Construction of consistent SysML models applied to the CPS

ADEL KHELIFATI, MOVEP, Faculty of Computer Science, USTHB university, Algeria
MALIKA BOUKALA-IOUALALEN, MOVEP, Faculty of Computer Science, USTHB university, Algeria
AHMED HAMMAD, Franche-Comté University, FEMTO-ST Institute, UMR CNRS 6174, France

With the increasing complexity of cyber-physical systems (CPS), it is interesting to decompose a CPS into sub-systems. This provides greater modularity and flexibility so that each system can be developed independently, making it easier to maintain. Also, it can improve its fault tolerance. However, this decomposition of the system can lead to inconsistency. This paper proposes an approach for early verification of cyber-physical systems decomposition using SysML. We address the limitations of SysML as a semi-formal language by introducing syntax and static semantics for its structural diagrams. The aim is to verify structural consistency before defining behavioral aspects. For that, the proposed approach verifies a set of structural consistency rules within a refinement relation to ensure that sub-components offer at least the same services as the abstract block and require the same services. Furthermore, the sub-blocks must satisfy all the requirements that the abstract block is supposed to verify. We used the CyCab as a case study to demonstrate the effectiveness of this approach.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**.

Additional Key Words and Phrases: Cyber-Physical System, SysML, Components, Early verification

1 INTRODUCTION

Cyber-Physical Systems (CPS) present significant challenges due to their inherent complexity, which arises from the integration of computational (cybernetic) and physical components. These systems exhibit both structural and behavioral characteristics, as well as functional and non-functional properties. To manage this complexity effectively and reduce associated costs, a top-down modeling approach is often employed. This approach involves starting with an abstract block and systematically decomposing it into smaller, more manageable sub-blocks.

Decomposing a CPS into sub-systems offers several key benefits. Firstly, it enhances modularity, allowing different components of the system to be developed, tested, and maintained independently. This modularity simplifies the management of complex systems and enables the reuse of sub-systems across different projects. Secondly, it increases flexibility by allowing specific parts of the system to be modified or upgraded without affecting the overall system architecture. However, the decomposition process must be guided by consistency rules to ensure the integrity of the system. These rules can be considered from multiple perspectives, including structural or behavioral aspects, as well as inter-model or intra-model consistency [12]. Ensuring consistency during decomposition is critical to maintaining the system's reliability and functionality as the design progresses from abstract models to detailed implementations.

Given that the specification of system behavior is complicated and time-consuming[17], performing an early verification by assessing a system's structural consistency that ensures system components' logical soundness

Authors' addresses: Adel Khelifati, MOVEP, Faculty of Computer Science, USTHB university, Algiers, Algeria, akhelifati@usthb.dz; Malika Boukala-Ioualalen, MOVEP, Faculty of Computer Science, USTHB university, Algiers, Algeria, mioualalen@usthb.dz; Ahmed Hammad, Franche-Comté University, FEMTO-ST Institute, UMR CNRS 6174, Besançon, France, ahmed.hammad@femto-st.fr.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2024 Copyright held by the owner/author(s).
ACM 1550-4840/2024/10-ART
https://doi.org/10.1145/3702326

Just Accepted

and interactions from the initial stages before embarking on detailed behavioral design can significantly optimize development time and cost. Such verification is essential for guaranteeing system correctness and mitigating the risk of costly rework during later development stages.

The industry-standard SysML [15] language offers distinct advantages for modeling Cyber-Physical Systems. It facilitates the creation of comprehensive models, including a system's cyber and physical aspects, aligning seamlessly with the Model-Based Systems Engineering (MBSE) approach. However, SysML's characterization as a semi-formal language presents a significant limitation. A well-defined formal semantics is necessary for the ability to perform early verification of structural consistency within SysML models.

This research aims to bridge the critical gap in the early verification of CPS design by proposing a formal specification of SysML structural diagrams. We achieve this by providing a formal syntax and static semantics for these diagrams. This formal foundation allows the application of robust verification techniques during the early stages of CPS design, enabling the systematic assessment of a system's structural consistency.

To formally define the structural components of SysML, we have used the OCaml [10] language, which is a functional programming language. OCaml's strong static type system ensures that many errors are caught at compile-time rather than at runtime. This feature is particularly beneficial for formal verification as it enhances the reliability and robustness of the model. The type system helps to enforce consistency rules and invariants within the model, aligning well with our goal of verifying structural consistency in SysML models. In addition to its powerful features, OCaml maintains simplicity and readability [14]. Its syntax is designed to be clear and concise, which aids in understanding and maintaining the formal specifications. This simplicity is crucial for making our approach accessible and easily understandable to other researchers.

This document is structured as follows: Section 2 provides a preliminary to SysML and OCaml. Section 3 presents our approach to constructing a consistent SysML model. Section 4 introduces the syntax and static semantics of SysML specifications. Section 5 shows our tool for transforming the XMI format into the proposed formal specification. Section 6 demonstrates the applicability of our approach to a case study. Section 7 discusses the applicability of our approach with SysML v2. Section 8 discusses related work and our contributions. Finally, we conclude this paper in section 9.

2 PRELIMINARIES

This section covers the fundamentals of the SysML and Ocaml languages, which we will use to model and checking the structural consistency of our system.

2.1 SysML

SysML is a graphical modeling language used in systems engineering. It is based on the Unified Modeling Language (UML) [16] and is maintained and developed by the Object Management Group (OMG). In our work, we use version 1.6 of SysML. In the following, we will now introduce some components and diagrams of SysML that we used in our approach.

Block: A block is a basic unit of modeling that represents a physical or logical entity in a system. Moreover, we distinguish two types of blocks: a composite block, which is composed of other blocks, and an elementary block, which is not composed of any other blocks.

Interface block: An interface block is a contract between two systems, specifying what one system expects from the other and what the latter provides. It also defines the port type.

Proxy port: Proxy ports are always typed by interface blocks, which define the boundary by specifying the visible features of the owning block through external connectors. These features typically include the services that the block either provides or requires. A service represents a specific function or capability that an interface block offers to its environment or expects from other blocks.

Just Accepted

Block Definition Diagram (BDD): Allows to describe the system's structure by showing the hierarchical relationships between its blocks.
Internal Block Diagram (IBD): An Internal Block Diagram (IBD) is a graphical representation used in SysML to show the internal structure of a system or a block within a system. It helps to model the relationship and composition between various parts or components of a system. The IBD consists of two types of connectors: delegation connectors which link the block's ports with the internal parts, and composition connectors which link the internal parts together. Figure 1 illustrates this concept. There are also two types of delegation connectors: input delegation connectors, whose source port is the input port of the abstract port, and output delegation connectors, whose target port is the output port of the abstract port.

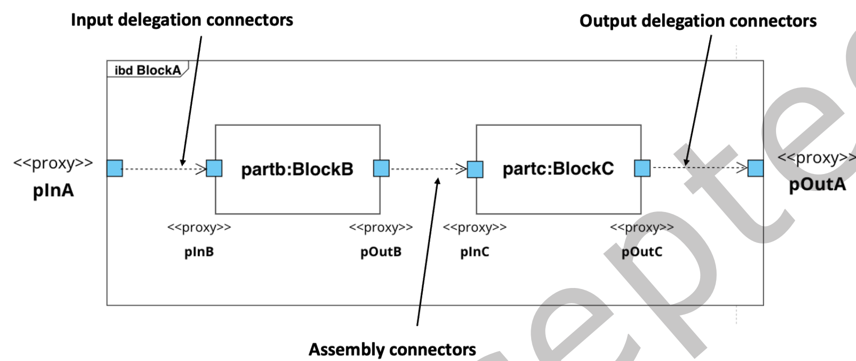

Fig. 1. Internal Block Diagram connectors

Requirement diagram: Requirement diagrams are used to describe system requirements. They make it easier to understand by ensuring the traceability of system development. There are two types of requirements: composites and elementary (atomic) ones, as shown in Figure 2. Composite requirements are composed of other requirements, while elementary (atomic) requirements are not composed of any other requirements.

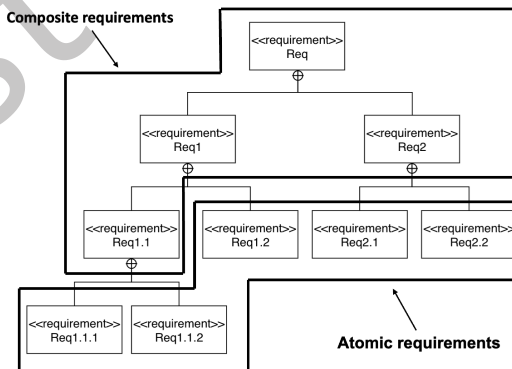

Fig. 2. Composite and Atomic requirements

Just Accepted

2.2 OCaml

In this section, we present the OCaml constructs used in our formalization of SysML diagrams. We focus on defining types, structures, and functions, with particular attention to recursion in functions.

Type in OCaml

In OCaml, we define custom types using the keyword **type**. For example, to define a type for a simple arithmetic expression. We use the following syntax:

```
type expr = | Int of int | Add of expr + expr | Mul of expr * expr
```

In this example, the expr type can represent integer values (Int of int), addition expressions (Add of expr + expr), and multiplication expressions (Mul of expr * expr).

Structure in OCaml

Structures in OCaml are defined using records. We illustrate this with a structure representing a person:

```
type person = {  name : string; age : int; }
```

This `person` structure includes fields for the person's name and age.

Function in OCaml

In OCaml, functions play a crucial role in structuring code and performing computations. To define a function in OCaml, you use the keyword **fun** followed by the parameter and the expression to be computed. For instance, let's define a simple function that adds two numbers:

```
let add_numbers x y = x + y
```

In this example, `add_numbers` is a function that takes two parameters (`x` and `y`) and returns their sum. When a function calls itself during its execution, it is termed a recursive function. Recursion is a powerful concept that simplifies the code for operations that involve repeated or nested computations. Let's consider a classic example of a recursive function: calculating the factorial of a number.

```
let rec factorial n =  if n <= 1 then 1  else n * factorial (n - 1)
```

In this example, `factorial` is a recursive function that calculates the factorial of a given number `n`. It continues to call itself until it reaches the base case (`n <= 1`), at which point it returns 1. The final result is the product of all the numbers from 1 to `n`.

3 PROPOSED APPROACH

To provide clarity on our approach, we classify systems based on the complexity of their decomposition and assembly processes:

Single-Step Assembly (Simple System): In scenarios where the system to be built consists of straightforward and well-defined components, the assembly can be accomplished in a single step. These systems, referred to as "simple systems," involve directly assembling elementary components to meet the abstract specification (see Figure 3). This approach is typically applicable when the system's functionality and interactions are relatively uncomplicated, allowing immediate composition without intermediate refinement stages.

Multi-Step Assembly (Complex System): Conversely, more intricate systems, referred to as "complex systems" require a multi-step assembly process (see Figure 4). These systems involve the definition of abstract sub-components, which are subsequently refined by assembling other components. This hierarchical and iterative refinement process ensures that each sub-component meets its specified requirements before being integrated into the final system.

Just Accepted

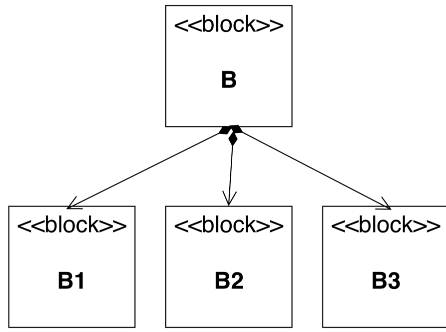

Fig. 3. Simple system

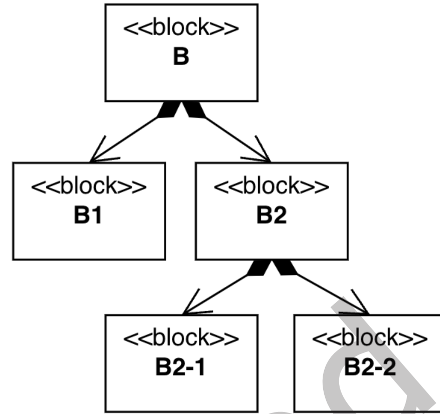

Fig. 4. Complex system

Definition 3.1 (Refinement of a composite SysML block with a composition of blocks). Let B be an abstract block described with the block definition diagrams BDD_B and the internal block diagram IBD_B. Let $B_1, ..., B_n$ be the sub-blocks of B, described with their SysML models (see Figure 5). The composition of $B_1, ..., B_n$ refines the abstract block A if, there is consistency at the structural level between the composition of the sub-blocks B_i and the composite block A.

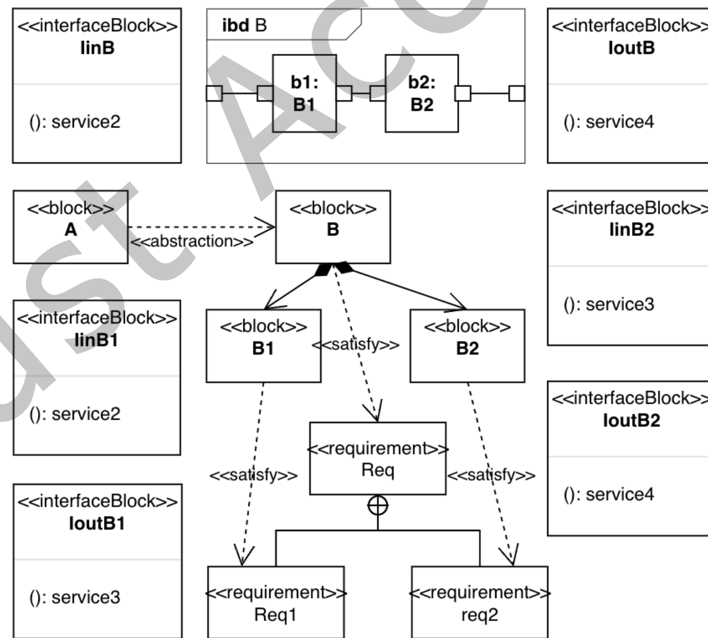

Fig. 5. The proposed structural architecture to model the abstract component

Just Accepted

To verify the consistency, we consider the three following conditions:

- **Condition 1 :**
  For all block B in Set_Blocks and B is composed of a set of blocks {B1,...,Bn} and B provides a Set_of services {S1,...,Sm} then {B1,...,Bn} provides at least Set_of services {S1,...,Sm}
- **Condition 2 :**
  For all block B in Set_Blocks and B is composed of a set of blocks {B1,...,Bn} and B requires a Set_of services {S1,...,Sm} then {B1,...,Bn} requires at most Set_of services {S1,...,Sm}
- **Condition 3 :**
  For each block $B \in$ Set_Blocks, let $\{B1, ..., Bn\}$ be the sub-blocks composing B, and let R be a requirement satisfied by B, where R is composed of atomic requirements $\{R1, ..., Rm\}$. The composition $\{B1, ..., Bn\}$ must collectively satisfy the set of atomic requirements $\{R1, ..., Rm\}$ such that: $\forall Ri \in \{R1, ..., Rm\}, \exists Bj \in \{B1, ..., Bn\}$ satisfying Ri

In order to confirm the first and second conditions, we will make use of the internal block diagrams and the connections between different parts via their respective ports. To verify the first condition, as shown in Figure 6, we need to ensure that the services provided by the interface block that types port PInA are included in the services provided by the interface block that types port PInB.

$$PInA \subseteq PInB$$

Similarly, for condition 2, we need to verify that the services offered in the interface block that types port PoutC are included in the service offered by the interface block that types port PoutA.

$$POutC \subseteq POutA$$

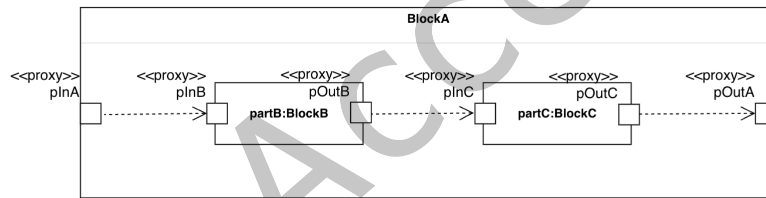

Fig. 6. Simple connection

Just Accepted

However, in cases where we have multiple input delegation ports, as shown in Figure 7, we need to ensure that the services offered in the interface block that types port PInA are included in the services offered by the interface block that types port PInB union PInC.

$$PInA \subseteq PInB \cup PInC$$

Another possible scenario is where we have several output delegation ports, as shown in Figure 8. In such a situation, we need to verify that the services offered by the interface blocks that type the POutB union POutC port are included in the service offered by the interface block that types the POutA port.

$$POutB \cup POutC \subseteq POutA$$

If there is an internal part offering a service required by another part linked by output delegation with the composite block, we remove the required services before verifying the inclusion, as shown in Figure 9. In this case, we have to verify that the services offered in the interface block, which types port POutC minus the services offered in the interface block, which types port POutB are included in the service offered by the interface block, which types port POutA.

$$POutC - POutB \subseteq POutA$$

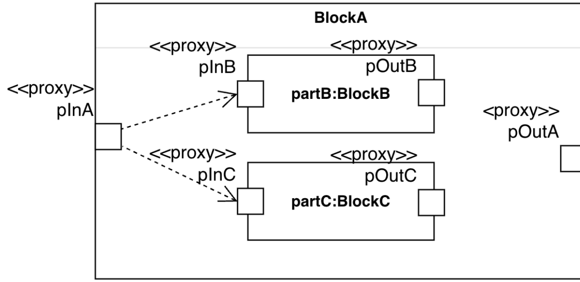

Fig. 7. Multiple In delegation connections

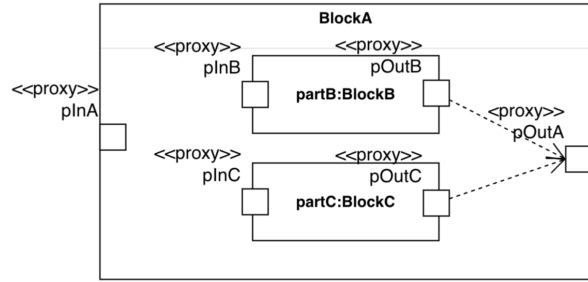

Fig. 8. Multiple Out delegation connections

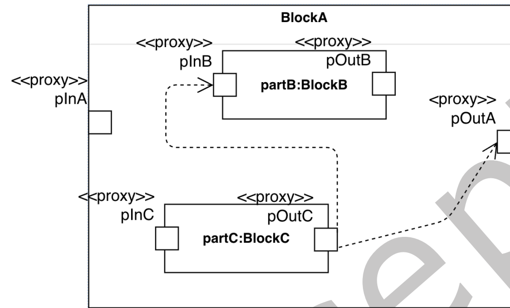

Fig. 9. Delegation connection with composition connection

Based on these conditions, we can deduce that checking the consistency of delegationINs is different from checking delegationOUTs.

As SysML lacks formalism and to automate verification, we need to formally define the SysML models specifying the architecture of a CPS. The proposed formal specification is defined in section 4.

4 FORMAL SPECIFICATION OF SYSML ARCHITECTURAL DIAGRAMS

4.1 Syntax of SysML structural diagrams

We propose a formal model to specify all SysML diagrams to analyze the architecture of a CPS specified with SysML and verify its structural coherence. We propose formalizing the block, interface block, port, requirement, block definition diagram, and requirement diagram.

Definition 4.1 (Formal Requirement). A formal requirement is defined by the following type:

type requirement = { id : string ; name : string ; text : string; }

- *id* is the identifier of the requirement,
- *name* is the name of the requirement,
- *text* is the textual description of the requirement.

To model the requirement diagram, we first need to define a structure we will refer to as coupleReq, which is defined by a pair that associates a requirement with a list of requirements.

Just Accepted

type coupleReq = requirement * requirement list

Definition 4.2 (Formal Requirement diagram). A formal requirement diagram is defined by the following type:

type reqDiagram = { iR : requirement ; sR : requirement list ; relC : coupleReq list }

- iR is the name of the initial requirement of the diagram,
- sR is the finite set of requirements in the diagram,
- $relC \subseteq SR * P(SR)$ is the composition relation, where $P(SR)$ is the set of subsets of SR,

Definition 4.3 (Formal Interface block). A formal model for a SysML interface block IB is defined by the following type:

type interfaceBlock = { nameIB : string ; opIB : string list}

- $namexIB$ is the name of the interface block,
- $opIB$ is the set of functions that the block requires or offers.

Definition 4.4 (Formal Port SysML). The formal SysML port is defined by the following type:

type port = {nameP : string ; typeP : blocInterface ; directionP : string; connectorInterneSBC : string list }

- $nameP$ is the name of the port,
- $typeP$ is the interface block that types the port.
- $directionP$ is the direction of the port (in or out).
- $connectorInterneSBC$ is the set of functions required by the block which contains this port and which are satisfied internally by composition.

To model the Block we first need to define couplePorts, which is defined by a pair that associates two ports. In order to represent all port connections.

type couplePorts = port * port

Definition 4.5 (Formal SysML Block). The formal SysML block B is defined by the following type:

type block = { name : string ; pinB : port ; poutB : port ; connectorsInB : port list; connectorsOutB : couplePorts list; sRB : requirement list}

- $name$ is the name of the block B,
- $pinB$ is the In port of the block B,
- $poutB$ is the Out port of the block B,
- $delegationInB = \{(p_i, p_j)\ such\ as\ (p_i, p_j) \in Ports \times Ports \wedge p_i.direction = p_j.direction = IN\}$, the set of connectors that link the ports of the sub-blocks (parts) with the ports of the composite block. They are called delegation IN connectors.
- $delegationOutB = \{(p_i, p_j)\ such\ as\ (p_i, p_j) \in Ports \times Ports \wedge p_i.direction = p_j.direction = OUT\}$, the set of connectors that link the ports of the sub-blocks (parts) with the ports of the composite block. They are called delegation OUT connectors.
- sRB is the set of requirements that the block satisfies.

Just Accepted

To model the Block Definition Diagram we first need to define coupleBloc, which is defined by a pair that associates a block to a set of blocks. In order to associate each block with its sub-blocks.

type coupleBlock = block * block list

Definition 4.6 (Formal Block Definition Diagram). The formal BDD of a system S is defined by the following type:

type fBDD = {iBs : block ; sBs : block list ; suBs : coupleBlock list }

- *iBs* is the main block,
- *sBs* is the set of all blocks in the system,
- *suBs* the set of couples that connect each block to its sub-blocks,

4.2 Static semantics

Static semantics define the meaning of the structures permitted by the syntax. It encompasses rules that ensure a syntactically correct sequence of symbols is meaningful within the model or program's context. In essence, static semantics verify properties that must hold before execution, such as type correctness, proper initialization, and adherence to logical rules that maintain the integrity of the model.

Before giving the static semantic, we introduce some useful auxiliary functions:

- $subB\,(b)$: the set of sub-blocks of the block b.
- $incluList\,(L1, L2)$: a boolean value. It equals true if all elements of L1 are included in L2.
- $soustractionElement\,(L1, L2)$: a set of elements that exists in L1 and does not exist in L2.
- $union\,(b.connectorsInB)$: all the services offered by all the blocks connected to the block b with a delegation input connection
- $AtomicReq\,(b)$: the set of atomic requirements verified by the block b.

Given an $fBDD\ \triangle$, $\triangle = \{ibs; sbs; subs\}$ and a $reqDiagram\ reqD$, $reqD = \{iR; sR; relC; relD\}$, we can define the static semantics as follows:

Ensure that the services provided by the abstract block are also provided by its sub-blocks. We must first combine the services offered by all the sub-blocks connected to the abstract block's input port. After that, we can check if the abstract block's services are included in the combined services of the sub-blocks.

$$S_1\,[\triangle] \overset{\text{def}}{=} \forall b \in sBs \wedge subB\,(b) \neq \{\} \wedge \exists\,(b.pinB, p1) \in b.delegationInB \wedge subB\,(b) \neq \{\} \wedge incluList\,(b.pinB.typeP.opB, (union\,(b.connectorsInB)))$$

Ensure that the sub-blocks do not require more services than the abstract block. To this matter, we add up all the services required by the sub-blocks linked to the abstract block with a DelegationOut relationship. Then, we subtract the services offered by internal components. At the end we check whether the remaining services belong to the list of services required by the abstract block or not.

$$S_2\,[\triangle] \overset{\text{def}}{=} \forall b \in sBs \wedge subB\,(b) \neq \{\} \wedge \exists\,(p1, b.poutB) \in b.delegationOutB \wedge incluList\,(soustractionElement\,(p1.typeP.opB, p1.connectorInterneSBC)\,, p2.typeP.opB)$$

For each block B, we collect all these atomic requirements in a list L1. then, we collect all the atomic requirements of its sub-blocks in a list L2. Finally, we check if all the elements of L1 are included in L2.

$$S_3\,[\triangle, reqD] \overset{\text{def}}{=} \forall b \in sBs \wedge subB\,(b) \neq \{\} \wedge AtomicReq\,(b, reqD) \in AtomicReq\,(subB\,(b, reqD))$$

The algorithm 1 receives a formal block definition diagram. Then, it goes through these blocks individually and checks that each of them verifies S1 and S2. Algorithm 2 receives a formal block definition diagram and a formal

Just Accepted

requirement diagram. Then, it goes through these blocks individually and checks that each one verifies S3. The implementation of these two algorithms in the OCaml language can be found in the listings 11 and 12 of the Appendix A. Therefore, the structural consistency of our model is captured by this static semantics, our model is consistent if the static semantics is true.

Algorithm 1 Service consistency

INPUTS :

```
 The initial block ``bc" and the system BDD ``bdd"
```

OUTPUTS :

```
 List of blocks annotated with true if the block verifies the first and the second conditions,
otherwise the block will be annotated with false

BEGIN
LET SubB = Sub (bc, bdd) and result = true;
IF (S1 (bc) AND S2 (bc)) Then result = true;
ELSE result = false;
WHILE SuBB! = ∅ Do
  LET bi ∈ SuBB ,
  result = result AND (S1 (bi) AND S2 (bi))
  IF Sub (bi, ibd)  != ∅ THEN
   result = result AND serviceConsistency (bi, bdd) ;
  END IF
   SuBB = SuBB − bi ;
 END WHILE;
 Return result
END
```

5 TRANSFORMATION TOOL

Eclipse Papyrus is an open-source modelling tool in the larger Eclipse Modeling Project. It provides a platform for creating various models, particularly those based on standard modelling languages such as the Unified Modeling Language (UML) and its profiles, SysML (Systems Modeling Language). Eclipse Papyrus typically saves SysML diagrams and models in the XMI (XML Metadata Interchange) format produced by the Object Management Group (OMG). The XMI format allows the interchange of objects and models through an XMI-formatted file.

Figures 10, 11, and 12 show an example of how SysML element data is represented in the XMI file. Figure 10 illustrates how requirements are represented. Figure 11 shows how the exigence hierarchy is represented. Figure 12 shows how the mapping between blocks and requirements is represented.

We developed a tool to automate the transformation of the SysML model into the proposed formal specification. However, some naming conventions must be respected during the modelling process with papyrus.

- **Convention 1:** Block names must end with the word “block”.
- **Convention 2:** Interface block names must begin with the name of the associated block and end with either “InterfaceIn” or “InterfaceOut”.
- **Convention 3:** Port names must end with “PortIn” or “PortOut”.
- **Convention 4:** The name of the delegation connectors that start from the block to parts must end with “Delegation-IN”.
- **Convention 5:** The name of the delegation connectors that start from the parts to block must end with “Delegation-OUT”.

Just Accepted

Algorithm 2 Requirement consistency

INPUTS :
The initial system block ``bc", system BDD ``bdd" and requirement diagram ``reqD"
OUTPUTS :
true if the block verifies the third condition otherwise false

BEGIN
Let SubB = $Sub(bc, bdd)$; and finalListBc = $AtomicReq(bc)$;
and resultat = $true$;
IF $finalListBc = \emptyset$ **Do result** = $false$;
WHILE SuB_B != $\emptyset$ **Do**
Let $b_i \in SuB_B$,
finalListBi = $AtomicReq(bi)$;
finalListBc = finalListBc - finalListBi;
IF $Sub(bi, bdd)$!= $\emptyset$ **Do**
resultat = $requirementConsistency(bi, bdd, reqD)$;
END IF
$SuB_B = SuB_B - b_i$;
END WHILE;
IF $result == true$ and $finalList = \emptyset$ **DO**
Return $true$
Else
Return $false$
END IF
END


```
<Requirements:Requirement xmi:id="_cS0RsN-LEeywT9d24_j_DQ" base_NamedElement="_cSorgN-LEeywT9d24_j_DQ" id="
00" text="Global energy consuming of the Cycab: the CyCab must not exceed an appropriate maximum energy
consumption" base_Class="_cSorgN-LEeywT9d24_j_DQ"/>
<Requirements:Requirement xmi:id="_f_D58N-LEeywT9d24_j_DQ" base_NamedElement="_f-5h4N-LEeywT9d24_j_DQ" id="
01" text="Global energy consuming of the Station: the Station must not exceed an appropriate maximum
energy consumption" base_Class="_f-5h4N-LEeywT9d24_j_DQ"/>
<Requirements:Requirement xmi:id="_kEsSkN-LEeywT9d24_j_DQ" base_NamedElement="_kEca8N-LEeywT9d24_j_DQ" id="
02" text="Global energy consuming of the Vehicle: the Vehicle must not exceed an appropriate maximum
energy consumption" base_Class="_kEca8N-LEeywT9d24_j_DQ"/>
<Requirements:Requirement xmi:id="_mp2-AN-LEeywT9d24_j_DQ" base_NamedElement="_mpsl8N-LEeywT9d24_j_DQ" id="
011" text="Global energy consuming of the Sensor&#xA;-the maximum energy consuming of the offered service
pos is limited to Opos&#xA;-the maximum energy consuming of the required service spos is limited to Ospos
&#xA;" base_Class="_mpsl8N-LEeywT9d24_j_DQ"/>
<Requirements:Requirement xmi:id="_nBHAYN-LEeywT9d24_j_DQ" base_NamedElement="_nBCu8N-LEeywT9d24_j_DQ" id="
012" text="Global energy consuming of the Computing Unit&#xA;-the maximum energy consuming of the required
service halt is limited to chall&#xA;-the maximum energy consuming of the offered service pos is limited
to Opos&#xA;-the maximum energy consuming of the required service far is limited to Ofar " base_Class="
_nBCu8N-LEeywT9d24_j_DQ"/>
```


Fig. 10. Requirement information representation in XMI file

Just Accepted

The first and second conventions are necessary to differentiate between blocks and interface blocks since the ".Uml" file generated by Papyrus uses the same tags for both. The third convention allows us to differentiate between port In and port Out since the ".Uml" file generated by Papyrus does not allow us to differentiate between them. The fourth and fifth conventions allow us to differentiate between the Assembly and delegation connectors.

```xml
<packagedElement xmi:type="uml:Class" xmi:id="_cSorgN-LEeywT9d24_j_DQ" name="GECC" isLeaf="true">
  <nestedClassifier xmi:type="uml:Class" xmi:id="_f-5h4N-LEeywT9d24_j_DQ" name="ECS">
    <nestedClassifier xmi:type="uml:Class" xmi:id="_nBCu8N-LEeywT9d24_j_DQ" name="ECCU"/>
    <nestedClassifier xmi:type="uml:Class" xmi:id="_mpsl8N-LEeywT9d24_j_DQ" name="ECSS"/>
  </nestedClassifier>
  <nestedClassifier xmi:type="uml:Class" xmi:id="_kEca8N-LEeywT9d24_j_DQ" name="ECV">
    <nestedClassifier xmi:type="uml:Class" xmi:id="_-YISQPiGEeyplPXJEsGKTg" name="ECSR"/>
    <nestedClassifier xmi:type="uml:Class" xmi:id="_Apyy4PiHEeyplPXJEsGKTg" name="ECEH"/>
    <nestedClassifier xmi:type="uml:Class" xmi:id="_18dtgPiGEeyplPXJEsGKTg" name="ECVC"/>
  </nestedClassifier>
</packagedElement>
```

Fig. 11. Relationship between requirements representation in XMI file

```xml
<Requirements:Satisfy xmi:id="_pwJA8PiOEeylPq067yBXHA" base_DirectedRelationship="_pv2tEPiOEeylPq067yBXHA"
base_Abstraction="_pv2tEPiOEeylPq067yBXHA"/>
<Requirements:Satisfy xmi:id="_3WilMPiOEeylPq067yBXHA" base_DirectedRelationship="_3WapYPiOEeylPq067yBXHA"
base_Abstraction="_3WapYPiOEeylPq067yBXHA"/>
<Requirements:Satisfy xmi:id="_38wQoPiOEeylPq067yBXHA" base_DirectedRelationship="_38t0YPiOEeylPq067yBXHA"
base_Abstraction="_38t0YPiOEeylPq067yBXHA"/>
<Requirements:Satisfy xmi:id="_4cs8APiOEeylPq067yBXHA" base_DirectedRelationship="_4cqfwPiOEeylPq067yBXHA"
base_Abstraction="_4cqfwPiOEeylPq067yBXHA"/>
<Requirements:Satisfy xmi:id="_5jaFYPiOEeylPq067yBXHA" base_DirectedRelationship="_5jYQMPiOEeylPq067yBXHA"
base_Abstraction="_5jYQMPiOEeylPq067yBXHA"/>
<Requirements:Satisfy xmi:id="_6IV-cPiOEeylPq067yBXHA" base_DirectedRelationship="_6ITiMPiOEeylPq067yBXHA"
base_Abstraction="_6ITiMPiOEeylPq067yBXHA"/>
<Requirements:Satisfy xmi:id="_65ThUPiOEeylPq067yBXHA" base_DirectedRelationship="_65RsIPiOEeylPq067yBXHA"
base_Abstraction="_65RsIPiOEeylPq067yBXHA"/>
```

Fig. 12. Mapping between blocks and requirements representation

6 RUNNING EXAMPLE

To illustrate our proposal, we present the case study of a CyCab car, which is a component-based system [2], developed by INRIA [1], and considered as a case study in the ANR TACOS[2] project, which is a French research initiative. The ANR TACOS project aims to develop a component-based methodology for specifying high-safety systems, particularly in land transportation, from the initial requirement stages to the development of formal specifications. The CyCab is a small, electric, and automatic vehicle used as a means of transportation designed primarily for autonomous transport. It allows users to move around via a set of pre-installed stations. It is controlled by a computer system and can be automatically piloted in many modes.

One of the critical aspects of the CyCab system is its modular design, which enhances both maintenance and fault tolerance. By decomposing the CyCab system into distinct sub-systems, such as the station components, sensors, and computing units, we improve the ease of identifying and isolating faults. This decomposition not only supports more efficient troubleshooting and repairs but also allows for the seamless integration of new features without disrupting the existing system. This modular approach is vital for maintaining the reliability and safety of the CyCab, particularly given its role in autonomous transport.

To illustrate our approach, we consider the following constraints: A CyCab has a dedicated road where the stations are equipped with sensors and computing units. There are no obstacles on the road. The driving of the CyCab is guided by the information received from the stations, which allows the CyCab to be located relative to the stations.The (*Starter*) receives a signal from the station to activate the vehicle. The vehicle is also equipped

[1]Institut National de Recherche en Informatique et Automatique
[2]Trustworthy Assembling of Components: from requirements to Specification

Just Accepted

with an emergency halt button associated with the Emergency Halt (EH) component. The emergency halt button can be activated anytime during the movement of the CyCab.

6.1 Modeling

To specify the architecture of a CPS, in this case, the CyCab, we propose to use the following SysML diagrams: the Block Definition Diagram (BDD), the Internal Block Diagram (IBD), and the Requirement Diagram (REQD). Figure 13 represents the BDD of the composite block *CyCab*, which is connected by compositional relations with its sub-blocks. And the IBD of CyCab, Station and vehicle blocks are illusstred in Figures 14, 15 and 16.

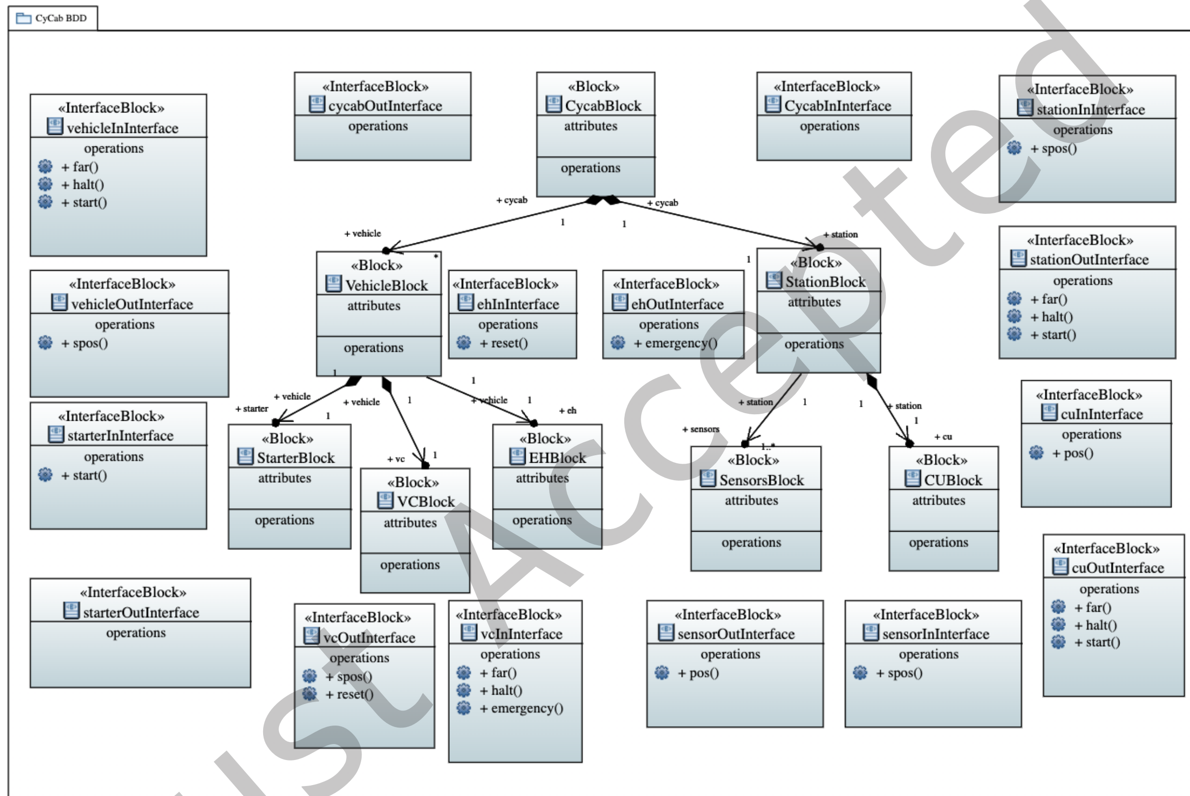

Fig. 13. Block Definition Diagram of the *CyCab* block

Just Accepted

Figure 17 represents the requirement diagram of the system. the requirement GECC, Global Maximal Energy Consuming of CyCab, indicates that the CyCab must not exceed the energy consumption limit. This requirement contains the requirements for the ECS (Maximum Energy Consumption of Station Component) and ECV (Maximum Energy Consumption of Vehicle Component). The ECS requirement contains the requirements ECSS, the maximum energy consumption of the sensor component, and ECCU, the maximum energy consumption of the computing unit component. The ECV requirement contains the requirements ECVC, the maximum energy consumption of the vehicle core component; ECSR, the maximum energy consumption of the starter component; and ECEH, the maximum energy consumption of the emergency stop component.

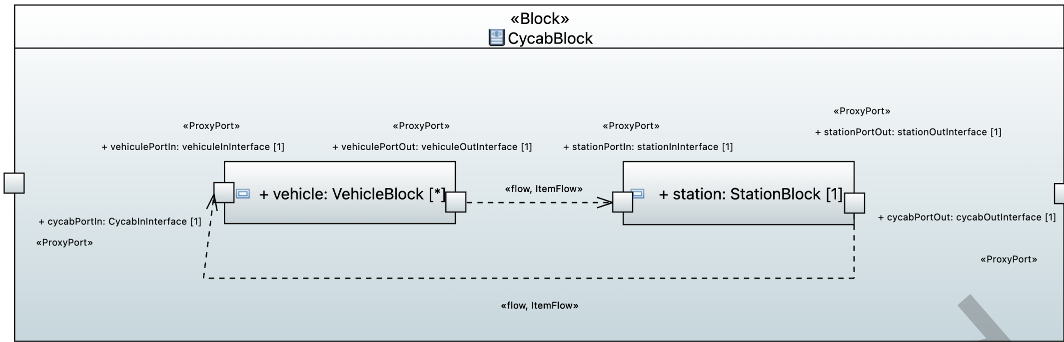

Fig. 14. Internal Block Diagram of the *CyCab* block

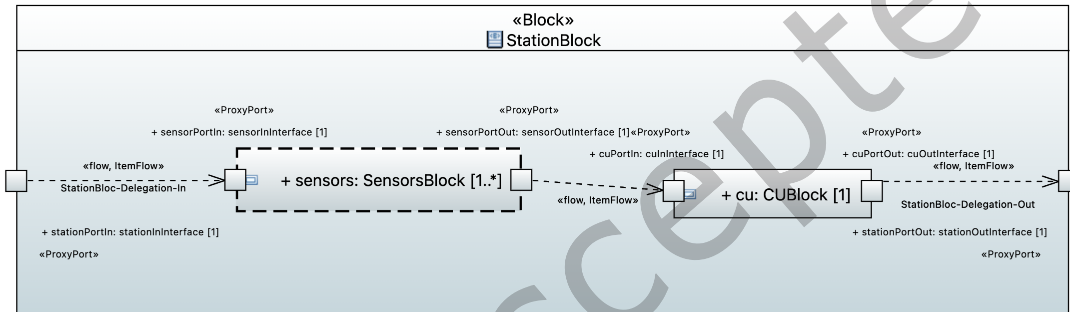

Fig. 15. Internal Block Diagram of the *Station* block

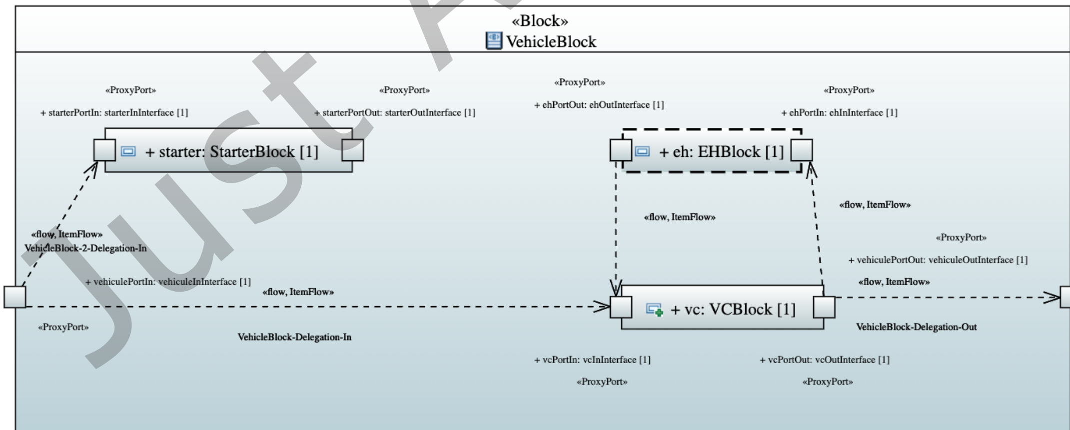

Fig. 16. Internal Block Diagram of the *Vehicle* block

Just Accepted

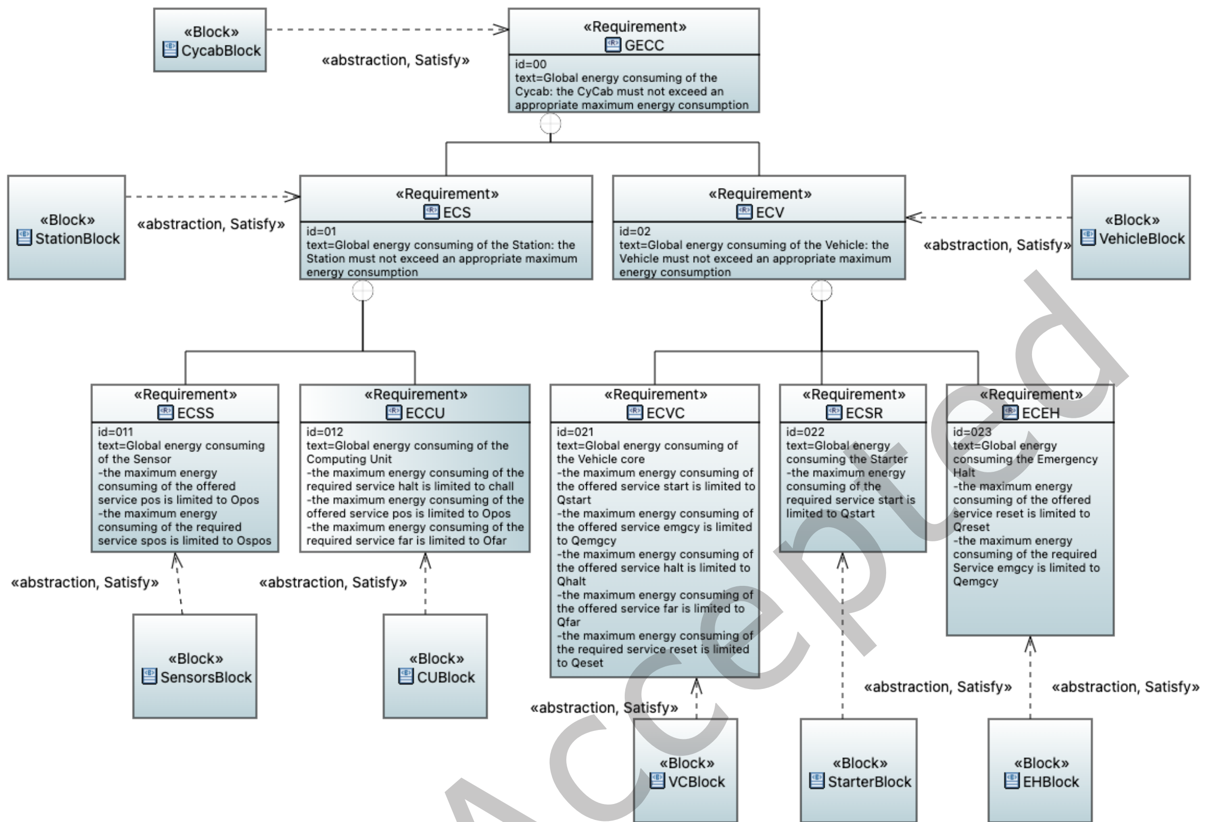

Fig. 17. The *CyCab* requirement diagram

6.2 Transformation

After modeling the Sysml diagrams, we proceed to the transformation step, We introduce the file generated by Papyrus into the tool presented in section 5, which will generate the formal specification. However, given the size of the case study, we will not be able to include the specification of the entire case study in this paper.

In the following we will present some specifications: In listing 1, we can find the GECC, ECS, and ECV requirements specifications. In listing 2, we find the formal specification of the requirement diagram. In listing 3 shows the formal specifications for the vehicle input and output interface blocks. In listing 4, we find the formal specification of the vehicle's input and output ports and that of the VC, noting that it contains the additional information that it is internally connected to a block that provides it with the reset service. This information will be used when checking structural consistency. Listing 5 shows the formal specifications for the station and vehicle block. Finally, in listing 6, we find the formal specifications for the IBD.

Listing 1. Requirements

Just Accepted

```
let gecc = { id = ``00"; name = ``GECC"; texte = ``Global energy consuming
    of the Cycab: the CyCab must not exceed an appropriate maximum energy
    consumption"}
let ecs = { id = ``01"; name = ``ECS"; texte = ``Global energy consuming of
     the Station: the Station must not exceed an appropriate maximum energy
    consumption"}
let ecv = { id = ``02"; name = ``ECV"; texte = ``Global energy consuming of
     the Vehicle: the Vehicle must not exceed an appropriate maximum energy
    consumption"}
```

Listing 2. Requirement diagram

```
let reqD = { iR = gecc ;
sR = [ gecc ; ecs ; ecv ; ecss ; eccu ; ecvc ; ecsr ; eceh ] ;
relC = [ (gecc, [ecs ; ecv]) ;(ecs, [eccu ; ecss]) ;(ecv, [ecsr ; eceh ;
    ecvc])] ; }
```

Listing 3. Interface block

```
let vehicleininterface = { nameIB = ``vehicleInInterface"; opB = [ ``far" ;
    "halt" ;"start"  ]  }
let vehicleoutinterface = { nameIB = ``vehicleOutInterface"; opB = [ ``spos
    "  ]  }
```

Listing 4. Ports

```
let vehicleportin = {nameP = ``vehicleportin" ;typeP = vehicleininterface ;
    directionP = ``in" ;connectorInterneSBC = []}
let vehicleportout = {nameP = ``vehicleportout" ;typeP =
    vehicleoutinterface ;directionP = ``out" ;connectorInterneSBC = []}
let vcportout = {nameP = ``vcportout" ;typeP = vcoutinterface ;directionP =
     ``out" ;connectorInterneSBC = ["reset" ]}
```

Listing 5. Blocks

```
let stationblock = {name = ``StationBlock" ;pinB = stationportin ;poutB =
    stationportout;connectorsOutB = [(cuportout,stationportout ) ] ;
    connectorsOutBIn = [(stationportin,sensorportin ) ] ; sRB = [ecs] ; }
let vehicleblock = {name = ``VehicleBlock" ;pinB = vehicleportin ;poutB =
    vehicleportout;connectorsOutB = [(vcportout,vehicleportout ) ] ;
    connectorsOutBIn = [(vehicleportin,vcportin ) ;(vehicleporti
```

Listing 6. Block definition diagram

```
let bddD = {
iBs =cycabblock;
sBs = [ cycabblock;stationblock;vehicleblock;cublock;sensorsblock;
    starterblock;vcblock;ehblock ] ;
```

Just Accepted

```
suBs = [ (cycabblock,[stationblock;vehicleblock]) ; (stationblock,[cublock
    ;sensorsblock]) ; (vehicleblock,[starterblock;vcblock;ehblock])   ] ;
}
```

6.3 Structural consistency checking

Finally, after transforming the SysML models into the proposed formal specification, we verify the structural consistency using the implementation of algorithms 1 and 2. We specify that our case study contains all the special cases considered in our approach. The IBD of the station contains the simple cases from which the services offered by the station must be included in the service offered by the sensors and the services required by the CU must be required by the station. The IBD of the vehicle contains two special cases, where the services offered by the vehicle must be included in the services offered by the VC union starter, and the services required by the VC must be included in the services required by the vehicle, but given that it is connected to the EH input port, we must first subtract the services offered by the EH from the list of services required by the VC. Also for the consistency of the mapping between blocks and requirements, we must verify that the atomic requirements that the GECC (ECSS, ECCU, ECVC, ECSR, ECEH) block verifies are included in the atomic requirements that the ECS(ECSS, ECCU) union ECV (ECVC, ECSR, ECEH) block verifies.

Figure 18 shows the result of verifying our model. We can see that all the blocks are annotated with "consistency is verified". Therefore, we can say that our model is structurally consistent.

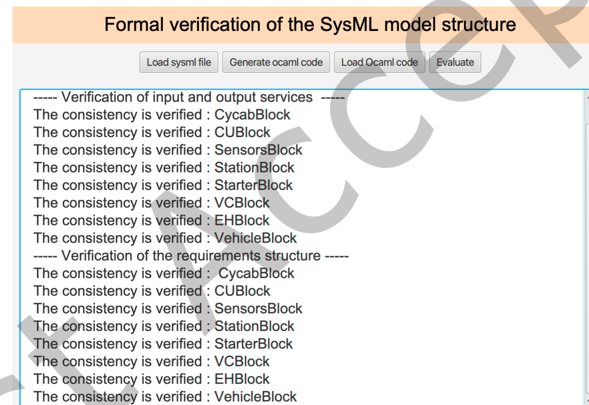

Fig. 18. Result of the consistency verification

Now, if the Station block offers an extra service of switching "off" the station (see Figure 19), The formal specification of the input block interface will be represented as follows:

```
let stationininterface = { namexB = ``stationInInterface"; opB = [ ``spos"
   ; ``off" ]  }
```

Figure 20 shows the result of verifying the model after the modification. We can see that the block Station is inconsistent because the CU and Sensor sub-components cannot provide the service of turning off the station. Specifically, the inconsistency arises because neither of these sub-components offers the "off" service that is required by the Station block. For our system to be consistent, one of the sub-components of the Station that is connected with an IN delegation connection must offer the "off" service. This inconsistency was avoided in our approach by ensuring that all required services for a composite block are provided by at least one of its

Just Accepted

sub-components. Without this verification step, the system might behave unexpectedly or fail to perform critical operations, such as turning off the station. Our approach helps identify such inconsistencies early in the design process, thereby ensuring the robustness and reliability of the CyCab system.

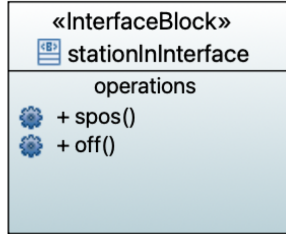

Fig. 19. Station block with OFF service

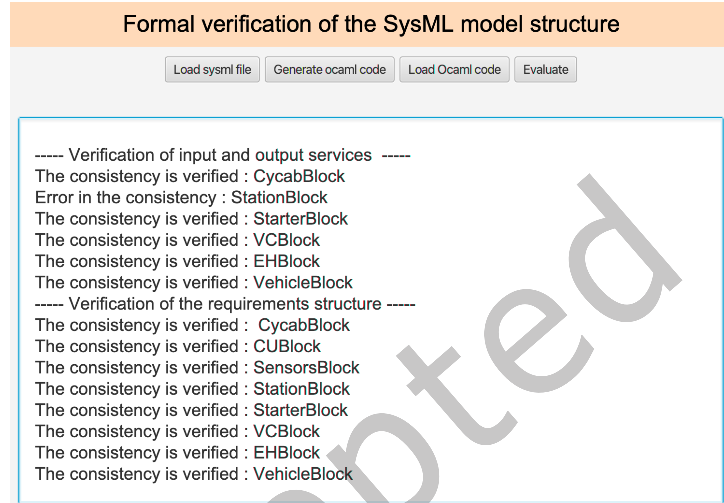

Fig. 20. Verification result after editing the station interface

Just Accepted

7 DISCUSSION ABOUT THE APPLICABILITY OF OUR APPROACH ON SYSML V2

In SysML v1, associations are commonly used to model port connections. However, these associations are abstract and do not provide details about the connection between ports. This abstraction can lead to potential inconsistencies, as there is no explicit mechanism to ensure that the services required by one port are provided by the connected port. SysML v2 [1] addresses these limitations by introducing interfaces that rigorously define port connections. Defined by interface definitions (interface def), specify the connection protocols and the services that must be provided or consumed by the connected ports. Table 1 gives some correspondence from SysML v1 to SysML v2.

Table 1. Mapping between SysML v1 and SysML v2

SysML v1	**SysML v2**
Part property / Block	Part / Part def
Proxy port / Interface block	Port / Port def
	Interface / Interface def

Even if SysML v2 significantly improves modeling by introducing interface definitions that define connection and binding port services, it still needs to detect the structural inconsistency we propose in our approach. To illustrate this, we modeled the CyCab's Station structure with SysML v2 (see Listings 7, 8, 9, 10) and included a service that the station offers **"off"**. Figure 21 illustrates the IBD of the station generated from the textual notation. After building the project, no errors were detected, although a consistency condition was broken when an abstract block "Station" offers more services than the composition of its sub-blocks (Sensor,Cu). This underlines

the need for our proposed structural consistency check to guarantee the integrity of these models. While our approach is applicable to SysML v2, it requires adaptation to accommodate the textual notation specific to SysML v2. The new structure, including interface definitions and connection protocols, necessitates adjustments in the algorithms used for structural consistency checking. These modifications will ensure that the integrity of the models is maintained within the SysML v2 framework, thereby extending the utility of our approach in this updated context.

Listing 7. Port def

```
port def IB_Station_in {
    in ref action pos {}
    out ref action halt {}
    out ref action far {}
    out ref action off {}}
port def IB_Station_out {
    out ref action halt {}
    out ref action far {}}
port def IB_Sensor_in {
    in ref action pos {}
    out ref action spos {}}
port def IB_Sensor_out {
   out ref action spos {}}
port def IB_CU_in {
   in ref action spos {}}
port def IB_CU_out {
   out ref action halt {}
   out ref action far {}}
```

Listing 8. interface def

```
interface def Station_Sensor_Interface{
    end port pp : IB_Station_in;
    end port pp_conj : IB_Sensor_in;
    bind pp_conj.pos = pp.pos;}
interface def Sensor_CU_Interface{
    end port pp : IB_Sensor_out;
    end port pp_conj : IB_CU_in;
    bind pp_conj.spos = pp.spos;}
interface def CU_Station_Interface{
    end port pp : IB_CU_out;
    end port pp_conj : IB_Station_in;
    flow pp_conj.far to pp.far;
    flow pp_conj.halt to pp.halt;}
```

Just Accepted

Listing 9. Part def

```
part def Station {
   port pStation : IB_Station_in;
   port pStation_out : IB_Station_out;
   part def Sensor{
       port pSensor : IB_Sensor_in;
   port pSensor_out : IB_Sensor_out;}
   part def CU{
   port pCU : IB_CU_in;
   port pCU_out : IB_CU_out;}
   connection def Station_Sensor {
       end part block1 : Station;
       end part block2 : Sensor;
       interface :Station_Sensor_Interface connect block1.pStation to block2.pSensor; }
   connection def Sensor_CU {
       end part block1 : Sensor;
   end part block2 : CU;
   interface :Sensor_CU_Interface connect block1.pSensor_out to block2.pCU;}
   connection def Cu_Station {
   end part block1 : CU;
   end part block2 : Station;
   interface :CU_Station_Interface connect block1.pCU_out to block2.pStation;} }
```

Listing 10. Sensor usage

```
part station : Station {
```

```
    part  sensor : Sensor{}
    part cu : CU{}
    interface : Station_Sensor_Interface connect pStation to sensor.pSensor;
    interface : Sensor_CU_Interface connect sensor.pSensor_out to cu.pCU;
    interface : CU_Station_Interface connect cu.pCU_out to pStation_out;}
```

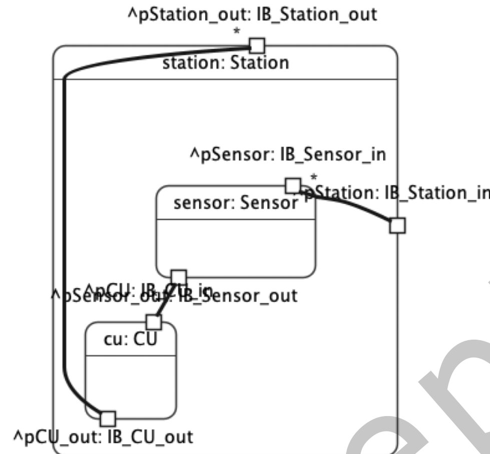

Fig. 21. Station IBD in SysML v2

8 RELATED WORK

This section reviews some related work done by researchers in the field of structural consistency verification and contrasts our approach with contract-based design approaches.

In [3], the authors propose verifying the consistency between the architectural views of various heterogeneous models and the base architecture (BA) of the complete system by using typed graphs to represent architectural views and the base architecture. The consistency between a view and the base architecture can be reduced to the existence of an appropriate graph morphism between the typed graphs of the view and the base architecture. In [13], the authors propose a methodology for validating static properties using Z, a formal specification language, to formally express and verify a CPS modeled using UML deployment diagrams. The authors in these works do not use SysML to model their systems.

Many authors have been working on the verification of the structural consistency of UML diagrams. The authors in [11] define formal semantics for UML sequence diagrams, ensuring consistency between sequence diagrams and other UML models like class diagrams and state diagrams. The formal semantics they present involve checking the consistency of sequence diagrams with class diagram declarations and their hierarchical structure. In [5], the authors provide a formal approach to the traceability of requirements and consistency verification in UML models. The authors propose a framework for verifying UML diagrams, including syntactic correctness, inter-diagram consistency, and requirement traceability. The authors in [18] focus on systematically identifying consistency rules for UML diagrams. Their research aims to provide an updated and consolidated set of UML consistency rules, offering a detailed overview of the current standards in this area. The study emphasizes the interdependence of different UML diagrams and the necessity for them to maintain consistency with each other to ensure the accuracy and integrity of software design models. However, none of these works included the

Just Accepted

internal block diagram or requirement diagrams when checking structural consistency. Other works focused on verifying the consistency between model-based security analysis (MBSA) and model-based systems engineering (MBSE). The authors in [8] focus on ensuring structural consistency between MBSE and MBSA models by using consistency links to build confidence in the security analysis. The authors in [19] propose an approach for harmonizing systems engineering and security assessment models. This is achieved by verifying the structural consistency of these models through three key elements: blocks, ports, and connections and support the languages that integrate them, such as SysML. Unlike our approach, which uses structural consistency to check whether the decomposition of a system is correct, this work verifies the structural consistency between MBSE and MBSA.

In various related works, authors often transform SysML specifications into formal representations like automata or Petri nets to enable verification, which typically requires detailed behavioral specifications. For example, in [4], SysML blocks are first translated into automata using behavioral diagrams to achieve formal verification. While this transformation allows a comprehensive verification, it heavily depends on the availability of behavioral specifications, which may not be feasible in the early stages of system design. Our approach, in contrast, introduces a formal specification framework based on consistency rules that can be derived solely from the structural specification of the system. This enables early verification of structural consistency without requiring behavioral specifications, thus providing a critical advantage in ensuring system integrity at an early design stage.

Contract-based design approaches provide a way to analyze both structural and behavioral properties of diagrams. Formalisms and tools, such as AGREE (Assume Guarantee REasoning Environment) [7] uses compositional reasoning to verify component-level assumptions and guarantees within a system architecture. PACTI (Passivity And Compatibility for Timed Interfaces) [9] focuses on checking the passivity and compatibility of timed interfaces, while OCRA (Operational Contracts for Real-time Applications) [6] leverages contracts to ensure the correct behavior of real-time systems. Our approach, focused on verifying the structural consistency of SysML models, complements these contract-based methods by ensuring the logical soundness and proper interaction of components before detailed behavioral specifications are considered. By integrating structural consistency verification into the early stages of CPS design, we provide a foundation for subsequent behavioral verification using contract-based approaches.

9 CONCLUSION AND PERSPECTIVES

Given the increasing complexity of cyber-physical systems, this paper proposes a new approach for verifying the decomposition of a CPS using the SysML language. By enabling increased modularity, flexibility, and fault tolerance, this decomposition is a crucial step towards ensuring the safety and reliability of these systems. The approach consists of an early verification of the structural consistency of cyber-physical systems modeled with SysML structural diagrams before defining behavioral aspects. To formalize this verification process, we address SysML's limitations for formal verification and introduce syntax and static semantics for structural and requirement diagrams. These enhancements allow us to verify a set of structural consistency rules within a refinement relation, ensuring the formal verification of structural consistency within the model. To demonstrate the effectiveness of our approach, we have applied it to a case study involving the design of a cyber-physical system for a self-driving taxi service known as the CyCab.

While the current approach only addresses the problem of structural consistency of structural and requirements diagrams. We believe that future research could involve extending the verification rules to cover behavioral diagrams and more rules for structural diagrams.

Just Accepted

Author contributions

Adel Khelifati: Conceptualization, Formal analysis, Methodology, Code development and Software, Experimentation, Formal analysis, Validation, Visualization, Writing, and editing.
Malika Boukala-Ioualalen: Methodology, Supervision, reviewing.
Ahmed Hammad: Methodology, Supervision, reviewing.

Conflict of interest

The authors declare no potential conflict of interest.

Declaration of generative AI in scientific writing

During the preparation of this work the authors have not used any generative AI or AI-assisted technologies in the writing process.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Just Accepted

REFERENCES

[1] 2023. SysML v2.0 Overview | Object Management Group. https://www.omg.org/events/2023Q2/special-events/SysML-Session.htm
[2] Gérard Baille, Philippe Garnier, Hervé Mathieu, and Roger Pissard-Gibollet. 1999. *The INRIA Rhône-Alpes CyCab.* Technical Report. INRIA. Describes the package natbib.
[3] Ajinkya Bhave, Bruce H Krogh, David Garlan, and Bradley Schmerl. 2011. View consistency in architectures for cyber-physical systems. In *2011 IEEE/ACM second international conference on cyber-physical systems*. IEEE, 151–160.
[4] Oscar Carrillo, Samir Chouali, and Hassan Mountassir. 2013. Incremental Modeling of System Architecture Satisfying SysML Functional Requirements. In *Formal Aspects of Component Software - 10th International Symposium, FACS 2013, Nanchang, China, October 27-29, 2013, Revised Selected Papers*. 79–99. https://doi.org/10.1007/978-3-319-07602-7_7
[5] Jayeeta Chanda, Ananya Kanjilal, Sabnam Sengupta, and Swapan Bhattacharya. 2009. Traceability of requirements and consistency verification of UML use case, activity and Class diagram: A Formal approach. In *2009 Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS)*. IEEE, 1–4.
[6] Alessandro Cimatti, Michele Dorigatti, and Stefano Tonetta. 2013. OCRA: A tool for checking the refinement of temporal contracts. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 702–705.
[7] Darren Cofer, Andrew Gacek, Steven Miller, Michael W Whalen, Brian LaValley, and Lui Sha. 2012. Compositional verification of architectural models. In *NASA Formal Methods: 4th International Symposium, NFM 2012, Norfolk, VA, USA, April 3-5, 2012. Proceedings 4*. Springer, 126–140.
[8] Romaric Demachy and Sébastien Guilmeau. 2022. Structural consistency of MBSE and MBSA models using Consistency Links. In *11th European Congress Embedded Real Time System (ERTS 2022)*. Toulouse, France. https://hal.science/hal-03697170
[9] Inigo Incer, Apurva Badithela, Josefine Graebener, Piergiuseppe Mallozzi, Ayush Pandey, Sheng-Jung Yu, Albert Benveniste, Benoit Caillaud, Richard M Murray, Alberto Sangiovanni-Vincentelli, et al. 2023. Pacti: Scaling assume-guarantee reasoning for system analysis and design. *arXiv preprint arXiv:2303.17751* (2023).
[10] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, KC Sivaramakrishnan, and Jérôme Vouillon. 2023. *The OCaml system release 5.1: Documentation and user's manual.* Ph. D. Dissertation. Inria.
[11] Xiaoshan Li, Zhiming Liu, and He Jifeng. 2004. A formal semantics of UML sequence diagram. In *2004 Australian Software Engineering Conference. Proceedings*. IEEE, 168–177.
[12] S Lu, A Tazin, Y Chen, et al. [n. d.]. Detection of Inconsistencies in SysML/OCL models using owl reasoning. SN Comput. Sci. 4, 175 (2023).
[13] Gabriela Magureanu, Madalin Gavrilescu, and Dan Pescaru. 2013. Validation of static properties in unified modeling language models for cyber physical systems. *Journal of Zhejiang University SCIENCE C* 14, 5 (2013), 332–346.
[14] Yaron Minsky. 2011. OCaml for the masses. *Commun. ACM* 54, 11 (2011), 53–58.
[15] Object Management Group (OMG). 2012. *OMG Systems Modeling Language SysML*. Technical Report.
[16] Object Management Group OMG. 1999. *Unified Modeling Language Specification UML, Version 1.3 ©*. Technical Report.

[17] Paul Pearce and Sanford Friedenthal. 2013. A Practical Approach for Modelling Submarine Subsystem Architecture in SysML. *Engineering Conference* (2013), 14.
[18] Damiano Torre, Yvan Labiche, Marcela Genero, and Maged Elaasar. 2018. A systematic identification of consistency rules for UML diagrams. *Journal of Systems and Software* 144 (2018), 121–142.
[19] Julien Vidalie, Michel Batteux, Faïda Mhenni, and Jean-Yves Choley. 2022. Category Theory Framework for System Engineering and Safety Assessment Model Synchronization Methodologies. *Applied Sciences* 12, 12 (June 2022), 5880. https://doi.org/10.3390/app12125880

A PROGRAMS

In 11 and 12, we can find a proposal implementation of the algorithms 1 and 2 in the OCaml language using the OCaml SysML specification proposed in section 4. The first program receives a block definition diagram (bddD) and the initial block (b) as input, while the second program receives a block definition diagram (bddD), the initial block (b), and the requirement diagram.

Listing 11. Service consistency verification

```
let first = ref 0 ;;
let rec serviceConsistency b bddD  =  let suBb = ref (fsuB b bddD.suBs) in
  let result = ref true in
  if (S1 b.connectorsOutB)   &&  (S2 b)   then (result := true)  else (result := false ) ;
  if !first == 0 then
    if !result  then Printf.printf ``The consistency is verified : %s\n%!" b.name else  Printf
        .printf ``Error in the consistency : %s\n%!" b.name ; first := 1;
  suBb := supp b !suBb;
  while !suBb <> [] do
    match !suBb with
    |[] ->()
    |hd::l -> (
        if !result == false then (result := true) ;
        if fsuB hd bddD.suBs <> [] then
          result := !result &&  S1 hd.connectorsOutB && S2 hd;
        if !result  then Printf.printf ``The consistency is verified : %s\n%!" hd.name else
            Printf.printf ``Error in the consistency : %s\n%!" hd.name ;
        if fsuB hd bddD.suBs <> [] then result := !result && serviceConsistency hd bddD;
        suBb := supp hd !suBb; )
  done;
  match !result with
    true -> true;
  | false -> false ;;
```

Listing 12. Requirement consistency verification

```
let first = ref 0 ;;
let rec requirementConsistency b bdd reqD = let suBb = ref (suB b bdd.suBs) in
  let finalList = ref (AtomicReq b reqD)in
  let result = ref true in
  if !finalList = [] then  result := false else result := true;
  if !first2 == 0 then
    if !result  then Printf.printf ``The consistency is verified :  %s\n%!" b.name  else
        Printf.printf ``Error in the consistency : %s\n%!" b.name ;  first2 := 1;
  while !suBb <> [] do
    match !suBb with
    |[] -> ()
    |hd::l->
        (let finalList2 = ref (AtomicReq hd reqD) in
```

Just Accepted

```
        finalList := soustractionElement !finalList !finalList2;
        if suB hd bdd.suBs <> [] then result := requirementConsistency hd bdd reqD;
        if !result  then Printf.printf ``The consistency is verified : %s\n%!" hd.name else
            Printf.printf ``Error in the consistency : %s\n%!" hd.name ;         suBb := supp
             hd !suBb; )
  done;
  match (!result , !finalList) with
    true , [] -> true
  | _ , _ -> false;;
```

Received 18 April 2024; revised 27 August 2024; accepted 24 October 2024

Just Accepted