A Maude-Based Formal Approach to Control and Analyze Time-Resource Aware Missioned Systemsof-Systems

Charaf Eddine, Nabil Hameurlain, Faiza Belala

A Maude-Based Formal Approach to Control and Analyze Time-Resource Aware Missioned Systems-of-Systems

Charaf Eddine[1,2], Nabil HAMEURLAIN[1], Faiza BELALA[2]
LIUPPA Laboratory, University of Pau, Pau, France
[1]{charaf-eddine.dridi, nabil.hameurlain}@univ-pau.fr
LIRE Laboratory, Constantine2 University-Abdelhamid Mehri, Constantine, Algeria
[2]{charafeddine.dridi, faiza.belala}@univ-constantine2.dz

***Abstract*— Systems-of-Systems (SoSs) are increasingly utilized to integrate multiple Constituent-Systems (CSs) and fulfill missions surpassing the capabilities of individual systems. Successful mission execution relies on effective management of time and resources. Time constraints denote the timeframes within these missions must be completed, and resource constraints involve the allocation of necessary resources for execution. Addressing these dynamic behavior factors in SoSs poses significant challenges and necessitates the adoption of control strategies for efficient coordination and execution of missions. To confront these challenges, this paper introduces a novel hybrid approach, merging an MDA technique (TRC-MM: Time-Resource Aware Control Meta-model) and Maude language to formally specify and control missions. This approach specifically incorporates time-resource aware missions, resource dependencies, and control mechanisms in the dedicated TRC-MM. Further, RT-Maude is utilized as the formal specification language to develop an executable formal model for the controller's behavior in SoSs' missions, offering a powerful analysis method to express and verify behavioral properties pertaining to time and resource constraints.**

Keywords—SoS, Maude, Formal Method, Model-checking, Timed mission, Control strategies.

I. INTRODUCTION

Systems-of-Systems (SoSs) have emerged as a prominent solution to tackle the challenges posed by large-scale integration and coordination of IT systems. SoSs involve the seamless integration and execution of multiple Constituent-Systems (CSs) to achieve missions that individual CSs cannot accomplish alone [1]. The centralized control in these systems ensures a cohesive and coordinated approach towards the common goals of SoSs. The presence of a central authority enables effective management, coordination, and alignment of the CSs' towards the common goals of the system [1] [2].

Designing SoSs to address centralized control challenges and ensure adherence to specified behavior is complex. This process includes selecting the optimal functional chain to achieve overall missions at the SoS level within time constraints and implementing effective strategies, regulatory mechanisms, and corrective actions at the CSs level. Developing a corresponding formal model is also crucial to analyze, simulate, and verify logical properties. To this end, this paper proposes a systematic approach using a meta-model, TRC-MM, designed to optimally arrange CSs to fulfill mission objectives within time and resource constraints. TRC-MM specifically addresses time aspects, resource dependencies, and control mechanisms, allowing for dynamic adjustment of behavior to changing runtime conditions.

Fig.1 describes the three main steps of our approach. Firstly, it starts by introducing a generic *TRC-MM* that captures the essential structural, behavioral, and quantitative aspects. The *TRC-MM* serves as a support tool that enables the creation of concrete models tailored to specific application domains within SoSs. The adopted MDA technique ensures a comprehensive representation of characteristics, including temporal aspects and resource allocation. Secondly, we rely on a formal execution semantic using a rewriting engine based on RT-Maude and Strategy Language [3] [4] to control and simulate the dynamic behavior of the proposed controller. Lastly, we use model-checking tool of RT-Maude to verify the satisfaction of some specific properties and constraints.

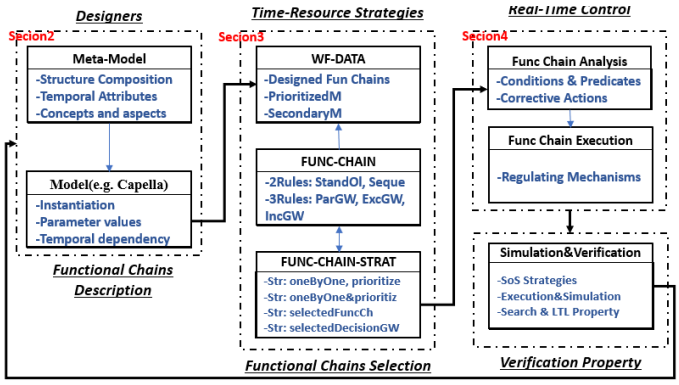

Fig. 1. The proposed approach.

II. MODELING TIME-RESOURCE AWARE MISSIONS IN SOS

This section focuses on creating the *TRC-MM*, specifically designed for Time-Resource Aware Missions in SoSs.

A. TRC-MM: Time-Resource Aware Control Meta Model

The *TRC-MM* serves as a standardized set of basic classes that capture the essential aspects of missioned SoSs. It integrates into a systematic framework, encompassing structural, behavioral, control strategies, etc. In *Part1* of Fig. 2, we find classes that represent specific missions within the CSs. They are associated with *MTemporalConstraints* and *States* classes that capture the various states of a given mission. Readers are referred to [2] for more details. *Part2*: represents the entities or components required for executing missions within the SoS. It is associated with *ResourceStates* with *PrvRes* and *PubRes* for private and global resources that describe the various states, types or categories of resources. The *ResPool* class represents a centralized repository or pool of resources that can be shared and allocated among missions in the SoS. In the controlled time bound missions, the *ResPool* serves as a shared repository for spare or unused resources within CSs. When a CS has surplus or available resources that are not currently being utilized, it contributes those resources to the Resource Pool, allowing other CSs to

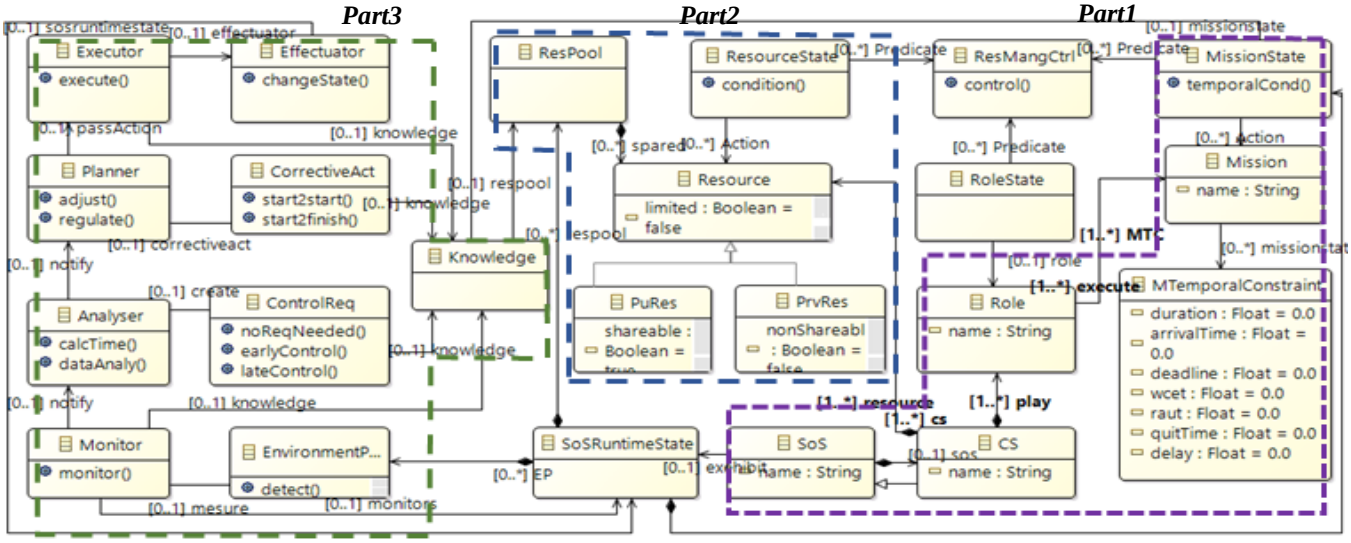

Fig. 2. Time-Resource Aware Control Meta-Model for SoS.

utilize them. *Part3* represents classes that are associated with *MissionState* and *ResPool*, serving as the knowledge base that stores information about the current state of missions and the availability of resources.

B. *Running Example: French Emergency SoS*

The case study highlights the logical architecture design of the French Emergency SoS [7] [8]. The deduced model of FESoS is based on *TRC-MM* and its logical architecture is designed using Capella editor [18] as show in Fig. 3. The FESoS is designed for emergency protection, it features interconnected CSs and missions such as MonitoringSoS deploying UAVs[1] and WSNCS[2], CODIS[3] overseeing operations, and SAMU[4] for emergency medical services. The HospitalCS manages patient reception and assessment, while medical treatments and continuous monitoring are ensured. Civil Security is crucial for coordination, ensuring information flow and collaboration. SDIS64[5] and SDIS65 represent firefighting efforts of two departments, deploying equipment and firefighters. Search and Rescue Teams prioritize finding and evacuating survivors, while the Fire and Rescue Services handle fire control and hazardous materials. The study underscores the integrated efforts of these CSs to safeguard people and property. The FESoS model focuses on the representation of multiple alternative paths or simultaneous missions' execution. In addition, parameter values offer a comprehensive insight into the functional chains, aiding designers in exploring and assessing various design options and system behaviors, Fig. 4 capture a piece of the entire figure which shows the description of the functional chains of the FESoS behavior.

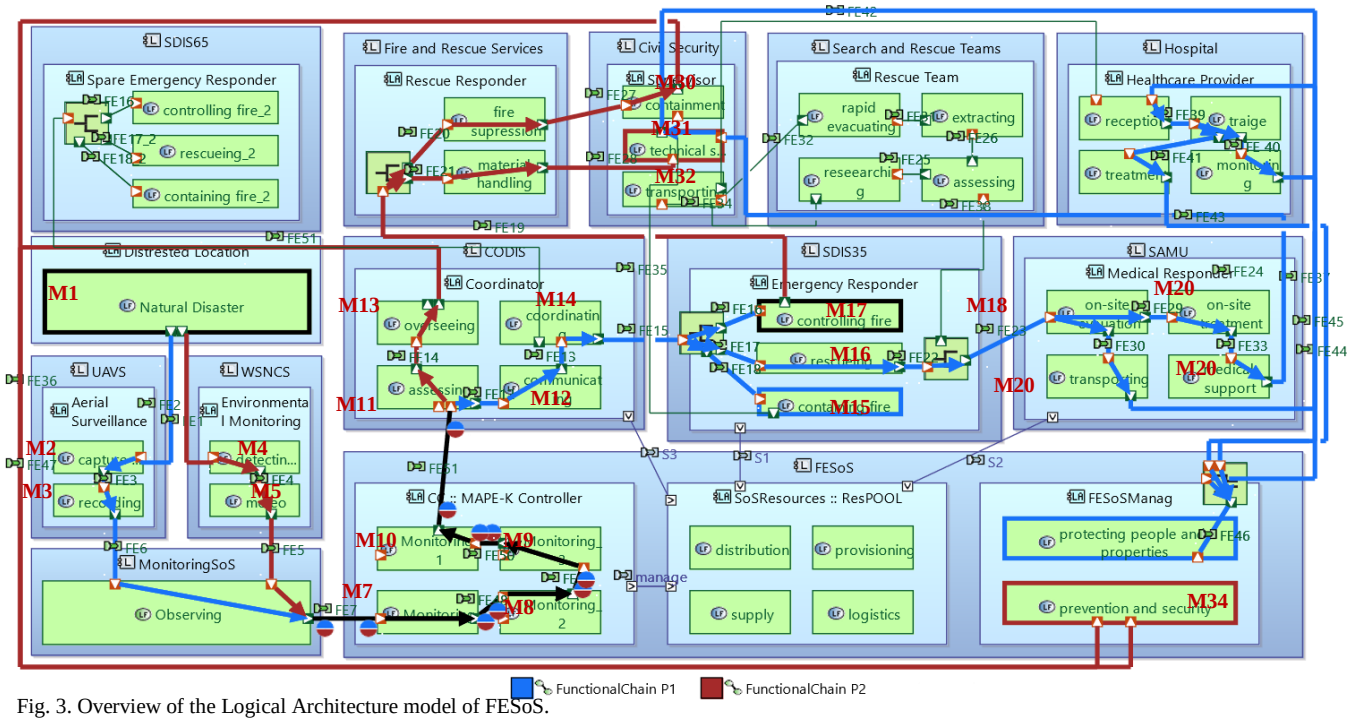

Fig. 3. Overview of the Logical Architecture model of FESoS.

[1] Unmanned Aerial Vehicles
[2] Wireless Sensor Net
[3] Centre Opérationnel Départemental d'Incendie et de secours
[4] Service d'aide médicale urgente
[5] Service Départemental d'Incendie et de Secours

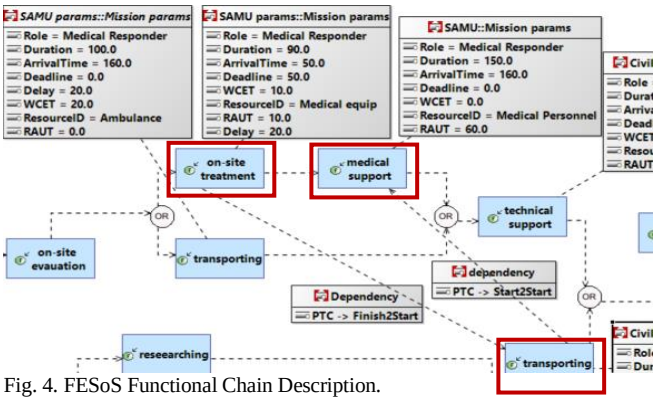

Fig. 4. FESoS Functional Chain Description.

III. MAUDE-BASED FORMAL SPECIFICATION OF SOSS CONTROL

This section presents the specification of dynamic SoS control mechanisms, emphasizing the SoS's global-level controller, with the objective of optimizing functional chain execution for efficient global mission attainment. Simultaneously, at the mission and local CSs levels, the controller oversees RT-regulation to reduce execution time.

A. RT-Maude and Strategy Language

By leveraging Maude's concurrent rewriting capabilities and equational structural axioms, we can logically deduce and reason about the behavior of the system[4]. In the context of temporal mission priorities and states, the modules in RT-Maude denoted as a tuple $R = (\Sigma, E \cup A, R)$ provide a formal and structured representation of the system's behavior and transitions, incorporating temporal aspects. The equational theory part of the modules, represented by $(\Sigma, E \cup A)$, encompasses the signature $\Sigma = (S, C, \leq, F, M)$, which includes sorts and subsorts S, class names C, a subclass relation $\leq$, and a set of functions and messages. The set E denotes equations and membership tests, some of which can be conditional, while A represents equational axioms associated with specific operators in the signature Σ. The tuple R consists of both conditional and nonconditional rewrite rules. In the context of Real-Time rewriting theory, $(\Sigma, E \cup A)$ includes an additional sub-theory $(\Sigma TIME, \Sigma TIME) \subseteq (\Sigma, E \cup A)$, specifically modeling time. The behavioral transitions are defined using rewrite rules of the form: ***crl[R]: {t} => {t'} in time u if (condition)***. Using these rules, we can accurately model how time and the states of the system evolve, which is crucial for understanding and verifying the properties of RT and hybrid systems. RT-Maude extension is designed to decouple the rewrite rules, from their execution via strategies. The system module's definition evolves into $(\Sigma, E \cup A, S(R, SM))$, where S characterizes the system's behavior. i.e. S is delineated by two components: R, which comprises a set of potential conditional rewrite rules, and SM, the strategy module directing the rewriting procedure. Rewrite rules define transitions between the configurations. RT-Maude' high-performance rewrite engine, allows simulation, analysis, and model-checking. Maude offers commands to explore system behaviors from an initial configuration, e.g. *rewrite*, *srewrite*, and *tsearch*.

B. Maude-based Specification of TRC-MM Static Entities

The *TRC-MM* entities can be mapped to a set of RT-Maude concepts. Table.1 summarizes the operational semantic of different static elements introduced in *TRC-MM*, i.e. we enumerate all sorts and operators essential for characterizing SoS. The primary classes defined are *SoS*, *CS*, and *Mission* accompanied by their associated sorts and ops.

TABLE 1: RT-MAUDE SEMANTICS OF THE TRC-MM's ELEMENTS.

Entities	RT-Maude specification
SoS CS Mission Role	class SoS | clock : Time, CSsSet : CSIdSet, , missionSet : GMIdSet class CS | clock : Time, missionSet : MIdSet . class Mission | localClock : Time, duration : Time, arrivalTime : Time, quitTime : Time, delay : Time, deadline : Time, missionState : MissionState resType : ResType, RAUT : Time . class Resource | localClock : Time, resType : ResType,
CSs Queue Mission Queue Global Mission resPool	sort CSIdSet . subsort Oid < CSIdSet . subsort Oid < MIdSet < GMIdSet . subsort Resource < ResourcePool .
MissionState ResType ResState Pool	ops arrived accomplished pending failed : -> MissionState . ops shareable nonShareable limited renewable: -> ResType op resType:_resState:_quantity:_ : ResType State Time -> Resource [ctor] . op emptyPool : -> ResourcePool [ctor] . op '[Pool:_'] : ResourcePool -> ResPool [ctor] . op indeterminatedWF : funcChain -> Bool .
Conditions	eq calcState(DD, ED, DT) = if (DD > ED plus DT) then accomplished else pending fi . eq calcQuitTime(ED, DT) ... eq calcQuitTime(ED, DT, ST) = ED plus DT plus ST . eq calcRemainedResource(ST, D) = ST plus D . eq indeterminatedWF(M2 M3 SecondoryM | PrioritizedM M1) eq indeterminatedWF(M1 M2 SecondoryM | PrioritizedM) ...
Corrective actions	msg initSoS : Oid -> Msg . msg finishAt : Oid Time -> Msg . msg start2Start : Oid -> Msg . msg start2Finish : Oid Time -> Msg . msg finish2Start : Oid Time -> Msg .

C. Control Strategies Formal Semantics

In this section, we propose strategies for optimal functional chain or mission's workflow selection, exploring their role in efficient global mission attainment by minimizing execution time and optimizing resources. Table 2 aligns each behavioral element of the *TRC-MM* with semantics, specifying the mission's workflow in three Maude modules:

(1) WF-DATA module: along with two other constants: *PrioritizedM* represents a subset of missions that are given priority in terms of short duration and resource constraints, and *SecondaryM* denotes missions that are not prioritized for scenario but still exist within the SoS and could be relevant in future executions. The operator "Union" denoted by _|_ signifies the classification of missions designating the secondary and prioritized missions that will be selected to execute in the functional chain. The conditions specified in the *indeterminatedWF* (Table. 1) outline the predicates under which certain mission workflows (caused by gateways) are in a nondeterminate states, making it challenging to predict or control the exact sequence of missions for optimal outcomes.

TABLE 2 : MISSION PRIORITIZATION AND MANAGEMENT

Rule	Condition	Action	Description
standaloneM (M1)	M1 without any order	Move M1 to PrioritizedM	A single mission M1, is promoted to the prioritized list.
sequenced (M1,M2)	M1, M2 Are sequenced	Move M1 and M2 to PrioritizedM	M1 and M2 are sequenced and both are promoted to the prioritized list.
parallelM-GW(M1,M2)	M1, M2 are in parallel	Move M1, and M2 to PrioritizedM	M1 and M2 are concurrently promoted to the prioritized list.
exclusiveM-GW(M1, M2) / inclusiveM-GW(M1, M2)	Duration(M1) < Duration(M2) OR RAUT(M1) < RAUT(M2)	Move M2 to PrioritizedM	M1 is prioritized if it completes faster or uses resources more efficiently; otherwise, if neither condition is met, M1 is prioritized based on its shorter duration

(2) FUNC-CHAIN module: it imports the previous module and it involves the possible executions of missions specified using rewriting rules (see Table 2). The first two rules are implemented to execute standalone and sequence missions, the other three rules are specified to execute the branching and merging nondeterministic behavior introduced by *gateways*(*parallelM-GW*, *exclusiveM-GW* and *inclusiveM-*

GW). These gateways might not always result in the most efficient execution because there are situations where the workflow can become indeterminate when using gateways.

(3) CHAIN-STRAT module: It tackles the challenge of managing nondeterminism states in mission execution. To achieve this, we enforce strict control over gateway rules before initiating standalone and sequenced execution rules. This enforcement is accomplished through the specification of a set of strategies within this module. The strategy *oneByOne* applies either of the two execution rules (*standaloneM* or *sequencedM*), and the strategy *prioritize* applies any of the three gateway rules. The strategy *oneByOne&prioritize* applies either of two previous strategies based on the rules of *FUNC-CHAIN*, executing only if *prioritize* isn't an option. The *selectedFuncCh* strategy is recursive, continually repeating this process until the global mission is achieved. Meanwhile, *selectedDecisionGW* strategy is more selective, focusing on the outgoing branches (i.e., prioritized missions) and sidestepping all secondary mission paths. Example:

```
sd selectedDecisionGW := (match SecondoryM | PrioritizedM M1 M2 M3) ? idle : (oneByOne&prioritize ; selectedDecisionGW) .
sd selectedFuncCh := (match SecondoryM | G) ? idle : (oneByOne ; prioritize ; selectedFuncCh) .
```

D. Local Control Mechanisms of Missions

We present in this section a formal specification of the self-regulating of the SoS controller; it allows to monitor locally the functional chain execution, analyze temporal attributes and execute corrective actions adjusting missions of a given CS to guarantee timely completion. See Table 3.

TABLE 3: TEMPORAL CONTROL MECHANISMS FOR SoS MISSIONS

Mech	Conditions	Corrective Action	Description
Early Arrival	M1 arrives earlier than expected	Start-2-Start (S2S) adjustment for M2	Adjust the start time of M2 based on M1's arrival
	M1 arrives at time T (T < T ')	M2 starts at time T	
	M1 arrives earlier than expected	Start-2-Finish (S2F) adjustment for M2	Adjust the finish time of M2 based on M1's arrival
	M1 arrives at time T (T < T ')	M2 finishes at time T	
Late Completion	M1 takes longer to complete	Finish-2-Start (F2S) adjustment for M2	Adjust the start time of M2 based on M1's delay
	T < than initially anticipated	M2 starts at time T	
	M1 takes longer to complete	Finish-2-Finish (F2F) adjustment for M2	Adjust the finish time of M2 based on M1's delay
	T > than initially anticipated	M2 finishes at time T	
No Needed	No deviations or issues	No corrective action required	missions are proceeding as planned
T: the expected time| T': the actual time it arrives or finishes			

Conditions part in the table considers resource availability and time constraints. It utilizes equations to assess the accessibility of required resources within the specified constraints. Moreover, the specification leverages RT-messages and operations to implement corrective actions in table such as *start2start, ASAP*, etc. This generates Regulating Mechanisms that address these temporal constraints, manages execution and facilitates corrective actions in response to changing conditions using rules. Table 3. explores the temporal control mechanisms seen in *TRC-MM*, highlighting their relevance in addressing early arrival, late completion, and no control needed scenarios. In the case of Early Arrival, if Mission M1 arrives earlier than expected (at time T, where T < T'), the *Start-2-Start* (S2S) corrective action is applied. This adjustment ensures that Mission M2 starts at the same time (T) as Mission M1, synchronizing their execution. The table's entries illustrate the corrective actions for each control mechanism, specifying the adjustments made to the starting and finishing times of the respective missions. These control mechanisms are related to the RT-Maude rules. For example, the *Start-2-Start* control mechanism can be represented as "**crl[Start-2-Start]: {M1 starts at time T} => {M2 starts at time T} in time u if (T < T')**". This rule signifies that if Mission M1 starts at time T and meets the condition of an early arrival (T < T'), then Mission M2 should also start at T.

```
rl [finish2start] : finish2start(M1, M2, ST) < M1 : Mission | loClock : D, duration : ED, arrivalTime : AT, quitTime : AT, delay : DT, deadline : DD, missionState : arrived, raut : RA >
< M2 : Mission | loClock : D', duration : ED', arrivalTime : AT', quitTime : AT', delay : DT', deadline : DD', missionState : arrived, raut : RA > =>
< M1 : Mission | loClock : R, duration : ED, arrivalTime : ST plus R, quitTime : calcQuitTime(ED, DT, ST), delay : DT, deadline : DD, missionState : calcState(DD, ED, DT), raut : RA >
< M2 : Mission | loClock : R, duration : ED', arrivalTime : calcQuitTime(ED, DT, ST), quitTime : calcQuitTime(ED, DT, ST) plus DT' plus ED', delay : DT', deadline : DD', missionState : arrived…
```

IV. Validation Through FESoS Case Study

In this section, we aim to assess the controller's efficacy and verify model adherence to specific properties, focusing on Maude's model-checking and invariant. The approach involves a breadth-first search of states to confirm crucial SoS properties, validating invariants. If certain outcomes, the inverse of the invariant, are unattainable and the invariant is deemed true. If *search init =>* C: Configuration such that not I(C:Configuration*)* yields no result, the invariant is valid.

A. Controlling FESoS with strategies

Fig. 3 illustrates various functional chains that have the potential to achieve global mission *M34*. These missions are designated as *M1, M2, M3,...,Mn* for simplicity. Using Maude Strategy Language, these specifications can be immediately executed. The 'search' command is particularly handy for this purpose, enabling exploration of all potential functional chain rules and identifying terms that correspond with a set target.

```
Maude> search initial =>* SecondoryM M1 M2 M3 M4 M5 M6 M7 M8 M9 M10 …| PrioritizedM .
Solution 1 (state 54)
states: 75 rewrites: 640
empty substitution
No more solutions.
states: 80
```

Even if the desired state can be achieved, there's always a certain degree of indeterminism regarding whether it adheres to the module's rules. Indeed, the path the *search* command followed to arrive at the global Mission M34 involves navigating through *indeterminatedWF* states. This can be validated by employing the *show path* command:

```
Maude> show path 10 .
state 0, M0: PrioritizedM  | SecondoryM M1 M2 M3 M4 M5 M6 M7
===[ rl ... [sequencedM] . ]===>
state 2, M1: SecondoryM M0 | PrioritizedM M2 M3 M4 M5 M6 M7
===[ rl ... [sequencedM] . ]===>
state 10, M2: PrioritizedM  M0 M1  | SecondoryM M3 M4 M5 M6 M7
...
```

Now, we can check whether the *M34* for instance can be reached by evaluating *selectedDecisionGW* from the initial state with the *srewrite* command. The answer is positive as shown in the following Maude code:

```
Maude> srew initial using selectedFuncCh .
Solution 1
rewrites: 74
result selectedFuncCh: SecondoryM | PrioritizedM M1 M2 M3 M3 M6 M7 M8 M9 M M10 M11 M12
M14 M16 M18 M20 M21 M31 M34
No more solutions.
rewrites: 74
```

B. Execution and Simulation

In FESoS model, missions *on-siteTreatment*, *transporting*, and *medicalSupport* are outlined, as visualized in Fig.5. These missions use specific resources, like *Ambulance* and *MedicalPersonnel*. The *on-siteTreatment* mission finishes before the other two missions which start simultaneously.

Designers, during the design phase, set parameters for each mission; for example, *on-siteTreatment* has a duration of 90 units, arrival time at 50 units, a delay of 20 units, and a resource usage time of 10 units. However, these inputs, grounded in available data and assumptions, don't fully factor in FESoS's dynamic nature or possible runtime changes. For instance, even if on-*siteTreatment* finishes punctually without delays, the following missions still wait for the preset 20 unit delay (as seen in Fig.5. Design Time). This delay essentially pushes their start to the 160-unit mark, potentially affecting their completion within the anticipated 300-unit timeframe.

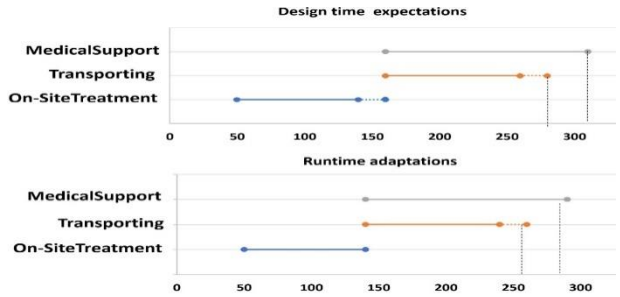

Fig. 5. Time laps during Design/Runtime of the three missions.

The gap between anticipated and realized outcomes shows a design flaw. By implementing controlled mechanisms and adaptive actions, the FESoS allows for addressing issues identified during the design phase. These corrections ensure that the three missions can be successfully accomplished within the specified time frame 300 units, Fig.5 runtime.

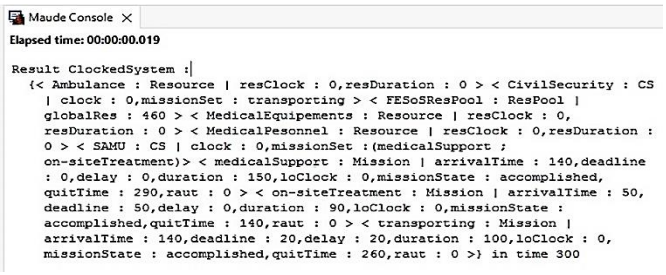

Fig. 6. Runtime execution and control.

The initial state of FESoS depicts three key missions connected to their associated CSs (*SAMU* and *CivilSecurity*) and requisite resources (*Ambulance*, *MedicalEquipments*, and *MedicalPersonnel*). This state also integrates the *FESoSResPool*, representing FESoS's current resources. While the initial design may have accounted for potential delays in the *on-siteTreatment* mission, it didn't cater to the scenario where this mission could conclude faster than expected. This oversight causes the subsequent missions to stick to a pre-established delay. In the runtime phase (Fig. 6), if *on-siteTreatment* completes without delays, its delay is recalibrated from 10 units to zero. This nimble adjustment signifies the system's responsiveness to real-time circumstances. Consequently, the *on-siteTreatment* concludes by 140, the *transporting* mission wraps up at 260, and *medicalSupport* ends at 290. Simultaneously, the resource pool's capacity swells to 430 units, with resource allocation being 10 units for *on-siteTreatment*, 30 for transporting, and 60 for *medicalSupport*. This guarantees adequate resource provision for each mission. Excess resources, 10 units from transporting and 30 from *medicalSupport*, flow back to the resource pool, bolstering its capacity to 460 units. The successful execution of this piece of code demonstrates how the control enables a centralized control, ensuring that the missions are done within the specified time.

C. Formal Verification

Thanks to *tsearch* command, we define additional RT-Maude modules to check some local/global proprieties. Due to the limited pages we are interested in checking these proprieties: (1)*isDeadlineViolated*: in the FESoS, timing planning ensures missions synchronize with operational schedules and resource requirements. Precise deadlines, combining expected mission duration and WCET, guarantee mission punctuality within the scenario. (2)*mutualExclusion*: resource constraints enforce exclusive resource use, preventing concurrent resource sharing between missions. e.g. it ensures that different CSs' Transporting missions don't access the same ambulance simultaneously. While missions can contribute resources, they only access them when not in use elsewhere. and(3) *overallMissionDuration*: the entire SoS aspires to achieve its global mission within a specified timeframe. This global mission duration ensures efficient SoS operation, meeting objectives promptly. The properties reflecting SoS designers' are formalized using invariants within. For example, the timed *tsearch* command indicates that no state where *isDeadlineViolated* (*transporting)* falls within the time frame of 240 to 260 (Fig.7) this results in a *No solution* outcome. This means the "*isDeadlineViolated*" property isn't met, and thanks to control mechanisms, any runtime delays are managed and missions are executed before their deadlines, aligning with the initial design phase.

```
(tsearch[1]{isDeadlineViolated(transporting)}=>*{isDeadlin
eViolated(transporting)} in time-interval between >= 240 and
< 260 .)
(tsearch [1] {mutualExclusion (transportingSAMU,
transportingCS, ambulance)} =>* {mutualExclusion (
transportingSAMU, transportingCS, ambulance)} in time-
interval between >= 240 and < 260 .)
rewrites: 8751 in 6374638180ms cpu (139ms real) (0 rewrites/second)

Timed search [1] in TEST-MANY-RTTS
    {isDeadlineViolated(transporting)} =>* {isDeadlineViolated(
    transporting)}
in time between >= 100 and < 200 and with mode default time increase 1 :

No solution
```

Fig. 7. Search results of Transporting mission.

V. Related Work

In this section, we review relevant approaches to our paper's proposals. In [5], the authors have employed Maude rewriting logic framework to create an executable specification that incorporates quantitative data and resource constraints. This technique allows for comparisons of provisioning strategies to enhance process efficiency. While it provides a systematic way to optimize performance through quantitative analysis, it focuses primarily on provisioning strategy comparisons and may overlook other factors impacting efficiency, like organizational structure and external temporal dependencies. Additionally, it assumes BPMN models conform to an extension with quantitative data, which may not always hold in real-world scenarios. In [6], the authors showcase the practical application of RT-Maude tool in specifying and analyzing the CASH scheduling algorithm. Utilizing RT-Maude, they define, analyze, and assess different design alterations to the algorithm. Notably, this analysis uncovers intricate behaviors within the modifications that can result in missed deadlines and aims to enhance overall system performance and meet critical activity deadlines. However, it's essential to note that the resource allocation strategy of this algorithm may not be suitable for all types of RT-systems.

In the paper [7], an approach is introduced to aid SoS architects in constructing models for SoS configurations. This approach involves adapting UPDM views and employing SysML notations to define boundaries, detail activities, organize constituents, and design configurations. It offers a systematic method for architects to design SoS configurations effectively, ensuring that the models accurately represent the configuration objectives in an SoS. In a subsequent work [8], the authors expand upon this framework by introducing reconfiguration patterns and processes. They emphasize the importance of defining constraints, employing a transition architecture, and creating iterative and recursive reconfiguration scripts. While this process provides a strong foundation, it also limits the autonomous executability and manageability of the process to specific characteristics when compared to our approach. The Dynamic-SoS approach [9] focuses on predicting and assessing the dynamic behavior of SoS architectures during the design phase, offering tools to characterize architectural changes and validate them through case studies. However, it lacks specific runtime control capabilities, missing the opportunity to employ quantitative controllers for optimizing SoS behavior during operations. On the other hand, SosADL [10] [11] presents an architectural description language based on π-Calculus with Concurrent Constraints, tailored for SoS architectures. It supports evolutionary architectures and automated verification but may benefit from further development in incorporating quantitative properties for real-world applications.

Studies [12] and [13] have introduced a core ontology for missions and capabilities in CSs, emphasizing resource-related aspects. This ontology covers a range of concept types, including physical objects, people, information, and their interactions, aiming to enhance the understanding of resource dynamics within SoSs. However, it lacks an executable environment and temporal information, limiting its capability to accurately model and analyze complex SoS interactions. In contrast, our previous work [2] proposed a comprehensive approach that leverages Maude to specify and model various missions in SoSs. This approach demonstrated Maude's robust tool support and expressive capabilities, showcasing how it can easily handle components like Missions, Resources, Roles, and their interactions with SoS management in a Direct SoS setting. The paper [14] has presented a formal model based on Maude Strategy language to handle dynamic reconfiguration in Smart SoS. The model enables the execution and analysis of Smart SoS. Maude strategies are utilized to specify dynamic reconfiguration, and their execution showcases the application of multiple crisis scenarios to achieve SoS missions. This work, although it focuses on modeling and managing SoSs and addresses same domains, it specifically don't consider the quantitative aspects of SoSs and their controlled mechanisms. The papers [15][16][17][19] have presented models to handle conflicts arising from resource consumption in SoSs. However, these models focused solely on resource consumption and did not address the production of resources at runtime.

VI. CONCLUSION

This paper presented a formal approach for analyzing and specifying temporal aspects and resource allocation in SoSs with a specific focus on their control. Firstly, the approach starts by leveraging the selecting Time-Resource strategies to select the best path that can reach the global goal of SoS. Secondly, it enables dynamic changes related to time bound missions and resource allocation, enhancing system performance and control. Lastly, the use of Maude's formal analysis language and tools adds rigor to the evaluation of model specifications, ensuring their compliance with desired properties and enabling the model checking of potential issues. Overall, the proposed approach contributed to improve the mission execution and system performance in dynamic and complex environments.

REFERENCES

[1] Maier, M. W. (1998). Architecting principles for systems-of-systems. Systems Engineering: The Journal of the International Council on Systems Engineering, 1(4), 267-284.

[2] Dridi, C. E., Hameurlain, N., & Belala, F. (2022, November). A Maude-Based Rewriting Approach to Model and Control System-of-Systems' Resources Allocation. In International Conference on Model and Data Engineering. Cham: Springer Nature Switzerland.

[3] Ölveczky, P. C. (2004). Real-time maude 2.3 manual. Research report http://urn. nb. no/URN: NBN: no-35645.

[4] Rubio, R., Martí-Oliet, N., Pita, I., & Verdejo, A. (2021). Strategies, model checking and branching-time properties in Maude. Journal of Logical and Algebraic Methods in Programming, 123.

[5] Durán, F., Rocha, C., & Salaün, G. (2021). Resource provisioning strategies for BPMN processes: specification and analysis using Maude. Journal of Logical and Algebraic Methods in.

[6] Ölveczky, P. C., & Caccamo, M. (2006, March). Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude. In FASE (Vol. 6, pp. 357-372).

[7] Petitdemange, F., Borne, I., & Buisson, J. (2018, June). Modeling system of systems configurations. In 2018 13th Annual Conference on System of Systems Engineering (SoSE). IEEE.

[8] Petitdemange, F., Borne, I., & Buisson, J. (2021). Design process for system of systems reconfigurations. Systems Engineering, 24(2).

[9] Manzano, W., Graciano Neto, V. V., & Nakagawa, E. Y. (2020). Dynamic-sos: An approach for the simulation of systems-of-systems dynamic architectures. The Computer Journal, 63(5), 709.

[10] Oquendo, F. (2016, June). Formally describing the software architecture of systems-of-systems with SosADL. In 2016 11th system of systems engineering conference (SoSE) (pp. 1-6). IEEE.

[11] Oquendo, F. (2016, November). Formally describing the architectural behavior of software-intensive systems-of-systems with SosADL. In 2016 21st International Conference on Engineering of Complex Computer Systems (ICECCS) . IEEE.

[12] Martin, J., Axelsson, J., Carlson, J., & Suryadevara, J. Towards a Core Ontology for Missions and Capabilities in SoSs.

[13] Feng, Y., Zou, Q., Zhou, C., Liu, Y., & Peng, Q. (2023). Ontology-Based Architecture Process of System-of-Systems: From Capability Development to Operational Modeling.

[14] 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. SCITEPRESS-Science and Technology

[15] 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.

[16] Maamar, Z., Faci, N., Sakr, S., Boukhebouze, M., & Barnawi, A. (2016). Network-based social coordination of business processes. Information Systems, 58, 56-74.

[17] Graiet, M., Mammar, A., Boubaker, S., & Gaaloul, W. (2016). Towards correct cloud resource allocation in business processes. IEEE Transactions on Services Computing.

[18] Roques, P. (2018). Modélisation architecturale des systèmes avec la méthode Arcadia: guide pratique de Capella. ISTE Group.

[19] Cheikhrouhou, Saoussen, et al. "The temporal perspective in business process modeling: a survey and research challenges." *Service Oriented Computing and Applications* 9 (2015): 75-8