRSoS, Fire)`.

5.1 Evolution Actions

A SoS behaviour evolution is expressed by its transition from one state to another according to a certain event. The event provokes a link between constituent systems, enabling them to combine their specific roles to achieve SoS missions. This is done through a set of actions, serving as a transition mean for SoS states. Actions represent the ArchSoS execution of the BRS reaction rules; they include the Bigraphical extension of the explicit ports. An action may contain a set of parameters `pr1, pr2,…, prn` carrying values used in the action act. Each parameter can be considered as a variable.

Definition 3. *The behaviour evolution of a SoS is defined by the following rule when applying an action* `A: A(pr1, pr2,…, prn) : ST → ST', where:`
`A` *may be of three types (see Table 3): 1) Link Actions:* Create/destroy links between SoS constituents. *2) Role Actions:* Combining (Deleting) roles of SoS constituents to emerge (delete) a specific SoS role. *3) Configuration Actions:* To add (respectively Remove or Replace) SoS constituent systems.

Table 3 summarizes the set of actions that can be applied to a SoS example to capture its evolution. For instance, a SoS that acquires a new role is defined through the acquire role action on the table: Both systems *Si* and *Sj* are already linked with a link of type '*e' (*hyper-edge *eij)* directed from *Si* to *Sj*, indicating that *Si* is exchanging data with *Sj*. This link occurred due to an event *Ei*.

Action Type	Action	Description	Algebraic form
Links Actions	Link System *Link(Ei, e, Si → Sj)*	Creates a link between two sub-systems *Si* and *Sj*, by linking their *Li* and *Lj* ports respectively with the event *Ei* (*Li_{Ei}* and *Lj_{Ei}*), the link is stated by the hyper-edge *eij*	*\Ei SoS{R, L}.(Si{Ri, Rj, Li}|Sj{Rk, Rl, Lj})* → *SoS{R, L}.(Si{Ri, Rj, Li_{Ei}(eij)}|Sj{Rk, Rl, Lj_{Ei}(eij)})*
	Destroy Link *DL(Ei, e, Si → Sj)*	Removes a link between the two-systems *Si* and *Sj*, by deleting the event *Ei* and the hyper-edge *eij* linking them	*SoS{R, L}.(Si{Ri, Rj, Li_{Ei}(eij)}|Sj{Rk, Rl, Lj_{Ei}(eij)})* → *\Ei SoS{R, L}.(Si{Ri, Rj, Li}|Sj{Rk, Rl, Lj})*
Roles Actions	Acquire Role *ARole(Ri: Si,Rk:Sj)*	The '*SoS*' role becomes active via the hyper-edge *x*, as roles *Ri* and *Rk*, belonging to the sub-systems *Si* and *Sj* respectively, are combined.	*SoS{R, L}.(Si{Ri, Rj, Li_{Ei}(eij)}|Sj{Rk, Rl, Lj_{Ei}(eij)})* → *SoS{R(x), L}.(Si{Ri(x), Rj, Li_{Ei}(eij)}|Sj{Rk(x), Rl, Lj_{Ei}(eij)})*
	Delete Role *DRole(Ri: Si,Rk:Sj, x)*	Removes the active role of a SoS by removing the hyper-edge *x* linking its role with the sub-systems roles *Ri* and *Rk*	*SoS{R(x), L}.(Si{Ri(x), Rj, Li_{Ei}(eij)}|Sj{Rk(x), Rl, Lj_{Ei}(eij)})* → *SoS{R, L}.(Si{Ri, Rj, Li(eij)}|Sj{Rk, Rl, Lj(eij)})*
Configuration Actions	Add subsystem *AddS(Sk, SoS)*	Adds a sub-system *Sk* to a specific SoS, this action is needed when all systems cannot be linked for example	*\Ei SoS{R, L}.(Si{Ri, Rj, Li}|Sj{Rk, Rl, Lj})* → *\Ei SoS{R, L}.(Si{Ri, Rj, Li}|Sj{Rk, Rl, Lj}|Sk{Rm, Lk})*
	Remove sub-system *Remove(Sk ,SoS)*	Removes a sub-system *Sj* from a SoS, when the sub-system is not needed (cannot be linked to any of the other sub-systems for instance)	*\Ei SoS{R, L}.(Si{Ri, Rj, Li}|Sj{Rk, Rl, Lj}|Sk{Rm, Lk})* → *\Ei SoS{R, L}.(Si{Ri, Rj, Li}|Sk{Rm, Lk})*
	Replace sub-system *Replace(Sj,Sk,SoS)*	When two systems have incompatible links, this action replaces a sub-system *Sj* with the sub-system *Sk* in order to have compatible links in *SoS*.	*SoS{R, L}.(Si{Ri, Rj, Li_{Ei}(eij)}|Sj{Rk, Rl, Lj_{Ei}(eij)})* → *SoS{R, L}.(Si{Ri, Rj, Li_{Ei}(eik)}|Sk{Rm, Lk_{Ei}(eik)})*

Table 3. Evolution Actions description

By applying this action, they have their roles combined (*Ri* and *Rk* in this case) into the SoS Role *R*, through an hyper-edge '*x*', defined on the right side of the action.

5.2 Formation Rules

To guide and constraint the creation and evolution of ArchSoS models, formation rules are defined and have to be respected when designing an SoS using the ArchSoS model. Table 4 details the conditions of formation rules Ci for each specific node. Systems created using the ArchSoS, as well as the

application of the defined actions on these systems are correct-by-definition, since they respect the structure of the Bigraphical model.

Table 4. Formation rule conditions for ArchSoS' Architecture

Condition	Description
C0	All children of a SoS node have a sort SoS, or a Sort Sub-system
C1	All SoS and Sub-system nodes are active
C2	In a SoS node, the R port is always linked to at least two of its Sub-systems' R nodes
C3	In a Sub-system node, the R port is always linked to both another sub-system's R port, and to a SoS' R node
C4	In a Sub-system node, the L port is always linked to both an inner name and to another L port of a Sub-system node
C5	In a SoS node, the L port is always linked to both an inner name and to another L port of another SoS node

5.3 SoS Evolution Scenario

We propose an execution scenario to illustrate the application of the proposed actions. We consider the mission *FireDistinguish* of CRSoS. For a SoS state evolution example, we may identify a possible achieved mission in a distinguished state ST as: (Ei, Li, Ri), where: Ei is an *event* affecting a SoS, Li is a link connecting the systems as a response to the event Ei, Ri represent the combined roles belonging to the linked systems.

Table 5 illustrates the missions, events, links and role combinations of this scenario. R is the CRSoS role resulting from combining R1 and R2, which are the active roles of the *Surveillance* and *FireFighting* SoS respectively. The resulted Bigraph after applying the above action on the scenario is shown on Figure 4. We note that this scenario does not violate the behavioural constraints of CRSoS, and permits to achieve the *FireDistinguish* mission.

Table 5. CRSoS execution scenario

Mission	Event	Links	Role combinations
FireAlert (Surveillance)	*FireAppearance*	*Link(FireAppearance : e, CrisisDetection, CommunicationsandAlert)*	R1 : (Surveillance SoS) [*(CrisisDetection:detection), (Communications and Alert :communication)]*
FieResponse (Firefighting*)*	*FireSignal*	*Link*(*FireSignal* : *u, FireService, Equipement*)	R2 : (FireFighting SoS) [*(FireService:Ffightin g), (Equipement and logistics:equipement)*]
FireDistinguish (CRSoS)	*Fire*	*Link(Fire : e, Surveillance, FireFighting)*	R : CRSoS [*(Surveillance:R1), (FireFighting:R2)*]

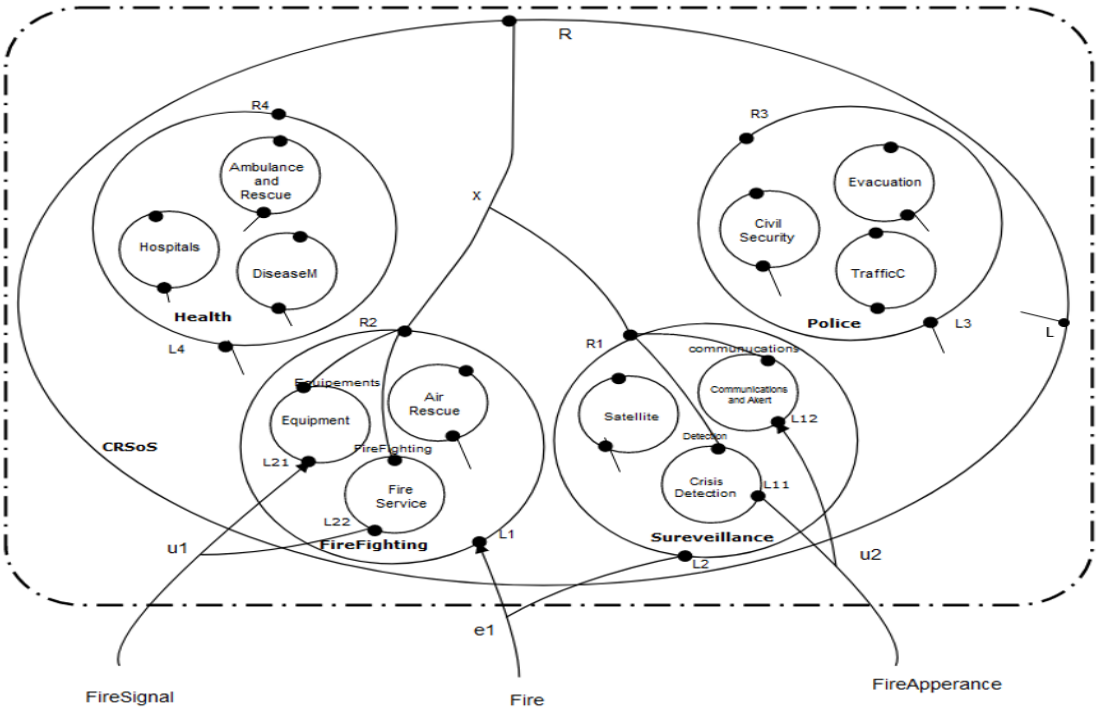

Figure 4. CRSoS final state of the scenario example

Through the previous scenario, we show how we may apply different actions resulting in different behaviours; we notice that the CRSoS acquire new roles, which allows it to potentially achieve missions in response to specific events. We note that CRSoS syntax, alongside the actions applications, respect the defined formation rules, as well as the given behavioural constraints.

5.4 Strategy Based Evolution

In order to reduce and guide the non-deterministic behaviours evolution of SoS in ArchSoS, we define Strategy rules which associate to each action type, a set of predicates $\varphi_1 ... \varphi_n$. Formally, this will be defined as follows:

Definition 4. *We define a SoS Strategy Rule SR as a guided evolution of a SoS:*

$(A_i, \phi_{Ai}) : ST \rightarrow ST' \text{ if } \phi_{Ai}$, *ST is the state of a SoS,* A_i *is the applied action,* ST' *is the resulting state, if the predicate* φ_{Ai} *holds.*

Table 6 summarizes the main ArchSoS evolution control predicates. Each φ_i (i=1...9) constrains the application of a corresponding action (or a set of actions). For instance, φ_4 checks if whether a specific system is linked or not. We note that the negation of the predicates $\varphi 2$ and $\varphi 6$ allows to represent the behavioural constraints defined previously (incompatible roles / incompatible links).

The main contribution of this strategy-based evolution in ArchSoS is that it allows a SoS to evolve naturally, in order to achieve some missions, according to specific events. This evolution is represented by a state transition, guided by a Strategy.

Table 6. Evolution control predicates in ArchSoS

Predicates	**Description**	**Associated actions**
φ1	A SoS role is active	Delete role, delete link, Acquire role
φ2	Roles are compatible	Acquire role
φ3	All roles are not compatible	Replace System
φ4	A system is linked	Link system, remove link, remove system
φ5	Two systems are linked together	Remove link , Link system
φ6	A link is possible	Link system
φ7	All links are not compatible	Add system
φ8	A Sub-system can be added	Add system
φ9	A Sub-system can be removed	Remove system

6 MAUDE BASED EXECUTION SEMANTICS

We choose to give a formal semantic to our proposed SoS language, based on Maude, which emphasizes ease of specifying distributed systems. It provides a large range of analysis techniques. Thus, we proceed to execute and simulate our ArchSoS specifications on Maude system and to gain more insurance about a SoS, we use Linear Temporal Logic (LTL) model-checking to verify SoS behaviours, and the consistency of SoS missions, through qualitative verification .

6.1 Maude Encoding

Specifically in this paper, the ArchSoS operational semantic is defined by using three distinct modules, offering a clear separation between structural and behavioural aspects of our ArchSoS language. The three modules permit the specification, modelling, and execution of an ArchSoS semantic respectively. For each element in ArchSoS, we associate a corresponding Maude definition. Table 7 summarizes the encoding of ArchSoS syntax into Maude. Figure 5 illustrates a step-by-step view of our Maude-based specification. It states the three modules that are detailed further below.

Functional module *ArchSoSSyntax:* A functional module that specifies the structure of a system in an equational theory, in terms of constructs types (sorts, operators etc.) for each ArchSoS element; these elements are built according to a constructor (defined by the value *ctor*). LinkT is the sort that defines the three pre-definite types corresponding to our link types *a*, *u* and e.

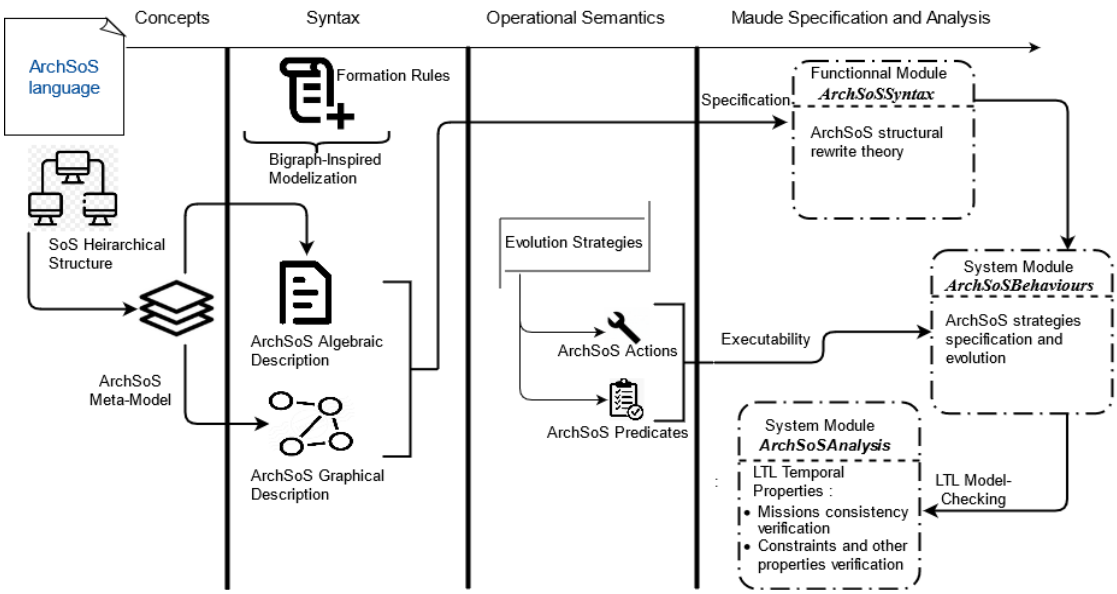

Figure 5. Step-by-Step View of the Maude-base specification and analysis of ArchSoS

Table 7. Integration of ArchSoS concepts in Maude

	ArchSoS concept	**Maude definition**
Syntax	SoS	*sort SoS idSoS.* *op opSoS <_,_>.P[_].L[_].E[_] :* *IdSoS Role SubSyS Link Event ->SoS [ctor] .* *op null : → SoS [ctor] .* *op _|_ : SoS SoS →SoS [ctor assoc comm id: null] .*
	Sub-System	*subsort subSyS <SoS .* *sort IdSubsystem .* *op opsubSyS<_,_>.L[_] : IdSubsystem Role Link → subSyS [ctor] .* *op _|_ : subSyS subSyS → subSyS [ctor assoc comm id: null] .*
	Role	*sort Role .* *op "RoleName" ->Role [ctor].* *op null : →Role [ctor] .* *op _+_ : Role Role →Role [ctor assoc comm id: null] .*
	Links	*sorts Link LinkT.* *op link<_:_; _ →_ > : Event LinkT IdSoS IdSoS ->Link [ctor] .* *ops a, u, e : →LinkT [ctor].*
	Events	Sort inside the opSoS construct.
Behaviour	Action Ai	*Maude's Rewrite Rule*
	State	*SoS sort with an active event*
	SRi Rule	*Maude's Conditioned Rewrite Rule* *crl [rewrite-rule-name] : State ->State' if* φ_i .
	Control Predicates φ_i	*Maude's Conditional Equations:* *IsActive(SoS), CompRoles(*$Role_i$*,* $Role_j$*), NoComp(SoS)* *isLinked(*$Subsystem_i$*), areLinked(*$Subsystem_i$*,* $Subsystem_j$*),* *PosLink(* $Subsystem_i$*,*$Subsystem_j$*, Event, Linktype),* *NoClinks(Si), CanAdd(SoS,SubSystemi*$_i$*) CanRemove(SoS,SubSystemi*$_i$*)*
	Properties	*LTL Properties: Link Properties, Reconfiguration Properties, Roles Properties, Mission Property*

For instance: the SoS operator is defined as *'op opSoS< _, _>.P[_].L[_].E[_])'* where : the SoS *name* (defined by the sort IdSoS) is the first identifier, followed by an attribute that specifies it's role. The P part represent the constituent systems of a SoS, and the L part is the links that a SoS shares with other, while E indicates the possible events.

System Module *ArchSoSBehaviours*: In this module, we identify: (1) a set of rewrite rules specifying the `Ai` actions dedicated to define the SoS behaviours in ArchSoS, these actions are conditioned by predicates to define a strategy rule. (2) SoS states as SoS sorts containing active events. Actions are applied to these states. (3) A set of conditional equations to define the φ_i control predicates.

As stated in table 7, the *isActive* predicate φ_1 for instance is defined as:

```
ceq isActive(opSoS < Sidi, Ri >.P[Si].L[Li].E[Ei]) = false if Ri == null.
```

Where *Sidi* is an SoS id, *Ri* it's roles, *Si* its constituent system(s), *Li* its current links, and *Ei* the current event, the equation returns false if the *Ri* role is null, meaning that a SoS is inactive at this state.

```
crl [ A1 ] :
opSoS< CrisisControl , null >.P[
 opSoS< Surveillance ,  null >.P[
   opsubSyS< CAlert , communications >.L[ null ]| opsubSyS< CDetection , detection >
   .L[ null ] ].L[ null ].E[ FireAppearance ] |
 opSoS< FireFighting , null >.P[
   opsubSyS< EquipementsL , equipements >.L[ null ] | opsubSyS< FireS , Ffighting >
   .L[ null ] ].L[ null ].E[ FireSignal ] |
 opSoS< Police , null >.P[
   opsubSyS< TrafficC ,  TControl >.L[ null ] | opsubSyS< EvacuationCC , CManagement >
   .L[ null ] ].L[ null ].E[ EarthQuakeSignal ]
].L[ null ].E[Fire ]
=>
opSoS< CrisisControl , null >.P[
 opSoS< Surveillance ,  null >.P[
   opsubSyS< CAlert , communications >.L[ link< FireAppearance  : e ;  CAlert --> CDetection > ] |
   opsubSyS< CDetection , detection >.L[ link< FireAppearance  : e ;  CAlert --> CDetection >] ]
   .L[ null ].E[ FireAppearance ] |
 opSoS< FireFighting , null >.P[
   opsubSyS< EquipementsL , equipements >.L[ null ]  | opsubSyS< FireS , Ffighting >
   .L[ null ] ].L[ null ].E[ FireSignal ] |
 opSoS< Police , null >.P[
   opsubSyS< TrafficC ,  TControl >.L[ null ]  | opsubSyS< EvacuationCC , CManagement >
   .L[ null ] ].L[ null ].E[ EarthQuakeSignal ]  ]
 .L[ null ].E[Fire]
    if ( ( isLinked(opsubSyS< CAlert , communications >.L[ null ] ) == false
    and isLinked( opsubSyS< CDetection , detection >.L[ null ] ) == false )
    and PosLink(opsubSyS< CAlert , communications >.L[ null ],
    opsubSyS< CDetection , detection >.L[ null ], FireAppearance  , e ) == true ) .
```

Figure 6. Strategy rule of the Link System action

The action illustrated in Figure 6 represents a link action rule on CRSoS, alongside its controlling predicates. It defines a Strategy rule for ArchSoS semantics, evolving a SoS from one state to another.

System Module *ArchSoSAnalysis*: introduced to specify LTL properties in order to verify that ArchSoS desirable behaviours are achieved, and that no behavioural constraints are violated, the properties include:

- *Link Properties (Plink, Dlink)*: They affect links. They check if links are possible, and if links can be destroyed respectively.
- *Roles Properties (ARoles, RRoles):* They affect roles by checking that only compatible roles are combined, and if they can be removed.
- *Reconfiguring Properties* (*RecReplace*, *RecA/R*): Properties checking that the execution does not terminate in a deadlock situation by replacing, adding/removing systems.
- *Mission property* (*Consistency*): It verifies if a SoS may reach a state where a mission is achieved, it checks the missions consistency.

6.2 Formal Analysis

The major benefit of adopting Maude as a semantical framework for ArchSoS is the exploitation of its rewriting engine and its LTL model-checker. The execution of the ArchSoS analysis module permits to check that our ArchSoS model satisfies some SoS inherent properties. Properties in this module are defined as LTL formulas. LTL have many symbols and operators, which may be found in [Rozier, 2011]. We may note that $\Rightarrow$ stands for an implication, |= defines a property validation, while $\wedge$, $\vee$ *and* $\sim$ express Conjunction, Disjunction and Negation respectively. As for the operators *O*, $\Diamond$ and □, they represent the next state, an eventual future state, and a globally/always state respectively. These formulas are defined as symbolic properties that check whether a state transition is coherent, and that we only get desired behaviours from a SoS transition. They are verified through a Maude Model-checking. Each property depends on a set of predicates, to which is associated a symbolic property $p_1 \ldots p_9$, defined as a conditional equation. For instance, ϕ_1 is encoded as the property: *ceq Si* |= *p*1 = *true if isActive*(*Si*) == *true*. Each ArchSoS property formula is specified via a Maude equation, for instance:

`Plink:` expresses that when a SoS is in a state when not all constituent systems cannot be linked (~ p7), and (conjunction symbol ∧) there is a possible link between two systems (p6), and if two systems are not

linked (~ p5), eventually (□) we will have a state where these two systems are linked (p5). It is noted as:

`eq PLink = □ ( ~ p7 ∧ p6 ∧ ~ p5 ⇒ □ p5 ).` Similarely, we define the rest of the formulas:

Dlink: `eq DLink = □ ( ~ p1 ∧ p4 ∧ p5 ⇒ O ~ p4 ∧ ~ p5 ) .`

ARoles: `eq ARoles = □ ( ~ p1 ∧ p2 ∧ ~ p3⇒ □ p1 ) .`

RRoles: `eq RRoles = □ ( p1 ⇒ O ~ p1 ) .`

RecReplace: `eq RecReplace = [] (~ p1 ∧ p3 ∧ ⇒ O ~ p3).`

RecA/R: `eq RecA/R = □ ( p7 ∧ ( p8 ∨ p9 ) ⇒ □ ~ p7 ).`

Consistency: checks if we may have a state where the corresponding tuple (event, links, and roles) of a mission holds. `eq Consistency = □ (~ p1 ⇒ □ p1 ∧ Mission).`

To validate the consistency property, we have to define a new property for each mission. For example, a property called `FireDistinguish`, to define the *FireDistinguish* Mission (in Section 6.3) is shown on Figure 7. By replacing `Mission` with the `FireDistinguish` in the `Consistency` property, we get the following property:

`eq Consistency = □ ( ~ p1 ⇒ □ p1 ∧ FireDistinguish).`

```
ceq opSoS< Sidi , Ri >.P[opsubSyS< Syidi , Rj >.L[ Lj ]
| opsubSyS< Syidj , Rk >.L[Lk] | Sm ].L[Li].E[Ei]
|=  FireDistinguish = true if
(Ri == detection + communications + equipements + Ffighting )
 and (Ei == Fire ) and
(Syidi == Surveillance ) and
(Syidj == Firefighting) and (areLinked <Surveillance, Firefighting, Fire, e)) == true ) .
```

Figure 7. FireDistinguish property definition

Then, we apply the model Checker on the initial state of the CRSoS (where CRSoS is inactive), to verify the `Consistency` property of the *FireDistinguish* mission. The resulting execution is shown on Figure 8, it returns true, stating that `Consistency` property is satisfied, and that we eventually have a state where CRSoS achieved the defined *FireDistinguish* mission.

```
reduce in ArchoSoSAnalysis : modelCheck(initial, []<> Consistency) .
rewrites: 74 in 15288604515ms cpu (2ms real) (0 rewrites/second)
result Bool: true

Maude>
```

Figure 8. Model-checking Result

We may note that the proposed analysis aims to execute and prototype the CRSoS system. More interesting properties related to SoS challenges may be demonstrated in future work.

7 CONCLUSION

In this paper, we have proposed ArchSoS, an ADL to specify and execute SoS software architectures. Its syntax notation has been inspired from Bigraphs, well appropriate when giving formal and visual modelling of the SoS constituents and their evolution. Moreover, we have defined an operational semantic for ArchSoS, by describing a SoS state evolution, a set of actions that may affect this state; and a number of predicates that constraint the application of these actions. The defined set of predicates and constraints gives a rise to the definition of a new type of transition rules called strategies rules, which guide the evolution of a SoS behaviour. ArchSoS semantics is encoded into Maude language, to enable its execution. We have defined various properties for our scenario example; they have been checked via LTL Model-Checking of Maude system. They check whether the behaviour of a SoS is correct, and whether ArchSoS behavioural constraints are respected. They also allow us to check if a SoS achieves desirable missions. As future work, we plan to develop a practical environment around ArchSoS, which will make it possible to identify the performance of the proposed approach and to reason on its soundness. Other case studies are also possible to better illustrate the contributions of this language.

REFERENCES

Akhtar, N., & Khan, S. (2019). Formal architecture and verification of a smart flood monitoring system-of-systems. *Int. Arab J. Inf. Technol.*, *16*(2), 211-216.

Bryans, J., Fitzgerald, J., Payne, R., Miyazawa, A., and Kristensen, K. (2014). Sysml contracts for systems of systems. In *2014 9th International Conference on System of Systems Engineering (SOSE)*, pages 73–78. IEEE.

Cavalcante, E. (2015). On the architecture-driven development of software-intensive systems-of-systems. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pages 899–902. IEEE Press.

Chaabane, M., Rodriguez, I. B., Colomo-Palacios, R., Gaaloul, W., & 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.

Clavel, M., Durn, F., Eker, S., Lincoln, P., Mart-Oliet, N., Meseguer, J., and Talcot, C. (2007) *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.

Dahmann, J., Markina-Khusid, A., Doren, A., Wheeler, T., Cotter, M., and Kelley, M. (2017). Sysml executable systems of system architecture definition: A working example. In *Systems Conference (SysCon), 2017 Annual IEEE International*, pages 1–6. IEEE.

Derhamy, H., Eliasson, J., & Delsing, J. (2019). System of system composition based on decentralized service-oriented architecture. *IEEE Systems Journal*, *13*(4), 3675-3686.

Gassara, A., Rodriguez, I. B., Jmaiel, M., and Drira, K. (2017). A bigraphical multi-scale modeling methodology for system of systems. *Computers & Electrical Engineering*, 58:113–125.

Guessi, M., Neto, V. V., Bianchi, T., Felizardo, K. R., Oquendo, F., and Nakagawa, E. Y. (2015). A systematic literature review on the description of software architectures for systems of systems. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1433–1440.

Hause, M. C. (2014). Sos for sos: A new paradigm for system of systems modeling. In *Aerospace Conference, 2014 IEEE*, pages 1–12. IEEE.

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), pp.1–46.

Lane, J. A. and Bohn, T. (2013). Using sysml modeling to understand and evolve systems of systems. *Systems Engineering*, 16(1):87–98.

Maier, M. W. (1996). Architecting principles for systemsof-systems. In *INCOSE International Symposium*, volume 6, pages 565–573. Wiley Online Library.

McCracken, D. D., & Reilly, E. D. (2003). Backus-naur form (bnf). In *Encyclopedia of Computer Science* (pp. 129-131).

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.: Bigraphical reactive systems. In: International Conference on Concurrency Theory. pp. 16{35. Springer (2001)

Neto, V. V. G. (2016). Validating emergent behaviours in systems-of-systems through model transformations. In *SRC@ MoDELS*.

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

Nilsson, R., Dori, D., Jayawant, Y., Petnga, L., Kohen, H., & Yokell, M. (2020, July). Towards an Ontology for Collaboration in System of Systems Context. In *INCOSE International Symposium* (Vol. 30, No. 1, pp. 666-679).

Oquendo, F. (2016). pi-calculus for SoS: A foundation for formally describing software-intensive systems-of-systems. *System of Systems Engineering Conference (SoSE), 11th*.

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

Rozier, K. Y. (2011). Linear temporal logic symbolic model checking. *Computer Science Review*, 5(2):163–203.

Seghiri, A., Belala, F., Benzadri, Z., and Hameurlain, N. (2018). A maude based specification for sos architecture. In *2018 13th Annual Conference on System of Systems Engineering (SoSE)*, pages 45–52. IEEE.

Silva, E., Batista, T., & Oquendo, F. (2020). On the verification of mission-related properties in software-intensive systems-of-systems architectural design. *Science of Computer Programming*, *192*, 102425.

Stary, C. and Wachholder, D. (2016). System-of-systems supporta bigraph approach to interoperability and emergent behavior. *Data & Knowledge Engineering*, 105:155–172.

Woodcock, J., Cavalcanti, A., Fitzgerald, J., Larsen, P., Miyazawa, A., and Perry, S. (2012). Features of cml: A formal modelling language for systems of systems. In *System of Systems Engineering (SoSE), 2012 7th International Conference on*, pages 1–6. IEEE.