Towards Enhancing Security and Resilience in CPS: A Coq-Maude based approach

Samir OUCHANI
LINEACT, CESI Engineering School
Aix-en-Provence, France
souchani@cesi.fr

Khaled KHEBBEB
LINEACT, CESI Engineering School
Aix-en-Provence, France
kkhebbeb@cesi.fr

Meriem HAFSI
LINEACT, CESI Engineering School
Lyon, France
mhafsi@cesi.fr

***Abstract*—Cyber-Physical Systems (CPS) have gained considerable interest in the last decade from both industry and academia. Such systems have proven particularly complex and provide considerable challenges to master their design and ensure their functionalities. In this paper, we intend to tackle some of these challenges related to the security and the resilience of CPS at the design level. We initiate a CPS modeling approach to specify such systems structure and behaviors, analyze their inherent properties and to overcome threats in terms of security and correctness. In this initiative, we consider a CPS as a network of entities that communicate through physical and logical channels, and which purpose is to achieve a set of tasks expressed as an ordered tree. Our modeling approach proposes a combination of the Coq theorem prover and the Maude rewriting system to ensure the soundness and correctness of CPS design. The introduced solution is illustrated through an automobile manufacturing case study.**

***Index Terms*—Cyber-Physical Systems, Security, Resilience, Rewriting Logic, Coq, Maude.**

I. Introduction

Designing modern systems, i.e. *systems-of-systems* and/or *Cyber-Physical Systems* (CPS) is challenging due to many requirements and difficulties, mainly: scalability, security, and resiliency. The latter is a system property that enables a system/system-of-systems to continue providing useful services in the face of disruptions, parasites, and attacks. Resilience is especially important for systems such as CPS that have to operate for extended periods in uncertain and disruptive environments. Unfortunately, the deployed resilience approaches tend to be ad hoc and do not scale. Importantly, it is difficult to assess their long-term impact since they are more static in general, and do not deal with the dynamic behavior of complex systems, like CPS.

In a dynamic an open environment, CPS are subject to physical faults, network failures and cyber-attack. Indeed, the resiliency of CPS should take into consideration at least, CPS stability, security, and systematicity. Stability means that the CPS can achieve a stable sensing-actuation close-loop control even though the inputs (sensing data) have noise (fault-tolerance) or Byzantine fault. Security means that the system can overcome the cyber–physical interaction attacks (internal and external) and Systematicity means that the system has a seamless integration physical components and updating software. However, from the literature, many techniques and approaches are used to design, validate, and analyze large scale systems. From a design perspective, algebraic specification is widely used since many tools support it and solid formal foundations are dedicated to it. Further, it helps to prove mathematically the soundness, the correctness, and the completeness of any analysis/verification/validation algorithm on the the designed CPS.

In this paper, we rely on two formal approaches, Coq[1] and Maude [1], to exploit the scalability of the first and the automation of the second. Coq is a proof assistant based on higher-order logic and λ-calculus powered with an extension of primitive recursion to check the correctness and build proofs using specifics programs (*i.e.* tactics). Maude is a high-level language and a high-performance interpreter that supports membership equational logic and rewriting-logic-based specification and programming of systems. Maude system (language and tools) is chosen as an implementation support platform of the model for several reasons. Its language is expressive enough to capture all the aspects of CPS required to reason on their structure and behavior. It also offers several rewriting-based tools that are adequate for simulating the specification, and for conducting formal verification of behavioral correctness.

From a practical point of view, to secure a CPS against potential sources of performance degradation or attacks exploiting the system attack surfaces, the CPS design should be considered first, then the CPS deployment and execution should be controlled and monitored. This paper considers both issues by initiating the needed fundamentals to design a correct CPS, and to verify its functionality. Furthermore, it discusses the CPS security and resiliency. The proposed approach, named Sec-Res-CPS, contains three main parts: (1) CPS design using an algebraic specification, (2) CPS correctness and validation using the proof assistant Coq, (3) fault diagnostics using Maude. In addition, we discuss security requirements, cyber-attacks, and failure/countermeasures resilient strategies for the designed CPS. Cyber-attacks and network failure resilient strategies act as a state machine system with several strategies and guards designed for specific cyber-attacks or network failure. Hence, decision making algorithms have to apply the best strategies for the existing cyber-attack or network failure through a model predictive control (monitor) that chooses the best available control action while guaranteeing a smooth behavior of CPS. This work focuses mainly on the following folds.

[1] https://coq.inria.fr/

- An automatic framework Sec-Res-CPS that analyzes and guarantees CPS behavioral correctness.
- A formalism that models large scaled CPS.
- Initiate security threats and system faults including attacks, weaknesses, vulnerabilities and countermeasures.

II. Related Work

This section reviews and discusses approaches that deal with modeling, functional and security analysis, as well as the resiliency aspect for CPS.

Aguida *et al.* [2] provide a holistic view of the initiatives done during the last five years in modeling and designing CPS. They define and survey the architecture and the design of three classes of CPS: smart healthcare, smart manufacturing, and smart city. Cheh *et al.* [3] rely on hybrid automata to analyze the safety of railway systems corresponding to different classes of venomous actors based on to their abilities defined in the system access control. Ouchani [4] uses the probabilistic model checker PRISM to analyze the functional correctness of IoT in an open environment.

Bakirtzis *et al.* [5] evaluate the security level of CPS at each step of their life cycle development with the focus on deploying and operating safety-critical applications. Chen *et al.* [6] studied the detectability of data deception in CPS by assuming that an attack detector has access to a linear function of the initial state of the system and it can not be changed by any attacker. Mitchell and Chen [7] used stochastic Petri nets to develop an analytical model that can capture the interactions between the adversary and the defense models of CPS. Agrawal *et al.* [8] try to design a secure CPS by relying on two categories of attacks. The external attacks were considered as a threat model, which might not be able to connect to an insider threat with the physical access to a CPS.

Ouchani [9] reinforces CPS against attacks that can be either technical or socio-technical by constraining a CPS through security policies expressed as a temporal logic formulae. Stoicescu *et al.* [10] identify a change model and establish a reference frame in which the dynamics faced by fault-tolerant systems can be illustrated. Then, the execution of the deployed protocols is broken down in three steps corresponding to their variable features, then mapped into a component-based middleware. Finally, they evaluate the transition execution time and the agility with respect to the fine-grained adaptive fault tolerance. Pedroni [11] manages the vulnerabilities of safety-critical systems by considering the external natural events, integrating vulnerabilities exposed to cyber attacks, and analyze risks from heterogeneous contributors in different locations for their aggregate evaluation. Fang *et al.* [12] try to assess the resilience of interdependent critical infrastructure systems under potential disruptive events using the game-theoretic attacker-defender and defender-attacker-defender modeling techniques. They advise using policy-makers on making pre-disruption decisions to improve the resilience of interdependent infrastructures. Saurabh *et al.* [13] structure the management of a system resilience by relying on resilience-based design patterns at various layers of the system stack. The patterns coordinate flexible fault management across hardware and software components. It also enables optimization of the cost-benefit trade-offs among performance, resilience, and power consumption.

With respect to the surveyed approaches, the proposed solution initiates an automatic analysis and validation of CPS. At this level, we focus more on the correctness of the CPS design in order to provide a solid foundation for a smart and automatic recovery and countermeasures that are briefly discussed in this paper.

III. Sec-Res-CPS Framework

Sec-Res-CPS framework proposes to models a system then analyze and harden its security to ensure its resiliency. A given system is considered as a composition of different kinds of entities where each entity has its proper structure and behavior. The entities can communicate and interleave in different environment. Sec-Res-CPS models the main components of a system, and rely on a library of attacks that can manipulate and harm the system. It then develops a set of counter measures that harden the system and counterfeit the attacks. Furthermore, the framework analyzes the vulnerabilities of a system, and reports the errors and counterexample. Finally, it generates the safe code. The considered part in this paper from the overall framework is depicted in Fig. 1.

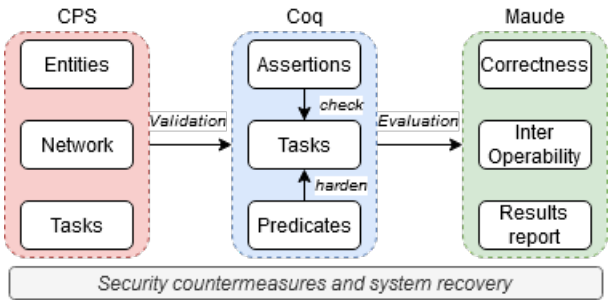

Fig. 1. The Interoperability and Integrity Validation and Evaluation

Sec-Res-CPS develops four stages: modeling (red rectangle), analysis (blue rectangle), and implementation (green rectangle). Herein, we explain each part and steps of Sec-Res-CPS. Initially at the modeling stage, $\mathcal{S}$ models a given system, $\mathcal{A}$ and $\mathcal{CM}$ are sets of attacks and countermeasures proper to $\mathcal{S}$, respectively. Then in the second stage of composition, the function Υ generates the potential attacks $\mathcal{A}_{\mathcal{S}}$ from $\mathcal{A}$ for $\mathcal{S}$, and the function $\Psi_{\mathcal{A}}$ selects from $\mathcal{CM}$ the countermeasures $\mathcal{CM}_{\mathcal{S}}$ of $\mathcal{S}$ with respect to the selected attacks $\mathcal{A}_{\mathcal{S}}$. Also, the function $\Gamma_{\mathcal{CM}}$ composes $\mathcal{S}$ with $\mathcal{A}_{\mathcal{S}}$ to produce a malicious system $\mathcal{S}_{\mathcal{A},.}$, then the function $\Gamma_{\mathcal{CM}}$ hardens $\mathcal{S}_{\mathcal{A},.}$ with $\mathcal{CM}_{\mathcal{S}}$ and produces $\mathcal{S}_{\mathcal{A},\mathcal{CM}}$. Further in the third stage of analysis, Ξ produces $\mathcal{R}_{\mathcal{A}}$ and $\mathcal{R}_{\mathcal{CM}}$ that show respectively the effect of $\mathcal{A}$ and $\mathcal{CM}$ on $\mathcal{S}$. Finally a last phase, Ξ generates the safe implementation code $\mathcal{P}$ of $\mathcal{S}$.

IV. Resilient CPS

We consider a system $\mathcal{S}$ as a composition of entities $\mathcal{E}$ that interact and interleave through a network of physical and logical infrastructure ($\mathcal{N}$) to accomplish a given task ($\mathcal{T}$). Formally, a system $\mathcal{S}$ is a tuple $\langle \mathcal{E}, \mathcal{N}, \mathcal{T} \rangle$.

A. Modeling the entities

An entity ε can behave independently executing sequentially or deterministically (or not) the allowed actions, or on harmony with other entities to form a system executing a global task. $\mathcal{E}$ is the main entities describing CPS, and it is defined by the tuple $\langle id, attr, Actuator, \Sigma, Beh\rangle$, where:

- id is a finite set of tags,
- $attr : id \to 2^{\mathcal{T}}$ returns the attributes of an entity,
- $Actuator$ specifies the status of an entity by evaluating its attributes,
- $\Sigma = \{\texttt{Send}, \texttt{Receive}, \texttt{Update}\}$ is a finite set of atomic actions that can be executed by an entity,
- $Beh\colon id \times \Sigma \to \mathcal{L}$ returns the expression written in the language $\mathcal{L}$ that describes the behaviour of an entity. The syntax of $\mathcal{L}$ is given by: $B ::= \alpha \mid B \cdot B \mid B +_g B$, where $\alpha \in \Sigma$, " $\cdot$ " composes sequentially the actions and $+_g$ is a guarded choice decision that depends on the evaluation of the guard, a propositional formulae, g. When $g \triangleq \top$ the guarded decision become a non deterministic choice.

Example 1. Fig. 2 represents the automation process of cars assembly industry. We will consider this example through all the paper to explain the proposed approach steps.

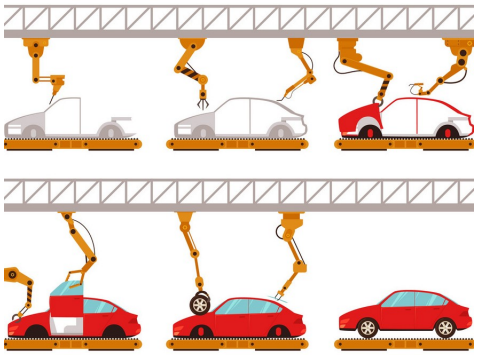

Fig. 2. Smart Robotic Automotive Assembly.

First, we define the entities $\mathcal{E} = \{\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{30}\}$ where ε_0 is the environment, such that:

- ε_1 is a robot arm using the tool ε_2 to place the front left door ε_3 on the cage ε_4. The behavior of the arm ε_1 is given as follows. The decision guard specifies the equality equation between the current position of the arm and where the left door should be placed.

$$Beh(\varepsilon_1) \triangleq \begin{array}{l}\texttt{Receive}(\varepsilon_2, \varepsilon_0) \cdot \texttt{Receive}(\varepsilon_3, \varepsilon_0) \\ \cdot(\texttt{Update}(\varepsilon_5, \varepsilon_6) +_{pos(\varepsilon_4)=val(\varepsilon_5)} \texttt{Send}(\varepsilon_3, \varepsilon_4))\end{array}$$

- The same description of ε_1 is applicable on the other three arms to place different parts. Regarding the painting task, it is slightly different since the color type and the used volume can be changed and repeated for each part. The following describes the behaviour of a painting arm.

$$Beh(\varepsilon_1') \triangleq \begin{array}{l}\texttt{Receive}(\varepsilon_2', \varepsilon_0) \cdot \texttt{Receive}(\varepsilon_3', \varepsilon_0) \\ \cdot((\texttt{Update}(\varepsilon_5', \varepsilon_6') +_{pos(\varepsilon_4)=val(\varepsilon_5)} \texttt{Send}(\varepsilon_3', \varepsilon_4')) \\ +_{size(\varepsilon_3')\leq val} \texttt{Receive}(\varepsilon_3', \varepsilon_0)))\end{array}$$

- We mention that fixing the wheels ε_{25}, ε_{27}, ε_{29}, ε_{31} is done by the arms ε_1, ε_7, ε_{13}, ε_{19} with a similar behavior as placing doors.

B. Modeling the Network

The network $\mathcal{N}$ defines how the entities are connected and communicate. An entity ε_i can be connected to another one through a physical or logical channel for communication or to a subsystem. In $\mathcal{N}$, we classify the communication protocols into four categories according to their rates and uses. Moreover, entities are also used to receive data from external sources, ranging from a USB flash drive plugged into the media player to online services granted through mobiles range communications. We define a network $\mathcal{N}$ as a graph where vertices are the entities and the edges are the way that they interact and connected $\mathcal{N} = \langle \mathcal{E}, \texttt{Chan}, \texttt{Prot}, \texttt{Rel}\rangle$, where:

- Chan is a finite set of channels,
- Prot is a finite set of protocols where $\epsilon_{\texttt{Prot}}$ is the empty protocol.
- $\texttt{Rel} : \mathcal{E} \times \mathcal{E} \to \texttt{Chan} \times \texttt{Prot}$ relies two entities with a channel and a protocol. When $\epsilon_{\texttt{Prot}}$ is assigned, it means both nodes are physically connected.

Fig. 4 shows a physical relation between ε_i and ε', and a logical relation through the protocol $Prot$ between the entities ε_i and ε.

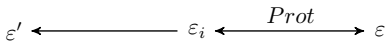

Fig. 3. A Network of logical and physical relation between the entities.

Example 2. Fig. 4 depicts how the entities of the system presented in *Example* 1 are connected through physical and logical channels as well communicate via the appropriate protocols. It defines the relations between arms ε_1 and ε_1 as well between them the environment. In addition, the relation with their used tools, painting and assembly. Further, the server entity ε_0' communicates with the arms using the protocol P′ whereas the arms evaluate their control variables ε_6 using the protocol P′.

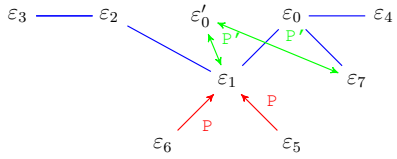

Fig. 4. The Network of the Entities in Example 1

C. Modeling the Tasks

The task $\mathcal{T}$ is the main goal of the system. It describes the sequence of actions that should be realized by each entity. We define a task by a tree where the root represents the main goal of the system $\mathcal{S}$, the children are sub-goals of the entities, and leafs are the final product for each entity. The task $\mathcal{T}$ is the tuple $\langle \texttt{Goals}, \preceq\rangle$, where:

- Goals is a finite set of goals where $\texttt{G} \in \texttt{Goals}$ is the root (the main goal),

- $(\texttt{Goals}, \preceq)$ is a preoder relation on Goals.

Example 3. Fig. 5 depicts the task tree proper to the model presented in *Example* 2. The red node is the global task, blue nodes are the sub-tasks whereas the green nodes are the final products for each entity. In Figure 5, T means car assembly, T_3 is dealing with a door, T_4 is fixing a door, and T_5 is painting a door.

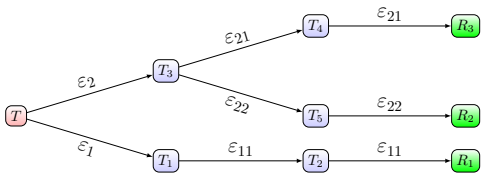

Fig. 5. Tasks Tree.

V. Security Threats

Resilient CPS should guarantee the following properties:

- *Confidentiality* where every message sent, a malicious can not eavesdrop it.
- *Authenticity* means that a sent message should be sent only by the authorized entity.
- *Availability* should assure that any entity can execute its behaviour.
- *Integrity* means an entity, physical or logical, could not be modified
- *Non-repudiation* means that an identified entity could not prove that it has not executed an action.

Since current networks are unable to guarantee these security properties, we need to analyze if the corresponding vulnerabilities can be exploited through different possible attacks, especially:

- *Local attacks*. They can be done physically by plugging an additional device on the targeted bus or the on board diagnostics port.
- *Remote attacks*: These attacks can be classified through different ranges: indirect physical access, short range wireless access and long range wireless access.
- *Indirect access*: They rely on a compromised third-party device which will later be connected to an entity through an attack surface. If a physical connection to the network is in required, it will be composed with a *local attack*.
- *Short range attacks*: These attacks use short-range wireless communication technologies through a smart phone or a connected object, for example.
- *Long-range direct attacks*: They use long-range wireless communication technologies like telephony and web browsing.
- *Long-range indirect attacks*: This class uses a long-range transmission channel to compromise a third party device, like side channel triggers.

Sec-Res-CPS relies on temporal logic and high order to express the security properties and tree to model the attacks (attack trees).

VI. CPS Recovery

Before a system recovery, a protection must be taken into account while designing CPS. The following are the protection mechanisms adopted by Sec-Res-CPS.

- *Constraints*: It can be a hardware (gate), software (access control), or a physical entity (lock). Sec-Res-CPS expresses the constraints as propositional logic formula.
- *External communications protections*: They are mechanisms to protect CPS from external attacks, like firewall. These mechanism are modeled as monitors where their semantics are equivalent the the semantics of the entities.
- *Internal protections*: They are solutions to how authenticate, lock or encrypt the physical/logical entities.

The above mechanisms should ensure the satisfiability of the CPS correctness and properties. When a fault or an attack occurs, the recovery mechanism is to improve the corrections and the countermeasures. Both of them are modeled as a game between two players, the proponent and the opponent, and represent the scenario of Attack-Defense and Fault-Correction Trees.

VII. CPS Validation with Coq

A Coq program is a sequence of declarations and definitions where each declaration associates a name with a qualification. It supports few primitive constructions (functions, (co)-inductive definitions, product types, sorts) and a limited number of rules for type-checking and computation. Coq is used to represent objects, functions, propositions and proofs. Further, it helps designing theories and proofs offering mechanisms like user extensible notations, tactics for proof automation. In the following, we provide in Coq the basic notions, definition, tactics proper to CPS defined in Section IV.

- The entities are defined as an inductive type.

```
Inductive Entity : Set := | CreateEntity :
nat -> Entity | Empty : Entity.
```

- The channels, logical or physical, define the network by connecting the entities. Each channel is an inductive type, ChannelType, describing the type/directions of the channel.

```
Inductive ChannelType : Set := | Phy: ChannelType
          |LogUnid:ChannelType |LogBid:ChannelType.
Definition Channel :
Set:= ChannelType * nat * Entity * Entity.
```

- The network describes all channels/connections between the entities.

```
Definition Network: Set :=(list Entity)*(list Channel)
                          *(list Entity).
```

- A CPS is defined as follows.

```
Record CPS : Type := myCPS {
            states : Set ; actions : Set ;
            net : Network ; transitions : Set}.
```

- The behaviour of an entity depends on the expression reordering its actions. Here, we define by induction the behaviours of the defined actions with respect to the connection operators (sequential and guarded decision operators).

```
Inductive alpha : Set := send | receive | update | Seq| Dec.
Fixpoint s1 (xs : list alpha) : bool :=}
  match xs with}
    | send :: rest => s2 rest} | receive :: rest => s2 rest}
    | update :: rest => s2 rest} | => false}
```

```
  end}
  with s2 (xs : list alpha) : bool :=}
  match xs with}
    | nil => true}
    | seq :: rest => s2 rest}  | => false}
 end}
 with s3 (xs : list alpha) : bool :=
  match xs with
    | Dec :: rest => s2 rest |  => false
  end.
```

Example 4. Based on the defined CPS in coq, the following fragment is a part of the generated Coq code proper to *Example* 1.

```
  Module Type types.
    Parameter P PI: Type.
  End types.

Inductive state : Type := BeforeAction  :
     state | AfterAction : state.
Definition next_state (s:state) : state :=
    match s with
      | BeforeAction => AfterAction
      | AfterAction => AfterAction
    end.
Eval simpl in (next_state BeforeAction).
Variable Entity : Type.
Variable EntityArms : Entity. Variable EntityObject : Entity.
Variable EntityEnvironment : Entity.
Definition relation := Entity -> Entity -> Prop.
Variable R : relation.
Definition Receive : Prop := forall a b c
    : Entity , R a b  /\  ~R c b -> ~R a b  /\  R c b.
```

The following theorem validates the correctness of the developed design.

```
Lemma ReceiveProf : forall a b c : Entity ,
    (R a b  /\  ~R c b) -> (~R a b  /\  R c b).
```

VIII. CPS Evaluation with Maude

In this section, we first give an overview of the Maude framework and rewriting logic. The Maude specification of CPS is then described through the implementation of the generic CPS model introduced in Section IV, and explained through the cars manufacturing case study (*Example* 1).

Maude [1] is a high-level formal specification language based on rewriting and equational logic. A Maude program defines a logical theory which can be seen as a domain-specific language and a Maude computation implements a logical deduction using axioms specified in the theory. A Maude-based specification is structured in two parts. (1) *functional modules* specifying a theory in *membership equational logic* as a pair $(\Theta,\ E \cup A)$ where signature Θ specifies the type structure (sorts, operators etc.). E is the collection of possibly conditional equations, and A is a collection of equational attributes for the operators (i.e., associative, commutative, etc.). (2) *system modules* specifying a rewrite theory as a triple $(\Theta,\ E \cup A, R)$ where $(\Theta,\ E \cup A)$ is the equational theory part and R is a collection of possibly conditional rewrite rules.

To implement the example of car assembly CPS, we define a Maude functional module *CPS-SPEC* to design the system structure: the set of entities (car elements and robotic assembly arms) and their attributes. together with monitoring predicates and actions (setters Σ and getters $Actuator$) to be applied over it (i.e., to retrieve/update attributes $attr$).

The entire system $\mathcal{S}$ is expressed in Maude as a sort `SYS` according to its associated constructor: `op SYS{_}: EL -> SYS [ctor]` where `EL` is a sort expressing a set of entities $\mathcal{E}$. An entity ε is expressed as a sort `E`. However, in the Maude specification we distinguish the types of entities through different sorts as the entities play different roles : a car frame (sort `F`), a door (`D`), a wheel (`W`) and a robotic arm (`A`). Every sort has its specific constructor but are all subsorts of the generic entity sort (`subsorts F D A W < E`).

This distinction is necessary for three reasons It allows : (1) the Maude interpreter recognizing the nature of an entity, (2) specifying generic actions (such as `getId(`ε`)`) that can be implemented (i.e., override) accordingly to all specific sorts (as their structural constructors differ) and (3) scaling up and extending the specification while minimizing the risks of regression. These points ultimately allow restricting the possible actions to the only concerned entities. It would for example prevent a robotic arm from grabbing another arm. Or more generally, all entities will simply ignore the entities they are not designed to interact with (even when scaling up).

Notice that the concept of network ($\mathcal{N}$) is integrated implicitly into the Maude specification. In fact, different entities are linked if mentioned in a common action or if their identifiers (id) appear in each other's structure. For example, a link is implicitly expressed between an arm `a` and a door `d` with the operation `setAttch(a,d)` which attaches `d` to `a`.

To specify the CPS behaviors, we define a Maude system module *CPS-BEH* to model the entire system behavior. We define a set of guarded rewrite rules R expressing the actions to be applied in order to achieve the system's purpose $\mathcal{T}$. In other terms, $R \equiv \mathcal{T}$. A conditional (guarded) rewrite rule (`crl`) named `r` is given with `crl [r]: term => term' if (condition).` The semantics of (`crl`) is rewriting or re-expressing (`=>`) the left-hand side of the rule (`term`) to its right-hand side (`term'`), where `term` and `term'` are structural states (i.e., configurations) of the system or a part of the system that have the same sort $\in \Theta$ (i.e., out of `SYS, E`, etc.). A conditional rewrite rule is applied only if the specified guard `condition` is satisfied.

Example 5. We give the `crl` describing the task of "grabbing a wheel" (`gw`) from the CPS car assembly use case :

```
crl[gw]:SYS{ f + frd + fld + brd + bld + ga + pa + wl }
  => SYS{ f + frd + fld + brd + bld
    + setBusyT(ga) + pa + getW(ga,wl) }
  if not isBusy(ga) and not isBusy(pa)
    and getW(ga,wl) =/= null  and isMounted(f)
    and isPaint(f) and not wheelsOK(f) .
```

In `gw`, the entire system structure `SYS{`$\varepsilon_i + ... + \varepsilon_n$`}` is expressed with variables of the suitable sort: the left side shows the frame `f`, the front/back right/left doors `(f/b)(r/l)d`, the painting/grabbing arms `(p/g)a` and a set of wheels `wl`. The right side specifies how the system is rewritten to express the action of a wheel being grabbed. The action `getW(ga,wl)` expresses the arm `ga` grabbing a wheel from the set of wheels `wl`, which will set a wheel as "attached to `ga`". The state of `ga` is also affected as it is marked as busy with `setBusyT(ga)`, which makes `gw` $\in Beh($`ga`$)$.

The rule `gw` is applied only if its specified guard is satisfied. The guard expresses the conjuction of different conditions: the grabbing arm needs to be free (`not isBusy(ga)`) to initiate the operation. The painting arm also needs to be free

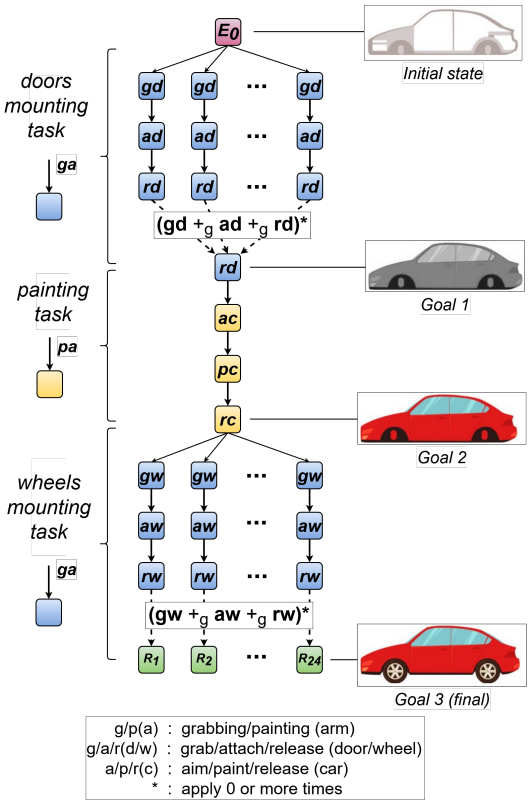

Fig. 6. Automatic car assembly task tree

to prevent any collisions. `getW(ga,wl) =/= null` is the post-condition of `getW` which has to return a wheel ($\neq$`null`). `isMounted(f)`, `isPaint(f)` and (`wheelsOK(f)`) are predicates expressing the state of `f` and therefore the different `Goals` of the system tasks $\mathcal{T}$. Moreover, this integrates the preorder relation ($\preceq$) on the goals (thus on the tasks: mounting doors $\preceq$ painting $\preceq$ mounting wheels).

Figure 6 shows the system task tree resulting from the behavior simulation from a naked car frame as initial state (E_0). The tree shows the tasks, their goals (in order) and the different arcs (as sequences of rules) leading to the goals. In the doors mounting task, the different paths show the possibility to attach doors in any order (but each door at a specific emplacement). However, the wheels mounting task produces 24 possible and correct outcomes (as any wheel can be attached at any position: $4! = 24$). The simulation is made through the Maude *search* command : `search INIT =>* s:SYS such that isComplete(s:SYS).`

Where all the possible paths of execution are explored from a predefined initial state `INIT` of sort `SYS`, in 0 or more steps (`=>*`), in search for a state `s` of sort `SYS` satisfying the predicate `isComplete(s)` which here is our final goal (i.e., a completely mounted and painted car). Our Maude specification of the car assembly CPS is available online[2].

As the system states can be expressed as predicates, notice that Maude also allows conducting a LTL state-based model-checking verification technique to (1) show the reachability of the desired state and (2) diagnose malfunctions by calculating counter-examples. Maude model-checking methodology is described in [14].

[2]https://github.com/aanorlondo/MAUDE-CPS

IX. Conclusion

In this paper, we have presented a first step towards a framework to automate the analysis of the functional correctness and security insurance for a more resilient `CPS`. The approach defines a system as a network of entities whereas each entity has its proper structure and behaviour to execute a precise task. The overall goal of the system is structured as an ordered tree of tasks. To ensure the functional correctness and safety of a `CPS`, the framework proposes using Coq theorem prover and provides a rewriting logic specification, through the language Maude in order to verify the `CPS` liveness property. We illustrated the effectiveness of the proposed framework on an automated car assembly case study.

As future work, We intend to extend the framework capabilities to (1) cover the agent operators, (2) decentralize the system with a blockchain architecture, and (3) detail the proposed formalism by applying the framework on more complex use cases.

References

[1] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, *All About Maude-A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic*. Springer, 2007, vol. 4350.

[2] A. Mohamed Anis, O. Samir, and B. ourad, "A review on cyber-physical systems: Models and architectures," in *The 29th IEEE Int. Conf. on Enabling Technologies: Infrastructure for Collaborative Enterprises*, ser. WETICE'20, 2020.

[3] C. Cheh, A. M. Fawaz, M. A. Noureddine, B. B. Chen, W. G. Temple, and W. H. Sanders, "Determining tolerable attack surfaces that preserves safety of cyber-physical systems," *2018 IEEE 23rd Pacific Rim Int. Symposium on Dependable Computing (PRDC)*, pp. 125–134, 2018.

[4] S. Ouchani, "Ensuring the Functional Correctness of IoT through Formal Modeling and Verification," in *The 8th Int. Conference on Model and Data Engineering, MEDI, LNCS, Springer*, 2018, pp. 401–417.

[5] G. Bakirtzis, B. T. Carter, C. R. Elks, and C. H. Fleming, "A model-based approach to security analysis for cyber-physical systems," in *2018 IEEE International Systems Conference (SysCon)*, pp. 1–8.

[6] Y. Chen, S. Kar, and J. M. F. Moura, "Dynamic attack detection in cyber-physical systems with side initial state information," *IEEE Transactions on Automatic Control*, vol. 62, no. 9, pp. 4618–4624, 2017.

[7] R. Mitchell and I. Chen, "Modeling and analysis of attacks and counter defense mechanisms for cyber physical systems," *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 350–358, 2016.

[8] A. Agrawal, C. M. Ahmed, and E.-C. Chang, "Poster: Physics-based attack detection for an insider threat model in a cyber-physical system," in *Asia Conference on Computer and Communications Security*, ser. ASIACCS '18. ACM, 2018, pp. 821–823.

[9] S. Ouchani, "Towards a security reinforcement mechanism for social cyber-physical systems," in *Smart Applications and Data Analysis*. Springer, 2020, pp. 59–73.

[10] M. Stoicescu, J.-C. Fabre, and M. Roy, "Architecting resilient computing systems: A component-based approach for adaptive fault tolerance," *Journal of Systems Architecture*, vol. 73, pp. 6 – 16, 2017.

[11] N. Pedroni, "Advanced Methods For The Risk, Vulnerability And Resilience Assessment Of Safety-Critical Engineering Components, Systems And Infrastructures, In The Presence Of Uncertainties," Ph.D. dissertation, Grenoble 1 UGA - Université Grenoble Alpes, Feb 2016.

[12] Y. Fang and E. Zio, *Game-Theoretic Decision Making for the Resilience of Interdependent Infrastructures Exposed to Disruptions*. Springer, 2019, pp. 97–114.

[13] H. Saurabh and E. Christian, "Resilience Design Patterns: A Structured Approach to Resilience at Extreme Scale," *Super computing Frontiers and Innovations*, vol. 4, no. 3, Sep 2017.

[14] K. Khebbeb, N. Hameurlain, and F. Belala, "A maude-based rewriting approach to model and verify cloud/fog self-adaptation and orchestration," *Journal of Systems Architecture*, p. 101821, 2020.