A Maude-based Rewriting Approach to Model and Control System-of-Systems' Resources Allocation

Charaf Eddine Dridi, Nabil Hameurlain, Faiza Belala

A Maude-based Rewriting Approach to Model and Control System-of-Systems' Resources Allocation

Charaf Eddine DRIDI[1,2], Nabil HAMEURLAIN[1], Faiza BELALA[2]

[1]LIUPPA Laboratory, University of Pau, Pau, France
{charaf-eddine.dridi, nabil.hameurlain}@univ-pau.fr
[2]LIRE Laboratory, Constantine 2 University – Abdelhamid Mehri, Constantine, Algeria
{charafeddine.dridi, faiza.belala}@univ-constantine2.dz

Abstract:
Systems-of-Systems (SoSs) are increasingly used to integrate and execute a large number of Constituent-Systems (CSs) to offer new functionalities that cannot be offered by its individual CSs. However, designing well-tuned directed SoS to deal with a variety of control issues such as resources management and temporal constraints violations while providing high-level assurance about their specified behavior is very challenging. Thus, due to the lack of explicit models for their resources allocations and quantitative features, SoSs executions are becoming more complex and cannot be effectively controlled. To solve these problems, a generic metamodel is proposed in this paper to control resources and behavioral features. The proposed approach deals with different resources of SoSs and provides control actions in order to manage their structural, temporal and behavioral aspects. The latter are grouped in a single model holding SoSs executions according to specific quantitative needs. One step further, the specifications of the proposed metamodel are encoded using Maude language that makes it possible to analyze various requirements needed by CSs, and enables an autonomic executability of their dynamic behaviors.

Keywords. SoS, Missions, Resources Allocation, Dynamic Behavior, Metamodeling, Maude.

1 Introduction

In the last decade, Systems-of-Systems (SoSs) appeared as new software technologies that integrate a set of various subsystems from different subfields, and which offer a reliable and more natural alternative to build an emerging vision for the next generation of large-scale systems [1] [2]. These systems are designed to integrate multiple independent and functional Constituent-Systems (CSs) into a larger system in several important application domains. Consequently, an SoS has the ability to offer new functionalities to users that cannot be offered by its individual CSs, but emerge from their combination.

Nowadays, critical and emergency SoSs, require the missions of various CSs to be accomplished in real-time and within specific amounts of time. Therefore, temporal properties are significant in such systems, where a violated constraint can result in a disaster. When referring to the period needed to execute a mission, *duration* is considered as a simple temporal constraint between the mission starting (resp. ending)

instants. Since we consider the duration of missions as a generalization of almost all other constraints[3][4][5], the challenge here is that the time constraints applied on missions can be implicitly affected by the physical environmental features of different constituents. These features are strongly related to resource properties. The resources that SoS requires at run-time, are sometimes limited, unlimited, renewable or not renewable [14][15][16]. As a solution, we use a set of features that specify different resources for local and global CSs. These features are the basis of developing a dependencies-based approach that has sets of attributes describing the status of the resources that the CSs may use concerning availability, consumption, (non)shareability, disruption, renewability, and withdrawal. Furthermore, these challenges are not easy to manage, i.e. independent evolution and dynamic resources allocation may cause these CSs to behave differently, for instance, they may affect the missions executions, the CSs interactions and communications within the SoS, and then they can cause the overall mission failure of the SoS.

In this paper, we provide a formal modeling approach that reduces the complexity of designing SoS temporal, resources and interaction behavior. We adopt an *MDA-based approach* to provide a *metamodel* for specifying resources properties, and temporal aspects of SoSs. We focus on defining the logical structure and behavior of qualitative and quantitative features involved in the SoS definition, which allows describing all the SoS features at the same high level of abstraction. Hence, to create this generic model, we have to consider a set of concepts, aspects, and features, i.e. *hierarchical composition of CSs*, *missions organization*, *roles interactions*, *temporal variations*, *resources allocation*, etc. Employing such a metamodel is a promising mean to show its ability to initiate our model into a wide range of specific technologies, and to provide more holistic solutions in SoSs applications.

To instantiate our metamodel, our choice was oriented towards the Maude language, thanks to its expressivity, it can execute, and validate all the relevant specifications of our proposed metamodel without losing information and features. Its executable semantics support and boolean expressions are relevant to design the states predicates of each CS. The language offers more accurate modeling of SoSs whose behavior depends on Missions quantitative time/resources. It uses the equations of rewriting theory to specify the data types of all components. It also offers a model-checker engine that provides a symbolic state-based verification of SoS properties. The proposed Maude-based SoS modeling approach is one of the effective solutions that enable the conditional rewriting rules to describe the unexpected behavior of SoSs. In this specification, we present a way to combine different Maude Modules at different entities levels to produce global knowledge about the entire SoS design.

2 Related Work

We represent in this section some relevant studies that analyze different SoSs aspects. Adopting a SysML visual modeling language, the authors of [6] [7] have proposed a semi-formal SoS conceptual model which serves as a domain-independent vocabulary for SoSs. However, their proposal does not provide a quantitative analysis method to exploit quantitative properties such as temporal constraints and resource allocation. Besides, no formal design to control roles (or missions) dynamic behavior is provided.

The authors of [8] [9] have introduced a formal approach based on Architectural Description Language, that inspires its syntax notation from Bigraphs to model hierarchical structures of CSs and their roles. They have focused on modeling the potential cooperation between CSs by offering a syntax-based description that manages a set of constrained events and roles links affecting the global mission of an SoS. Nevertheless, no attention is paid to qualitative aspects related to the execution time of missions accomplished by playing different roles, neither they considered the need of CSs for a global resources control to execute these missions. Besides, no execution control is presented to describe a logic that governs the execution of submissions.

The authors of [10] have proposed B3MS for SoSs modeling based on the formal technique of Bigraphical reactive systems with an inspiring vision from multi-scale modeling. They have given a method to address the dynamic aspect of SoS by providing model-based rules of basic reconfigurations. They have relied on bigraphical reaction rules to only express the different scales of configurations at the levels of composition, communications nodes, etc. and they have not paid attention to providing a resource controlling support nor a time-based qualitative evaluation of the introduced behaviors.

In [11], the authors have investigated the interplay between SoS and CSs architectures. The approach includes both the design of an SoS architecture that considers the architecture of an existing CS, and the architecting of CSs so that they can later become constituents of an SoS. However, they lacked concrete features that can govern CSs operating in an autonomic manner in a constrained environment. In addition, no formal executable support for their solution is provided as well.

The work presented in [12] have focused on developing a metamodel that represents SoS ontologies. It can be used for both modeling activities and ontology definitions. Even so, to support complete and systematic analysis and design of an SoS. However, the authors did not provide executable support to simulate and verify their proposed model. Additionally, no control of behavioral constraints is provided in their work.

The authors of [13] have presented a hybrid modeling method based on service-oriented and ontology-based requirements for SoSs modeling. Initially, they introduced an SOA conceptual model, then, rigorous modeling semantics through defining multiple ontologies which is important for domain knowledge reuse is defined. However, their model does not include supplement repositories to improve the efficiency of temporal behaviors e.g. services durations, and temporal constraints, and a validation method to ensure dynamic consistency with the needed resources.

We take advantage of some interesting coordination models presented in [14] [15] and [16] to build up our generic model for SoSs. The latter addresses the issue of conflicts over different types of resources and categorization during their consumption by missions or production by SoS roles. However, our proposed model is governed by a set of behavioral constraints controlling the execution. SoSs may evolve using transition systems, by using specific actions which are conditioned by specific predicates stating whether the application of the action is allowed or not. The metamodel semantic is implemented using Maude language, offering enough expressiveness and simulation with its executable rewriting logic.

3 A Metamodeling Approach for SoSs Architecture Description

In this section, we provide a generic integration model that introduces different features.

3.1 Principle: a generic metamodel for SoSs architecture

An SoS consists of a set of CSs that are playing different roles in an organizational and technical environment. These roles jointly realize a common goal. More precisely, an SoS model is composed of several missions or goals that execute together, to offer one global mission. These missions may need resources for execution such as humans, machines, services, etc. Moreover, SoSs can take four different types [2], *Directed* (with a central managed purpose and central ownership of all CSs), *Virtual* (lack of central purpose and central alignment), *Collaborative* (with voluntary interactions of CSs to achieve a goal), and *Acknowledged* (independent ownership of the CSs). Subsequently, the proposed definition is introduced based on the key consideration that SoSs are more than simply a set of connected CSs sharing data and offering missions, but, it defines the logical structure and behavior of qualitative/quantitative features involved in *Directed SoSs*. Therefore, the aim is to define a generic metamodel that will be used as a basis for the modeling and designing of these specific types, whose CSs can have their operational/managerial independence but their emergent behavior is subordinated to a specific mission and *Central Controller* of all CSs.

In this context, we are interested in modeling and specifying public resources (*PubRes*) which are shareable and consumed by missions within the SoS, and private resources (*PrvRes*) which are not shareable and consumed by missions within the CS. They could also be characterized by some other properties, i.e. limited, unlimited, renewable, and not renewable, as well as whether they are logical or physical resources. Therefore, Resources Management Controller consists of specifying the behavior to be adopted to manage the consumption and the production techniques in the SoS. It consists of a set of actions that are triggered in case the specified triggering conditions are fulfilled. The consumption relationships between missions and resources (resp. production relationships between roles and resources) in our specification are based on specific features that they have. We consider it as an SoS-specific refinement of model property-based resource consumption cycles introduced in[14][15], the proposed controller supports the consumption/production by the matching of specific Missions/Roles states with the features of the different resources. Other important characteristics that we should take into account are: Mission duration, Roles interactions, Desired/ Unwanted behavior, Composition, and CSs configuration [17].

The reason of proposing this metamodel is the need to cope with the variety of the features, and to offer a comprehensive specification to design any SoS architecture. Thus, the proposed metamodel is specified by enhancing the concepts with the essential quantitative features that an SoS should have. Therefore, we need to provide a generic metamodel for SoS design, to be able to show the ability of the metamodeling approach in instantiating architectural models of such systems and to explicitly represent all features. The specification level (see the five classes on top, red rectangle in Figure 1) concerns concepts related to the definition of behavior controllers (*ResMangCtrl* and *SoSCtrl*) that can manage the different states of *Resources*, *Missions* and *CSs*. Their

Precondition *Predicate* associations represent the states conditions that must hold before and be fulfilled after a transition *Action* of a *Resource*, a *Mission*, or a *CS*.

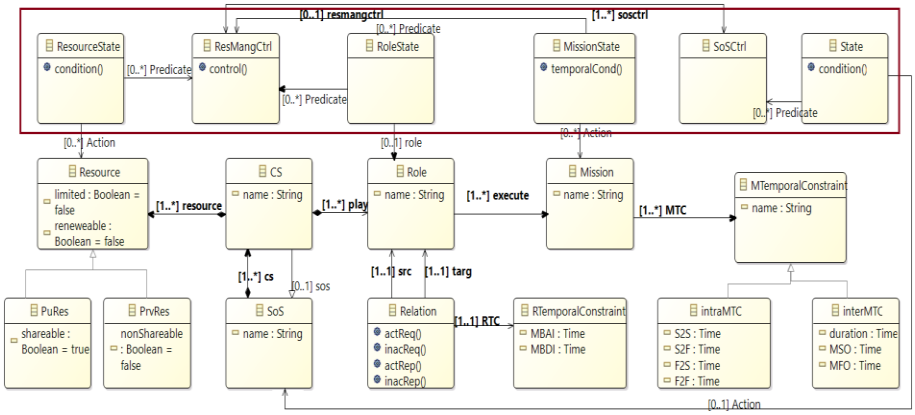


Fig. 1. General components of SoS metamodel.

3.2 Motivating example

An example of *EmergencyIncidentManagementSoS* [18], abbreviated *EIMSoS*, will be considered to concretize the basic contributions made by our approach. As an illustrative example of a critical SoS, consider a collection of autonomous and interacted CSs tasked with transporting and evacuating trauma patients and persons injured in emergencies. To accomplish this Mission, the considered *EIMSoS* must be designed in such a way that the CSs can perform some missions that cannot be provided by any CS. The visual representation of the EIMSoS elements is shown in Figure 2. A general idea of this SoS is that, after receiving an emergency call, the *CallCenterSoS*(*CCSoS*) which provides the *ComputerAidedDispatch*(*CAD*) as a public resource containing all the necessary information and the data of the incident, directly interacts with its two sub-call centers CSs i.e. *AmbulanceStation*(*ASCS*) and the *RetrievalServices* (*RSCS*). The three systems have access to shared resource *CAD* and then, they start switching between different roles (e.g. *Receiver* and/or *Dispatcher*) to accomplish a set of local missions, i.e. they discuss the available decisions and determine the optimal response after the emergency's call. If the decision is made to dispatch a *helicopter* and/or an *ambulance*, the *Dispatcher* will activate the role *Aero-Transport* and/or *GroundEmergency* of *AeroMedical-EmergencyCS* (*AMECS*) and *GroundEmergencyCS* (*GECS*) to start performing the corresponding missions, i.e. *preparing the medical teams*, *flight*, *departure*, *first aid*, and *evacuating*, etc. At this stage the *ambulance* and the *helicopter* are considered as *private (PrvRes)* and *limited resources* provided by *AMECS* and *GECS*, respectively. Upon arrival at the hospital, the *HospitalSoS*(*HSoS)* will interject and hands-over to the corresponding wards (e.g. *LabCS*, *RadioCS*, etc.) that can accomplish the necessary missions (e.g. *triage assessment*, *diagnosis*, *lab tests*, etc.) and take care of the patient.

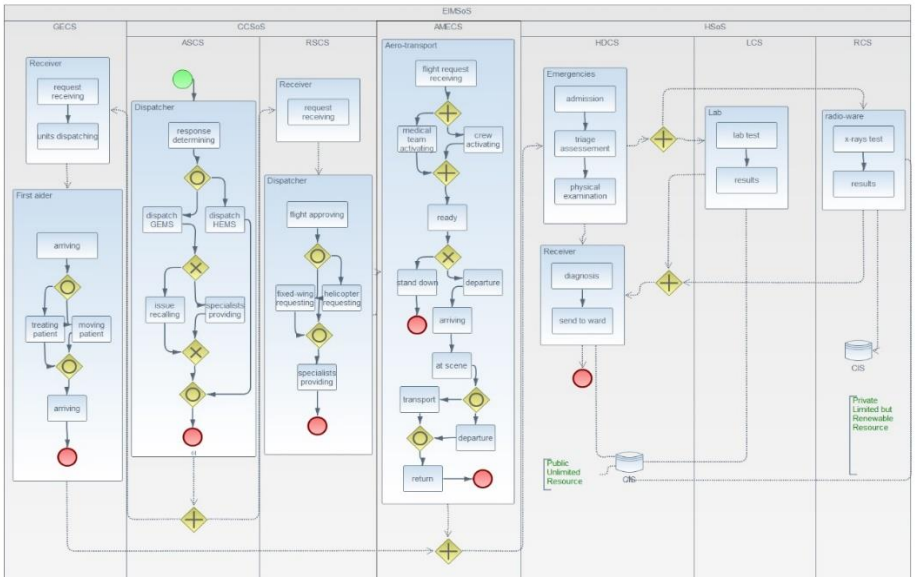


Fig. 2. BPMN Model for Emergency Incident Management SoS (EIMSoS).

4 An executable Maude-based specification of SoSs Metamodel

We have chosen the class diagram notation to represent graphically the proposed metamodel. UML Classes, aggregations, inheritance, etc. are used to represent elements and entities of this metamodel. In this section, we give an overview of our proposed object-oriented modules, and how they enable the implementation of the different entities and concepts mentioned in the previous metamodel; and how to model and control their emergent behavior. On the one hand, implementing these entities enables the designers to easily understand the SoSs architectures, including reasoning about their features by providing a high-level model of their structures. On the other hand, it enables them to instantiate it by specifying several mappings or transformations to deduce various executable models. To this end, we use Maude as a logical basis that can provide a clear definition of the object-oriented semantics and makes it a good choice for the formal specification in the form of a transition system.

The following subsections introduce the encoding of our metamodel in Maude. The latter is chosen as a formal specification language and verification platform, because it is expressive enough for specifying all the concepts, aspects and features of the metamodel. On the one hand, it gives a clear specification of the object-oriented notations classes in the metamodel (e.g. *SoS, CS, Mission, Roles, Ressource* classes, etc.), and describes all the relevant specifications without losing information and features. And on the other hand, it also offers several rewriting-based theories that are adequate for simulating the interdependencies (i.e. the *Predicate* and *Action* associations) between these classes by controlling the states of each *Mission, Resource* and *CS*, and offering more accurate modeling of SoSs whose behavior depends on Missions quantitative time/resources properties.

4.1 Motivating the use of Maude

The declarative language Maude [19] is a very expressive equational and rewriting logic language, as well as one of the most powerful languages in programming, executable formal specification, and formal analysis and verification. Its computation consists of a logical deduction by concurrent rewriting modulo the equational structural axioms of each theory. Maude is an object-oriented modeling language for the specification of distributed systems. Our specification is structured using two types of combined Maude modules, i.e. Real-Time and Object-Oriented ones. Each module consists of one specific class with a set of attributes, and it can import other specifications using that can provide the basic data types like objects, and configurations. What interests us is that Maude employs asynchronous message passing and, thus, the rewrite rules specify the possible transitions that may happen in different entities in SoSs. We specify the behavioral transitions using the following rewrite rules[19][20]:

crl[R] : {t} => {t'} in time u if (condition)

The triggering conditions of these rules "*R"* are state predicates of different entities in the metamodel and their triggered actions are encoded as Maude functional computation. These rules describe how to trigger an action, and how the corresponding control states are transited from one to another. It rewrites the left-hand term "*t"* of the rule into its right-hand term "*t'"* on which an action or a transition is applied. A conditional rewrite rule is triggered when its specified (condition) is satisfied. The idea is that in concurrent CSs forming an SoS, the concurrent states, which are called a configuration, consist of a multiset of objects and messages. Rewrite rules then define transitions between such configurations. These transitions represent the different actions that may occur in CSs. In our case, we present rules modeling the effects of time and resources on missions or the solicitations requests between roles.

4.2 SoS Global Structures and behaviors Modeling

In this section, we present Maude specification modules and classes that encode the different concepts mentioned in the metamodel (*Mission*, *Resource*, *Role*, etc.) including the necessary information to describe their *Predicates*, *Action*, *States*, and the overall behavior emerging from their relationships, structures and features.

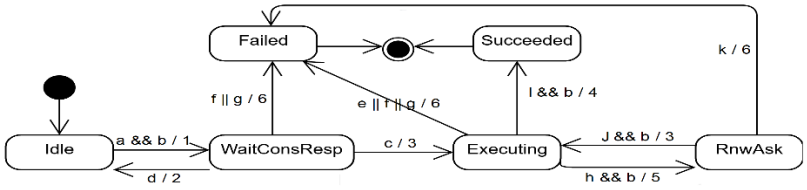
4.2.1 Modeling Missions and their durations:

Structure encoding: the Mission specification structure, the checking predicates, the temporal constraints and their violations signals are specified in the system module *MissionSpec*. The latter contains all the necessary declarations defining the class Mission, which has seven attributes, namely, the starting *sm* and the ending *em* instants of a Mission instance *m*, the allowed completion duration of the mission expressed as an interval *dn*, *dx*, the mission state during *st*, whether its temporal constraints are respected or not *sg*, and the required resources *rs* to finish the execution.

```
class Mission | sm : Nat, em : Nat, dn : Nat, dx : Nat, st : Stt, sg : Sig, rs : Rs
```

States and temporal constraints predicates encoding: we use (sort *Sig*) to model three signal events that are triggered to represent and check the duration constraints i.e. *isTCR(M)* is triggered to inform that there are no duration violations, *isMnV(M)* and *isMxV(M)* to indicate that minimum duration is violated (resp. maximum duration). Additionally, *Stt* attribute in class Mission is employed to specify different states of the

Mission instance (see Fig. 3). For convenience and simplicity, note that in this Fig. 3. and the upcoming statecharts, all transitions have a set of *Predicates* that are denoted by letters before the "/", and *Actions* that are denoted by numbers after it. Thus, we encode in Maude a set of predicates $P_{Mis}()$ that represent the conditions (left table) which must be true to trigger the transitions. These predicates will be employed by the *Resources Controller* to enable (or disable) the different actions $A_{Mis()}$(right table) in a given mission instance. One simple example of how this transition system works: consider a mission *M* waiting in *WaitConsResp* state for an authorization from the controller to consume *R*, (i.e. it has already sent to the controller a Consumption Request using the action *(1)* A_{Mis}). If the controller accepts the request, the predicate *(c)* P_{Mis} *= isConsReqAccepted (M, R)* becomes true and it will trigger the action *(3)* A_{Mis} *= execute(M)* to move the *M* from *WaitConsResp* state to *Executing* to state.



Mission's Predicates	*Description*
(a) P_{Mis}	isConsReqSent(M, R)
(b) P_{Mis}	isTCR(M)
(c) P_{Mis}	isConsReqAccepted (M, R)
(d) P_{Mis}	isConsReqRejected (M, R)
(e) P_{Mis}	isFailed(M)
(f) P_{Mis}	isMnV(M)
(g) P_{Mis}	isMxV(M)
(h) P_{Mis}	isAskRenewing(M, R)
(i) P_{Mis}	isFinished(M)
(j) P_{Mis}	isRenewOk(M)
(k) P_{Mis}	isRenewRejected(M, R)

Mission's Actions	*Description*
(1) A_{Mis}	sendConsReq(M, R)
(2) A_{Mis}	returnIdle(M)
(3) A_{Mis}	execute(M)
(4) A_{Mis}	exit(M)
(5) A_{Mis}	renew(M)
(6) A_{Mis}	cancel(M)

Fig. 3: Mission transition system.

Afterward, we use a set of Maude conditioned rules in combination with different monitoring and TC predicates to control and manage the current mission states, as well as its consumption requirements. These rules are reactive and take the form:

crl [name] : action (A_{Mis}) < Mid : Mission | state : S > => < Mid : Mission | state : S' > if (P_{Mis})

4.2.2 Modeling Resources Types

Structure encoding: using Maude specification, the resource specification in terms of structure, transitions predicates, and features are specified in the module *ResourceSpec*, it contains all the necessary declarations defining the class *Resource*. The latter has four attributes, namely, *prp*, *resSt*, *resT* and *resC* to keep track of, respectively, the resource property *prp* and its actual state *rSt*, whether it is a public or private resource via *rT* attribute and their category, i.e. (logical or physical).

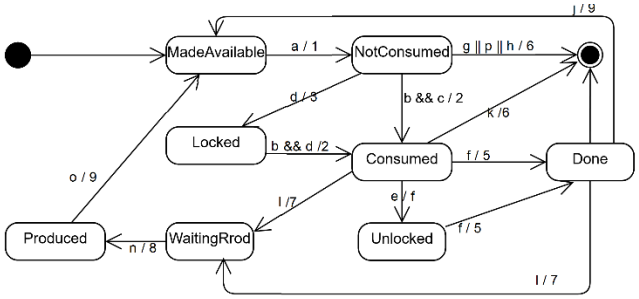
class Resource | prp : Prop, resSt : ResSt, resT : ResT, resC : ResC.

States predicates encoding: we use (*sorts Prop ResSt ResT*) to express their properties (*ul*: unlimited, *lim* : limited and *lr* : limited but renewable), their states (see Fig. 4.), if they were *PubRes* or *PrvRes*, and their *categories*. We identify a set of predicates $P_{Res}()$ and actions $A_{Res}()$ to specify the states and transitions of each resource in Maude. E.g. consider a resource R, available and holding in *NotConsumed*, the

controller will fire the transition "*b && c / 2"* and perform the action *(2)* A_{Res}*=consume(R)* to move to *Consumed* state whenever the two predicates *(b)*P_{Res} and *(c)*P_{Res} become true. These later indicate that there is a consumption approval and R is public. In case *R* is private the transition system will fire "*d / 3*", i.e. if *(d)*P_{Res}*=isPrvRes(R)* execute *(3)* A_{Res}*=lock(R)* and move to state *Locked.*

The Maude rules will be employed along with the *Controller rules* to manage the resource logical and physical behavior. These rules take the following form:

crl [name] : action (A_{Res}) < Rid : Resource| resStt : S > **=>** < Rid : Resource| resStt : S' > **if** (P_{Res})



Resource's Predicates	*Description*	*Resource's Predicates*	*Description*
(a) P_{Res}	isAvailable(R)	(i) P_{Res}	isUnlimited(R)
(b) P_{Res}	isConsApproved(R)	(j) P_{Res}	isRenewable(R)
(c) P_{Res}	isPubRes(R)	(k) P_{Res}	isStopped(R)
(d) P_{Res}	isPrvRes(R)	(l) P_{Res}	isLogical(R)
(e) P_{Res}	isConFinished(R)	(m) P_{Res}	isPhysical(R)
(f) P_{Res}	isUnbounded(R)	(n) P_{Res}	isProductionApplied(R)
(g) P_{Res}	isNotAvailable(R)	(o) P_{Res}	isProductionFinished(R)
(h) P_{Res}	isLimited(R)	(p) P_{Res}	isNotRenewable(R)

Resource's Actions	*Description*
(1) A_{Res}	prepareCons(R)
(2) A_{Res}	consume(R)
(3) A_{Res}	lock(R)
(4) A_{Res}	unlock(R)
(5) A_{Res}	update(R)
(6) A_{Res}	withdraw(R)
(7) A_{Res}	waitProdReq(R)
(8) A_{Res}	produceF(R)
(9) A_{Res}	makeAvailable(R)

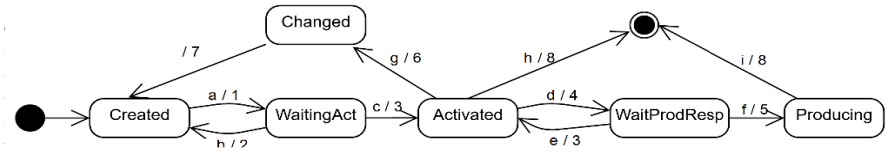
Fig. 4: Resource transition system.

4.2.3 Modeling Roles structures and behaviors

Structure encoding: the Role specification structure, states and transitions are specified in the system module *RoleSpec*. The latter contains all the necessary declarations defining the class Role.

```
class Role | rlSt : RlSt, rclck : Time, mrt : TimeInf, timer : TimeInf .
```

This class highlights the current state *rlSt*, local clock (*rclck*), the time value needed to receive the response message from the role whose (*mrt*) value is interested, and a timer (*timer*); four kinds of actions involving roles *actReq(R, R'), actRep(R, R'),* etc. On the one hand, to specify the resources production behavior provided by different Roles, we encode in Maude a set of predicates P_{Rol}() and actions A_{Rol}() in a given Role.



Role's Predicates	*Description*
(a) P_{RoI}	isActivationReqSent(RL)
(b) P_{RoI}	isActivationReqRejected(RL)
(c) P_{RoI}	isActivationReqAccepted(RL)
(d) P_{RoI}	isProductionReqSent(RL)
(e) P_{RoI}	isProductionReqRejected(RL)
(f) P_{RoI}	isProductionReqOk(RL)
(g) P_{RoI}	isRoleChanged(RL)
(h) P_{RoI}	isInactive(RL)
(h) P_{RoI}	isProductionFinished(RL)

Role's Actions	*Description*
(1) A_{RoI}	waitActivation(RL)
(2) A_{RoI}	rejectActivation(RL)
(3) A_{RoI}	active(RL)
(4) A_{RoI}	waitProdReq(RL)
(5) A_{RoI}	produce(RL)
(6) A_{RoI}	changeR(RL)
(7) A_{RoI}	create(RL)
(8) A_{RoI}	exit(RL)

Fig. 5: Role transition system.

On the other hand, we provide in the same module, an RT object-oriented specification with a protocol for specifying roles *Active/Inactive* actions times. To manage the unwanted behavior related to missions-based conflicts using rewrite rules *([!Synch-missions]*, *[!after-missions]*, *[!before-missions*]), e.g. the rule *[!Synch-missions]* offers to the concerned Role the ability to prevent simultaneous access of two missions *M1* and *M2* to a non-shared private resource, by executing the mission with the shortest execution duration and leaving the second waiting.

rl[!Synch-missions]: *notSynch(M1, M2)* < M1 : Mission | sm : T1, em : T'1, dn : Dn1, dx : Dx1, st : WaitResp, sg : Sig, rs : PrvR, rC : RC > < M2 : Mission | sm : T2, em : T'2, dn : Dn2, dx : Dx2, st : WaitResp, sg : Sig, rs : PrvR, rC : RC > **=>** < M1 : Mission | sm : T1, em : T'1, dn : Dn1, dx : Dx1, st changeSt(WaitResp, Dx1 - Dn1, Dx2 - Dn2), rs : PrvR, rC : RC > < M2 : Mission | sm : T2, em : T'2, dn : Dn2, dx : Dx2, st : changeSt(WaitResp, Dx1 - Dn1, Dx2 - Dn2), rs : PrvR, rC : RC >

4.3 SoS Controller Behavior Modeling

In this section, we present our controller which supports the Resources Management, dynamic roles interactions, and Systems behavior states.

4.3.1 Modeling Resources Management Controller

The consumption via the combination of Missions and Resources states and the production is supported by matching the states of Roles and Resources. The goal is to decide whether to accept or not Missions consumption (resp. Role production) requests. More specifically, this is accomplished by managing different predicated actions to decide to either change or not the different states of Missions, Roles, and Resources.

Structure encoding: to simulate the controller decisions, we present the module *ManagementCtrlSpec* for allocation/production control. We describe these specifications in terms of structure, consumption/production predicates, and actions. Missions may have access to resources, and Roles may affect their availability during the production process. These relationships are represented by the following class.

class ResManagementCtrl | mission : Oid, role : Oid, resource : Oid resMangS : ResMangStt.

Resources Management behavior: in the same module, we use the previous predicates (resp. actions), those related to Missions $P_{Mis}()$, Roles $P_{Rol}()$ and Resources

$P_{Res}()$, (resp. $A_{Mis}()$, $A_{Rol}()$ and $A_{Res}()$) to formalize the new ones $P_{RC}()$ (resp. $A_{RC}()$) that manage the consumption/production of *Resources* (Fig. 6.). To illustrate how this controller *RC* works, consider that *RC* is holding in *AnalyseConReq*, the transition system will fire the transition "*b / 2"* and perform the action *(2)* A_{RC} to move to *ConsReqAccepted* state whenever the predicate *(b)*P_{RC} becomes true. The latter is composed of five predicates (see table in Fig.6), namely, *(a)*P_{Mis}=a consumption request of *M* is sent, *(b)*P_{Mis} = its temporal constraints are respected, *(a)*P_{Res} = resource R is available, *(b)*P_{Res} = consumption is approved, and *(c)*P_{Res} = R is public see table in Fig. 3 and Fig. 4, respectively. Whenever these predicates become true, the action *(2)* A_{RC} in turn will trigger two actions*(2)* A_{Res} = *Consume* R and *(3)* A_{Mis} = *execute M*.

Resource Controller's Predicates	*Corresponding Action*
(a) P_{RC} = (a) P_{Mis}	(1) A_{RC} = (1) A_{Mis}
(b) P_{RC} = (a) P_{Mis} && (b) P_{Mis} && (a) P_{Res} && (b) P_{Res} && (c) P_{Res}	(2) A_{RC} = (3) A_{Mis} && (2) A_{Res}
(c) P_{RC} = (a) P_{Mis} && (b) P_{Mis} && (a) P_{Res} && (d) P_{Res}	(3) A_{RC} = (2) A_{Res}
(d) P_{RC} = (a) P_{Mis} && (b) P_{Mis} && (b) P_{Res} && (d) P_{Res}	(4) A_{RC} = (3) A_{Mis} && (2) A_{Res}
(e) P_{RC} = (h) P_{Mis}	(5) A_{RC} = (3) A_{Mis}
(f) P_{RC} = (h) P_{Mis} && (j) P_{Res}	(6) A_{RC} = (3) A_{Mis} && (9) A_{Res}
(g) P_{RC} = (p) P_{Res}	(7) A_{RC} = (6) A_{Res}
(h) P_{RC} = (d) P_{Rel}	(8) A_{RC} = (4) A_{Rel} && (7) A_{Res}
(i) P_{RC} = (f) P_{Rel} && (n) P_{Res}	(9) A_{RC} = (8) P_{Rel}

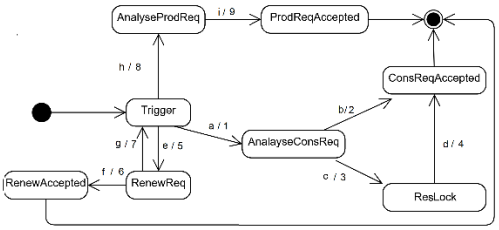


Fig. 6: Resources Management Controller transition system.

We encode allocation actions as conditional rewrite rules, i.e. their triggering conditions are the state predicates encoded above and their triggered actions are encoded as Maude computation. The different actions express atomic controlling behaviors for resource consumption/production, missions and roles. The conditional rewrite rules in this case will be as follow:

```
crl [rewrite-rule-name] : action (A_RC) < Rid : Resource| resStt : RS > <: Mission | state : MS, res : Rid > < RCid : ResManagementCtrl |mission : Mid, resource : Rid, state : RCS > => < Rid : Resource| resStt : RS' > <: Mission | state : MS, res : Rid' > < RCid : ResManagementCtrl |mission : Mid, resource : Rid, state : RCS' > If (P_RC)
```

So as not to go into too much detail, we give a simplified example of the rule*[lock-resource]:* If *Rid* is *PrvRes*, or *renewable*, the controller sends a *lock(Rid)* message to *Rid* and moves to the *Lock* state.

```
crl[lock-resource]: lock(Rid) < Rid: Resource | resSt : PrvRes > < Mid : Mission | st : WaitResp, rs : Rid > < ACid : AllocationCtrl | Mid, Rid, resSt : AnalyseConsReq > => < Rid: Resource |resSt : Locked, rT : PrvRes > < Mid : Mission |st : Executing, rs : Rid > < ACid : AllocationCtrl | Mid, Rid, resSt : ResLock > if (isLock(Rid) && isPubRes(Rid) && Sig =TCR)|| (isLimited(Rid) && isRenwable(Rid) && isAvailable(Rid) && Sig =TCR)
```

4.3.2 Systems behavior specification

To control the desired behaviors and prevent the unwanted ones, it is necessary to study the global state of the system that could be affected by various factors which are related to the different states of roles, missions, and resource consumption/production. For this aim, the module *SoSSpec* provides the rewriting logic mean that can specify and check the global configurations of SoS design, and then, ensure the correctness of the evolved SoS substructures.

```
class SoS| csL : CSL, ml : ML, rl : RL, resl : ResL, csst : CSSt .
```

Structure encoding: the SoS Maude-based specification in terms of organizational composition (e.g. ownership, dependency, etc), transitions, and states are defined in

class SoS. The latter has five attributes, namely, *mL*, *rL*, *resL* and *CSL*, for CSs, missions, roles, and resources lists, and *CSSt* to describe the CS state (*ignored, joined, active*, etc.). This class describes the global state of the system according to its CSs.
Systems global behavior: we take advantage of the specified states of missions, resources, and roles to describe the behavior of the participating CSs. We give rewrite rules that express guarded configuration implementing the desired behavior of CSs playing Roles (the rules *[to-ignore]*, *[to-prepare]*… are well detailed in section 4.2).

5 Simulation and Analysis: case study 'EIMSoS'

The SoSs features depend on different states, which are expressed in terms of the relationship between Missions and Resources considering the temporal features. Then, we will specify the dynamic global behavior that emerges from the SoS.

5.1 Maude-based modeling of the Resource Management

We show the use of the Resource Controller by applying a sequence of rules (module *ResourceCtrlSpec*) on a configuration of the *EIMSoS* specified in Section 2.3. Given object identifiers, *H1*, *Dep*, and *AMECSctrl*, the following term may represent a configuration with a private and limited resource which is a *Helicopter*, a mission *Departure*, a resource controller for *AeroMedicalEmergencyCS*, respectively, and an *access(Dep, H1)* message from *Dep* to *AMECSctrl* to enable the allocation process.

```
< Dep : Mission | 5000, 450, 550, Idle, TCR > < H1 : Resource | lim, NotCns, PrvRes > < AMECSctrl : AllocationCtrl | Dep , H1 >
```

The execution of this controller is modeled with a set of rewriting rules. For a given scenario, when the mission *Dep* needs to consume/use the resource *H1* the request-analysis rule, will send the controller an access request and it will wait at *WaitResp* state for a response. If ever *H1* is not available (state *notAv*), the controller will send a rejection (*sendRej(H1, Dep)*) replying to the mission, and the latter will back to state *Idle* in *request-rejection* rule.

```
crl [request-analysis] : < Dep : Mission | 500, 450, 550, Idle, TCR > <H1: Resource | lim, resSt, prvRes > accesReq(Dep, H1) => < Dep : Mission | 5000, 450, 550, WaitResp, TCR > < H1: Resource | lim, resSt, prvRes > < AMECSctrl : AllocationCtrl | Dep, H1> sendRej(H1, Dep) if resSt (H1) = notAv .
crl [request-rejection] : isReqRej(Dep) < Dep : Mission | 500, 450, 550, WaitResp, TCR > => < Dep : Mission | 500, 450, 550, Idle, TCR >
```

Contrarily, if the mission receives an *ok* message (*isRespOK (H1, Dep)*), it will start the execution (*State Exec*), and the controller sends a *ResLock(H1)* message to *H1* and moves to the *ResLock* state.

```
crl [request-accept] : < Dep : Mission | 500, 450, 550, Idle, TCR > <H1: Resource | lim, resSt, prvRes > accesReq(Dep, H1) => < Dep : Mission | 5000, 450, 550, Exec, TCR > < H1: Resource |lim, lckd, prvRes > < AMECSctrl : AllocationCtrl | Dep, H1> ResLock(M, R) if canBeCons(H1) && isTCR(Dep)) || (isLimButRnwable(H1) && isAv(H1) && isTCR(Dep)) (lock(H1) && isPubRes(H1) && canBeConsumed(H1) && isTCR(Dep)) || (isLimButRnwable(H1) && isAv(H1) && isTCR(Dep))
```

If the mission is taking longer than expected, a signal event *MxV(Dep)* is transferred to inform the SoS that the duration constraint is violated.

```
crl [request-TCViolated] : < Dep : Mission | ET, 450, 550, Exec, TCR > <H1: Resource | lim, resSt, prvRes> accesReq(Dep, H1) => < Dep : Mission | ET, 450, 550, Exec, MxV > < H1: Resource | lim, lckd, prvRes > < AMECSctrl : AllocationCtrl | Dep, H1> if ET > 550 .
```

5.2 Maude-based modeling of the SoS states configuration

According to the SoS class, each CSs state has a relation with the list of roles *RL* that he plays and the relevant list of missions *ML* as well. Based on these two lists, several states and transitions naturally emerge within a CS that is relevant to the SoS:

- *IgnoredCS*: a CS which has neither the relevant *ML* to accomplish, neither the roles to play [to-ignore], because *RL* and *ML* lists are empty.
- *PreparedCS*: a CS which has the relevant requirements of the SoS, but their states are not triggered yet. Notice that the system *RSCS* is rewritten to move to the state *ignoredCS*. This is due to the mission *Dep* state is still *idled*.
- *PassiveCS*: a CS that has joined the SoS but none of its roles as activated i.e. waiting for activation requests. Notice that the system *RSCS* is rewritten to move to the state *PassiveCS*. This is due to the mission Dep waiting *WaitResp* for a response to start execution and the role *Rec* is not activated yet.
- *Active*CS: a CS that is participating in a constellation of the SoS once he activates one or more role from the *RL*. This time the *RSCS* is rewritten to move to the state *ActiveCS*. Because mission *Dep* state is not *idled* and the role *Rec* is activated.

```
rl [to-ignore] : < RSCS : SoS | ML, RL, csst : CSSt > ignoreCS (RSCS, RL, ML) => < RSCS : SoS | ML, RL, IgnoredCS : CSSt >  crl [to-prepare] : < RSCS : SoS | M[Dep : Mission | eT, 450, 550, Idle, TCR], RL, csst : CSSt > prepareCS (RSCS, RL, ML) =>  < RSCS : SoS | M[Dep : Mission | eT, 450, 550, MSt, TCR], RL, PreparedCS : CSSt > if MSt = Idled .
crl [to-passive] : < RSCS : SoS | M[Dep : Mission | eT, 450, 550, WaitResp, TCR],  R[Rec : Role | clck : 0,  timer : INF, rid : R, mrt : INF], csst : CSSt > ignoreCS (RSCS, RL, ML) => < RSCS : SoS | M[Dep : Mission | eT, 450, 550, MSt, TCR],  R[Rec : Role | clck : 0, timer : INF, rid : R, mrt : INF], PassiceCS : CSSt >  if MSt = WaitResp && isRoleAct(Rec) = false.
crl [to-prepare] : < RSCS : SoS | M[Dep : Mission | eT, 450, 550, WaitResp, TCR], R[Rec : Role | clck : 0, timer : INF, rid : R, mrt : INF],  csst : CSSt > ignoreCS (RSCS, RL, ML) => < RSCS : SoS | M[Dep : Mission | eT, 450, 550, MSt, TCR],  R[Rec : Role | clck : 0, timer : INF, rid : R, mrt : INF], ActiveCS : CSSt >  if MSt = WaitResp && isRoleAct(Rec) = true .
```

6 Conclusion

The main contributions of this paper are threefold, we first proposed a generic metamodel that emphasizes different aspects oriented towards a set of qualitative and quantitative features. Then, we proposed a formal approach for the modeling and specification of SoSs structures and behaviors based on temporal and resources features. Finally, we provided an execution and verification solution of the defined behaviors, using the rewriting-logic-based Maude system, and we validated it using an Emergency Incident Management SoS as a case study. For future work, we plan to formally verify the controller's properties with a symbolic state-based model-checking technique relying on (LTL) as a temporal logic to consider symbolic high-level system states. In addition, we seek to extend the work in progress by considering new strategies of coordination related to temporal modeling, analysis, and verification.

References

[1] C. B. Nielsen, G. Peter Larsen, J. Fitzgerald, J. Woodcock, andJ. Peleska, "Systems of systems engineering: Basic concepts, model-based techniques, and research directions,"ACM Computing Survey, vol. 48, no. 2, pp. 18:1–18:41, sep 2015.

[2] MaierMW. Architecting principles for systems-of-systems. Systems Engineering 1998; 1(4): 267 – 284
[3] Combi, C., Oliboni, B., & Zerbato, F. (2017, April). Modeling and handling duration constraints in BPMN 2.0. In Proceedings of the Symposium on Applied Computing pp. 727-734.
[4] Graja, I., Kallel, S., Guermouche, N., & Kacem, A. H. (2019). Towards the verification of cyber-physical processes based on time and physical properties. International Journal of Business and Systems Research, 13(1), 47-76.
[5] Cheikhrouhou, S., Kallel, S., Guermouche, N., & Jmaiel, M. (2013, December). Toward a time-centric modeling of business processes in BPMN 2.0. In Proceedings of International Conference on Information Integration and Web-based Applications & Services (pp. 154-163).
[6] Mori, M., Ceccarelli, A., Lollini, P., Bondavalli, A., & Fromel, B. (2016). A holistic viewpoint-based SysML profile to design systems-of-systems. 2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE).
[7] Mori, M., Ceccarelli, A., Lollini, P., Frömel, B., Brancati, F., & Bondavalli, A. (2017). Systems-of-systems modeling using a comprehensive viewpoint-based SysML profile. Journal of Software: Evolution and Process, 30(3), e1878.
[8] Seghiri, A., Belala, F., & Hameurlain, N. (2022). A Formal Language for Modelling and Verifying Systems-of-Systems Software Architectures. International journal of systems and service-oriented engineering (IJSSOE), 12(1), 1-17.
[9] Seghiri, A., Belala, F., & Hameurlain, N. (2022, April). Modeling the Dynamic Reconfiguration in Smart Crisis Response Systems. In 17th International Conference on Evaluation of Novel Approaches to Software Engineering (pp. 162-173). SCITEPRESS-Science and Technology Publications.
[10] Gassara, A., Bouassida Rodriguez, I., Jmaiel, M., & Drira, K. (2017). A bigraphical multi-scale modeling methodology for system of systems. Computers & Electrical Engineering.
[11] Axelsson, J., Fröberg, J., & Eriksson, P. (2019). Architecting systems-of-systems and their constituents: A case study applying industry 4.0 in the construction domain. Systems Engineering, 22(6), 455-470.
[12] Baek, Y., Song, J., Shin, Y., Park, S., & Bae, D. (2018). A meta-model for representing system-of-systems ontologies. Proceedings of the 6th International Workshop on Software Engineering for Systems-of-Systems - SESoS '18.
[13] Zhang, Y., Liu, X., Wang, Z., & Chen, L. (2012). A Service-Oriented Method for System-of-Systems Requirements Analysis and Architecture Design. JSW, 7(2), 358-365.
[14] Halima, R. B., Klai, K., Sellami, M., & Maamar, Z. (2021, September). Formal Modeling and Verification of Property-based Resource Consumption Cycles. In 2021 IEEE International Conference on Services Computing (SCC) (pp. 370-375). IEEE.
[15] Maamar, Z., Faci, N., Sakr, S., Boukhebouze, M., & Barnawi, A. (2016). Network-based social coordination of business processes. Information Systems, 58, 56-74.
[16] Graiet, M., Mammar, A., Boubaker, S., & Gaaloul, W. (2016). Towards correct cloud resource allocation in business processes. IEEE Transactions on Services Computing.
[17] Axelsson, J. (2019, May). A refined terminology on system-of-systems substructure and constituent system states. In 2019 14th Annual Conference System of Systems Engineering (SoSE) (pp. 31-36). IEEE.
[18] Andrews, R., Wynn, M. T., Vallmuur, K., Ter Hofstede, A. H., Bosley, E., Elcock, M., & Rashford, S. (2019). Leveraging data quality to better prepare for process mining: an approach illustrated through analysing road trauma pre-hospital retrieval and transport processes in Queensland. International journal of environmental research and public health, 16(7), 1138.
[19] J. Meseguer. “Rewriting logic and Maude: a Wide-Spectrum Semantic Framework forObject-based Distributed Systems”.In S. Smith and C.L. Talcott, editors, Formal Methods forOpen Object-based Distributed Systems, (FMOODS’2000), p. 89-117. Kluwer, 2000.
[20] M. Clavel, F. Duran, S. Ecker, P. Lincoln, N. Marti-Oliet, J. Meseguer, C. Talcott. “The Maude 2.0 System”. In Proc. Rewriting Techniques and Applications (RTA), Volume 2706 of LNCS, Spring-Verlag , p. 76-87., 2003.