Joint use of SysML and Reo to specify and verify the compatibility of CPS components

Perla Tannoury[1,2], Samir Chouali[1,3], and Ahmed Hammad[1,4]

[1] University of Bourgogne Franche-Comté, France
FEMTO-ST Institute - UMR CNRS 6174
[2] perla.tannoury@femto-st.fr
[3] schouali@femto-st.fr
[4] ahammad@femto-st.fr

Abstract. Modeling and verifying the behavior of Cyber-Physical Systems (CPS) with complex interactions is challenging. Traditional languages such as SysML diagrams are not enough to capture CPS coordination. In this paper, we propose a novel approach called SysReo, which extends SysML diagrams (RD, BDD, IBD, SD) with the Reo coordination language. Our main objective is to enhance the interoperability of CPS by providing a more precise representation of system behavior and interaction protocols. To achieve this goal, we extend the SysML sequence diagram (SD) with Reo to create the SysReo SD. Through this integration, we bridge the gap between traditional modeling languages and the coordination demands of CPS. We develop an algorithm to generate Constraint Automata (CA) from SysReo SD, which ensures that CPS components can seamlessly work together. These automata are used in a verification tool that checks formulas expressed in Linear Temporal Logic (LTL). By leveraging LTL and Constraint Automata, we enhance the precision and rigor of CPS verification processes, while guaranteeing that CPS components can seamlessly work together. Furthermore, we apply our approach to a medical CPS case study, illustrating its effectiveness in identifying design flaws early and ensuring system behavior aligns with desired properties.

Keywords: CPS · SysReo · SysML · Reo · Constraint Automata · LTL · Specification · Verification.

1 Introduction

In today's technologically advanced world, Cyber-Physical Systems (CPS) have become crucial in a range of applications, such as autonomous vehicles [38], modeling smart city software interactions [39], and healthcare systems [1]. These systems combine the physical and digital worlds, resulting in improved automation, control, and data processing. Despite the importance of CPS, modeling them can be challenging, especially in the healthcare and medical sectors, as it involves integrating various system components, behaviors, and interaction protocols. In addition, collaborative efforts between designers, developers, and stakeholders

are required to address these challenges. As CPSs continue to become more complex, it is essential to establish an environment that facilitates the modeling process while highlighting their fundamental, structural, and behavioral aspects.

Several languages and formalisms are used to model CPS [34,35,17,23]. In our research, we have opted to use the System Modeling Language (SysML) [37] language due to its ability to model heterogeneous systems that combine hardware and software components. We aim to develop an approach that enables users to easily create specifications using SysML, while taking into account both verification and validation processes. SysML is widely used in industrial applications to model various aspects of a system, including its architecture, behavior, and requirements. However, CPSs are typically composed of various components that interact through various protocols, leading to complex system behaviors. While SysML is a valuable language for describing CPS, it may not be sufficient to formally specify and verify the complex interactions between CPS components. To tackle this problem, we have proposed in our previous work [41,42] a new domain-specific language (DSL) called `SysReo` that effectively uses the strengths of both semi-formal and formal languages to improve the validation and verification of CPS.

`SysReo`, is used to overcome the challenges in designing a CPS by clearly expressing its interaction protocols at any design stage. By extending SysML, a semi-formal language, with Reo [5], a formal coordination language, in the `SysReo` framework, it becomes possible to model the complex components of a CPS in an effective manner. This integration provides a powerful tool for representing component interactions, allowing for more precise and accurate modeling of complex systems. SysReo empowers CPS designers to model all facets of a CPS while explicitly defining the conditions under which data can flow between components. However, it should be noted that SysReo has limitations in formally specifying and verifying the behavioral aspects of CPS. In contrast, the SysML Sequence Diagram (SD) [37] excels in representing component interactions over time in CPS. Nonetheless, SD is semi-formal and lacks direct verification capabilities. On the other hand, Reo offers a formal representation of component coordination and allows for system property analysis. However, stakeholders may find Reo challenging to comprehend due to its complex representation.

In this paper, we first introduce a novel approach called "SysReo Sequence Diagram (SysReo SD)" that enhances the modeling and analysis of CPS. By extending the SysML Sequence Diagram (SD) with Reo notation, we create a "semi-formal-formal" model that captures the behavior and coordination of CPS components using an exogenous protocol. Unlike traditional SysML SD, which focuses on internal system behavior, SysReo SD imposes an external order on the flow of data between components without directly affecting their behavior. The SysReo SD model serves two main purposes: first, it bridges the gap between visual representation and formal modeling, providing a comprehensive view of system behavior and interaction protocols. Second, it enables the formal verification and analysis of the interoperability and correctness of the CPS. To facilitate the verification process, we develop an algorithm that generates Reo Constraint Automata (CA) [15] directly from the SysReo SD model. In the next phase, we

use the generated automaton CA as input in the vereofy [12] model checking tool to formally verify Linear Temporal Logic (LTL) [11] properties. This allows us to effectively verify the interoperability of CPS components, confirm compliance with specified requirements, and ensure correct system behavior. To demonstrate the effectiveness of our approach, we conducted a case study involving a smart medical bed, showcasing its potential for real-world applications in the medical CPS domain.

To the best of our knowledge, no previous research has comprehensively explored the extension of SysML Sequence Diagram (SD) with Reo to effectively model the behavior and interaction protocols of CPS. Existing approaches either resulted in verbose and less readable Reo circuits derived from scenario specifications [9,7], or encountered difficulties in establishing correlations between Reo circuits and the original specifications [36]. Our work addresses this gap by enhancing SysML SD with the coordination capabilities of Reo and introducing a novel algorithm that directly generates Constraint Automata (CA) from SysReo SD diagrams. This is followed by a formal verification process to ensure CPS interoperability and validate design correctness.

The paper is structured as follows. Section 2 provides a concise introduction to Reo, Constraint Automata, and SysReo, highlighting their key concepts and features. In Section 3, we present the related works. Moving forward, Section 4 offers an in-depth case study that showcases the practical application of our SysReo model, focusing on the specification and verification processes involved. Finally, Section 5 concludes the paper and briefly discusses future work.

2 Preliminaries

In this section, we give a brief introduction to Reo, constraint automata (CA), and SysReo.

2.1 Reo and constraint automata (CA) in a nutshell

Reo, as described by Arbab in [5], is an external coordination model that prioritizes efficient communication and coordination among different components. It achieves this by using channel-based connectors to establish complex coordinators. However, Reo does not focus on internal activities and communications within individual components. Instead, its main emphasis is on the coordination and interaction between components. The fundamental elements of Reo consist of components, channels, nodes, and ports, working in harmony to enable seamless data exchange and synchronization between these components [5,8,10].

The formal semantics of Reo are rigorously captured through the use of Constraint Automata (CA) [15]. CA provides a systematic representation of interactions among anonymous components, describing behavior and data flow in coordination models. This formalism involves labeling transitions with sets of ports that are triggered simultaneously, complemented by data constraints applied to these ports. As a result, constraint automata offers a powerful means to

precisely specify and analyze component interactions within the Reo coordination model.

Definition 1 *Constraint Automata (CA):* A constraint automaton B = (S,S_0,N,δ) is composed of:

- S: set of states (or locations).
- S_0: initial state where $S_0 \in S$.
- N: set of port names.
- δ: transition relation $\delta \subseteq S \times 2^N \times DC \times S$, where DC is the set of Data Constraints (DC) over a finite data domain Data.

An example of a constraint automata B is illustrated in **Fig.6 step2**, where B= ({S_0, S_1}, S_0, {(A,V), (B,W)}, {(S_0, (A,V), [A]=$|V|$, S_1), (S_1, (B,W), [B]=$|W|$, S_0)}).

2.2 SysReo

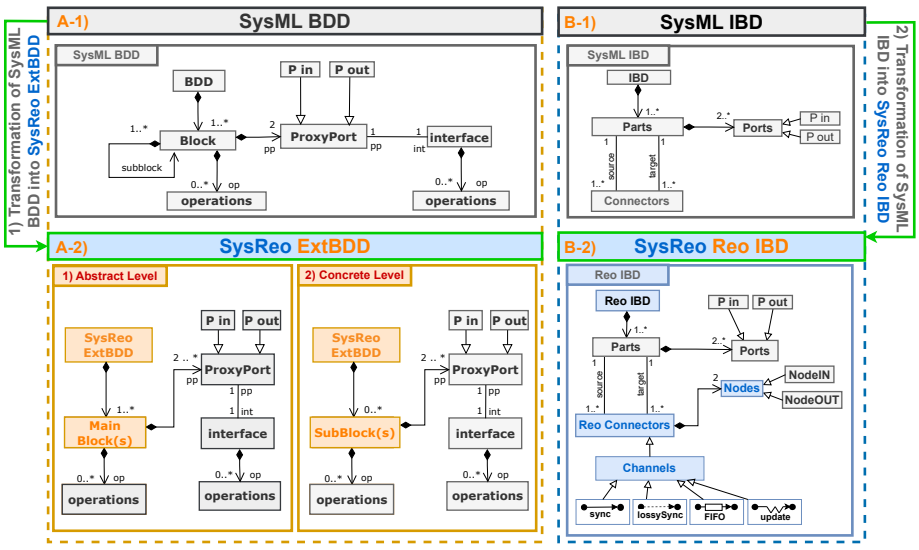


Fig. 1. From SysML diagrams to SysReo diagrams.

`SysReo` was introduced in [41,42] as a powerful modeling language that combines the strengths of both SysML and Reo to provide a more comprehensive and flexible approach to model a CPS. One of the main advantages of `SysReo` is its ability to model all facets of CPS, from requirements to architecture and interaction protocols, which is essential for designing complex heterogeneous systems. **Fig. 1** represents the process of transforming the SysML Block Definition Diagram (BDD) and Internal Block Diagram (IBD) into SysReo Extended Block

Definition Diagram (ExtBDD) and Reo Internal Block Diagram (Reo IBD). The transformation consists of two parts:

(A) The SysML BDD (Fig. 1 A-1) is transformed into SysReo ExtBDD (Fig. 1 A-2) by using SysML BDD meta-models and dividing the CPS hierarchy into two levels. The first level presents the abstract model of the CPS, where the primary components of the system are modeled as main blocks. The second level presents the concrete components of the CPS, where they are modeled as sub-blocks. Overall, the traditional SysML BDD offers a high-level view of the system being designed, whereas the Extended BDD offers a more in-depth view of the system structure. Using multiple levels of abstraction and additional information in the ExtBDD can help to control the complexity of the system being designed and ensure that the system design meets the intended requirements.

(B) The SysML IBD (Fig. 1 B-1) is converted into SysReo Reo IBD (Fig. 1 B-2) by using SysML IBD meta-models and replacing the IBD connectors with "Reo connectors". This replacement allows for more explicit representation of the internal composition of components and their interaction protocols by setting constraints on data flow, whereas SysML IBD connectors only provide a generic way of depicting the interactions between components. With Reo connectors, the CPS designer can more accurately capture the specific communication and synchronization patterns between components, and thereby improve the reliability, safety, and performance of the system. In addition, Reo connectors have formal semantics, making them precise and verifiable using formal methods. Overall, the Reo IBD diagram improves system reliability and can save time and cost by detecting errors early in the development process.

In summary, `SysReo` offers a more comprehensive and adaptable approach to system design, which results in more effective and reliable CPS. The use of ExtBDD provides a more precise and detailed hierarchical view of the system, while Reo IBD allows for explicit modeling of the internal composition of the system and the interaction protocols among its components. Although SysReo has many advantages in modeling the structure and internal composition of the CPS, it falls short of capturing the behavioral and coordination aspects of the CPS. To bridge this gap, we present a new approach named SysReo SD in this paper. Section 4.3 provides a comprehensive overview of SysReo SD, focusing on its ability to effectively address the behavioral and coordination challenges in CPS.

3 Related works

CPSs are networks of different embedded systems connected in a physical environment, making their specification and formal verification difficult due to their complex and large-scale computing infrastructure.

Previous works in [22,3,33,27] proposed different approaches to model different aspects of CPS with SysML. To formally verify critical safety systems,

the authors in [19,18] proposed to extend the SysML sequence diagram (SD) and automate consistency verification using the Clock Constraint Specification Language (CCSL) [4]. Additionally, an approach was proposed [16] to transform SD into interface automata (IA) to verify the compatibility and consistency of components modeled with SysML. However, these works take an endogenous approach to coordination and neglect the coordination of message exchanges, leading to communication problems within CPSs. To address this problem, Reo is proposed as a coordination language to fill the interfacing gaps and enhance the modeling and coordination of CPSs.

Various researchers explored different approaches to model CPS using Reo and have demonstrated its effectiveness through formal analysis techniques. These include co-algebraic semantics [10], operational semantics [15] using constraint automata [15] and timed constraint automata [6,31], coloring semantics [21], and converting Reo models to other formal models such as Alloy [29] and mCRL2 [32] to leverage existing verification tools. Despite its advantages, Reo remains complex for stakeholders to comprehend due to its lack of semi-formal representation.

To our knowledge, there is no prior research that explores the extension of SysML Sequence Diagrams (SD) with Reo to effectively model behavior and interaction protocols in CPS. Previous approaches resulted in complex and less comprehensible Reo circuits derived from scenario specifications [9,7], or faced challenges in establishing connections between the Reo circuits and the original specifications [36]. In our work, we bridge this gap by enhancing SysML SD with Reo's coordination capabilities and introducing a novel algorithm that directly generates Constraint Automata (CA) from SysReo SD diagrams. This is followed by a formal verification process to ensure interoperability of the CPS and validate the accuracy of the design, addressing the limitations of existing approaches.

4 Case study: Smart Medical Bed (SMB)

In this section, we present our case study of the Smart Medical Bed (SMB) system. First, we begin by briefly introducing the SMB system. Then, we gather information about the SMB system and analyze it using our `SysReo` models. This process involves specifying the system's requirements, designing its structure and internal composition, and modeling the system's behavior and interaction protocol. Finally, we move on to the verification phase, where we rigorously verify the correctness of our SysReo models.

4.1 SMB overview

A Smart Medical bed `(SMB)` is equipped with various sensors and monitoring devices that collect data on vital signs of the patient, such as heart rate, blood pressure, temperature, oxygen saturation, and other key indicators of their health. These data are then transmitted to a Remote Terminal Unit `(RTU)`, where it can be stored and analyzed by healthcare providers as we can see in **Fig. 2**. `RTU`

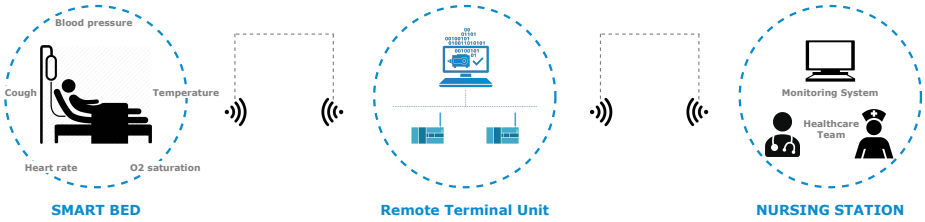


Fig. 2. Smart medical bed architecture.

plays a key role in connecting, controlling, analyzing, and communicating data from the smart bed to the nursing station, providing valuable information to the healthcare team and ultimately improving patient care. Continuous monitoring helps identify potential health issues early, leading to informed decision-making and efficient care delivery. The `SMB` infrastructure comprises the following components: (1) the `Smart Bed (SB)`, where the patient resides, (2) the `Remote Terminal Unit (RTU)`, where the collected data are stored, processed and analyzed, and (3) the `Nursing Station (NS)`, where the monitoring system and the healthcare team are located. In this article, we focus on modeling and verifying the requirement, structure, behavior, and interaction protocol between the Smart Bed (SB) and the RTU in the SMB system. Using our model-driven approach, SysReo, we can analyze the system's requirements, model the architecture and internal structure of the SMB. To handle complexity, we introduce SysReo SD, an extension for modeling complex component behavior and interaction protocols.

4.2 Modeling SMB with SysReo

In this section, we focus on our modeling approach. As shown in **Fig. 3**, the process is broken down into two phases. During the first phase, the CPS designer begins by collecting requirements about the system and analyzing it. Using our `SysReo` model, the designer then specifies the system's needs, which results in three main diagrams: (1) The requirement diagram that models the functional and non-functional needs of the system. (2.1) The `ExtBDD` diagram that represents the hierarchical structure of the system as blocks, followed by (2.2) the `Reo IBD` diagram that is used to model the system's internal structure and interaction protocols. (3) The SysReo sequence diagram (SysReo SD) used to model the behavior and coordination of CPS components. Finally, to satisfy our predefined requirements, we link them to the Reo IBD diagram, and to verify them, we establish a link to SysReo SD.

In the second phase, we focus on the verification process. To verify the interoperability of CPS components, `SysReo SD` is translated into CA through our algorithm that directly generates CA from SysReo SD. Then the requirements to `Reo IBD` are formally defined through property formulas such Linear Temporal Logic (LTL) [11] for verification. This step is crucial to achieve the

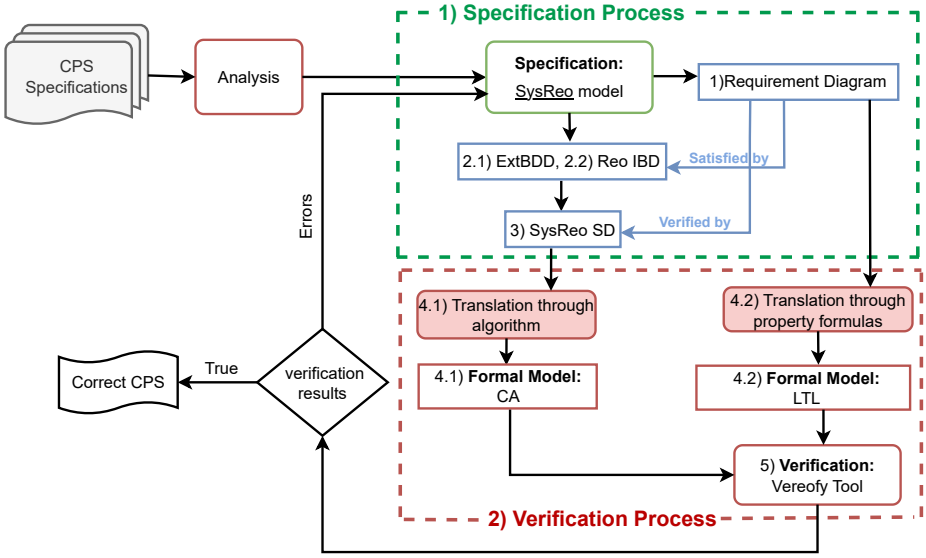


Fig. 3. Specification and verification process.

objective of our modeling approach: creating a precise representation of the CPS system's needs, behavior, and coordination. By generating CA and translating the requirements into LTL formulas, the CPS designer can ensure both the interoperability of the CPS components and the accurate representation of requirements. Furthermore, the formal verification of these requirements can be seamlessly conducted using specialized verification tools like vereofy [12,14]. This helps to guarantee that the CPS system will function as intended and meets the designer's requirements.

After evaluating the verification results, the CPS designer checks if there are any specification errors. If so, the process returns to the `SysReo` model specification phase until a correct CPS model is obtained.

4.3 Specification process: SysReo models

Requirement The design process for any system is crucial for ensuring its functionality and usability. The first step in this process is to identify the specific needs of the system, as outlined in the requirements table such as **Table 1**. In this table, we only present two functional requirements of the `SMB` system to guarantee the proper flow of data between its components. This is essential to ensure that the system runs smoothly and meets the needs of its users. For example in **Table 1**, requirement R1 states that the smart bed must constantly send temperature data to the RTU component. This requirement ensures that the RTU receives up-to-date temperature readings from the smart bed (SB) and it is satisfied by the SB component.

Table 1. Requirement table of SMB.

Req ID	Requirement description	Satisfied by
R1	The “SB” must constantly send temperature data to the “RTU” component using ”sendTempData” message.	SB
R2	The “RTU” shall respond to the “SB” component with an “ack” message.	RTU

Based on the requirements of **Table 1**, we can identify the main components of the `SMB` system that are the `Smart Bed (SB)` and the `Remote Terminal Unit (RTU)`. In the next section (Section 4.3), these components will be used to create a hierarchical view of the `SMB` system, to better understand the relationships and overall functioning of the system.

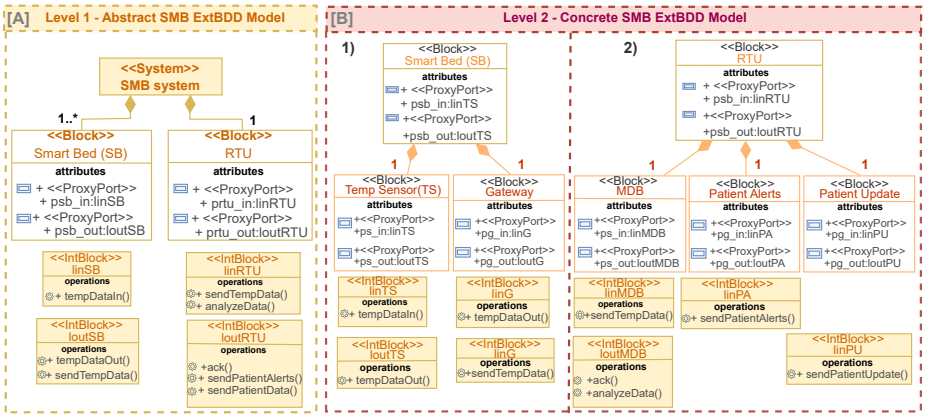
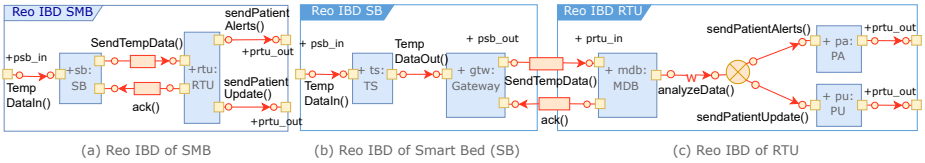


Fig. 4. The ExtBDD model of the SMB system.

ExtBDD The Extended Block Definition Diagram is used to model the hierarchical view of the `SMB` system where each component is modeled as a block. The block defines a component by describing its internal operations as private operations of the block and also its required and offered services as follows. Each block is decorated by two proxy ports: 1) Input port that provides information on the services that are available. These services are listed in an interface block which specifies the type of the port. 2) Output port that describes the required services in a similar manner. **Fig. 4** **[A]** represents the abstract part of the system by only showing the main components. It consists of the block named `"SMB"` that represents the system as a whole. It is decomposed into two sub-blocks: `Smart Bed (SB)` and the `Remote Terminal Unit (RTU)`, in which

it is linked to them by the composition relationship. **Fig. 4** [**B**] shows the concrete level of the `SMB` system. It depicts the sub-components that make up each main component. As an example, the smart bed component is broken down into two blocks, a `Temperature Sensor` and a `Gateway`. The main function of the `Temperature Sensor` is to continuously measure, record, gather and transmit the measured data to the Gateway.



(a) Reo IBD of SMB (b) Reo IBD of Smart Bed (SB) (c) Reo IBD of RTU

Fig. 5. Reo IBD of SMB, SB, and RTU.

Reo IBD The Reo Internal Block Diagram is a combination of the coordination language Reo and the SysML Internal Block Diagram (IBD) [25] where the IBD's connectors are translated into Reo circuits. `Reo IBD` is used to characterize the internal components and structure of a system block, including its properties, parts, connections, and interaction protocols. It explicitly specifies the rules and conditions of the data that can be transferred from the component's input to its output through channels. In **Fig. 5(a)**, two different channels are used to model the interaction protocols among the following main components: the Smart Bed (SB) and Remote Terminal Unit (RTU).

1. FIFO channel: for example, two FIFO channels "sendTempData()" and "ack()" are used to model the asynchronous communication between the two components "SB" and "RTU". The FIFO channel helps to efficiently manage memory in a real-time system by processing the oldest data first, thus preventing loss of information.
2. Sync channel: used to model the synchronization properties among components.

Fig. 5(b) models the internal structure of the smart bed. First of all, the "Temperature Sensor (TS)" sends the vital signs data of the patients to the "Gateway" component via the "TempDataOut()" flow port using a sync channel. Once received, the "Gateway" component sends the data, which has been pre-processed, to the RTU through the "Medical Data Base (MDB)" component using a FIFO channel "SendTempData()". Once it is received, the MBD replies to the smart bed component with an "ack()" using a fifo channel. Then in **Fig. 5(c)**, the "MDB" component analyzes the data and sends "analyzeData()" to `xrouter` component ($\otimes$) via a filter channel where it models the routing replication of data to "Patient Alerts (PA)" or to "Patient Update (PU)". Once data enters the `xrouter` component, it is sent either to "PA" or to "PU" but never to both.

Reo IBD has many advantages when it comes to modeling a CPS. Such as effectively modeling component connections, characterizing message flow, satisfying predefined requirements, and enabling design flexibility while simplifying documentation with its graphical representation. However, it is limited to modeling temporal behaviors and formally verifying predefined requirements, which are crucial aspects in correctly modeling a CPS. Therefore, in the next section, we will extend the SysML sequence diagram (SD) with Reo and explore its benefits on CPSs.

SysReo SD SysReo Sequence Diagram extends SysML SD with Reo notation, including the introduction of the Reo Sequencer as an intermediary component. This integration enriches the representation of message exchange, coordination, and synchronization in a unified manner. In contrast to conventional SysML sequence diagrams, SysReo SD facilitates the explicit specification of protocols, eliminating the need for manual implementation of locks and buffers. This approach enhances accuracy, efficiency, and mitigates errors typically associated with manual synchronization mechanisms. As a result, SysReo SD offers a robust and comprehensive approach for specifying protocols in system behavior, particularly advantageous for capturing communication flow and coordination patterns within a single diagram. **Fig. 6 (A)** presents a simple example of a

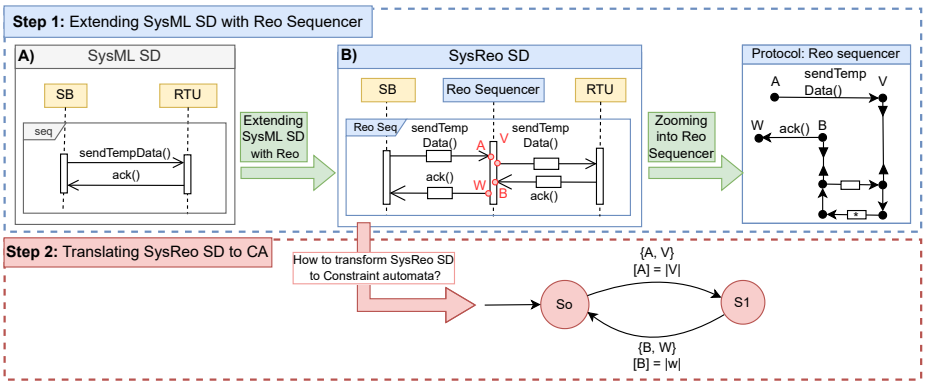


Fig. 6. Extending SysML SD with Reo then translating SysReo SD to CA.

SysML SD, demonstrating the behavior and message exchange between the two components of SMB, smart bed (SB) and RTU. This coordination follows an endogenous approach where protocols are implicitly expressed within the embedded code fragments. However, modifying these protocols can be challenging and may require extensive changes to multiple software components, potentially impacting previously validated properties.

On the other hand, exogenous methods like Reo offer a more explicit and modular approach to define protocols. In the given scenario **Fig. 6 (A)**, where

components SB and RTU need to exchange messages, an endogenous approach would involve directly implementing the exchange within their respective code. However, in **Fig. 6 (B)** illustrates a different strategy, where a separate component called the "Reo sequencer" explicitly defines the message exchange protocol between SB and RTU. For example, SB sends "sendTempData()" to RTU using reo ports {A, V} (depicted as red circles), and the Reo sequencer coordinates this message exchange. This decoupling allows for easier protocol modifications without affecting the implementation of SB and RTU. Employing Reo connectors in exogenous approaches provides more flexibility and simplifies the specification and adjustment of complex protocols in CPS.

The next step involves converting SysReo SD into constraint automata. To achieve this, we will first present a formal definition of SysReo SD. Subsequently, in Section 4.4, we will outline an algorithm that facilitates the automatic generation of constraint automata from SysReo SD.

Formal definitions of SysReo SD This section presents the formal definition of SysReo message and SysReo SD.

Definition 2 *(SysReoMes).* A SysReo message is a tuple SysReoMes = ($comp_s$, action, $comp_f$, P, Σ) where:

- $comp_s$: is the source component of the SysReo message.
- *action*: is the called method.
- $comp_f$: is the target component of the SysReo message.
- P: is the set of Reo ports.
- Σ = input, output is the set of synchronization constraints, specifying the allowed input/output actions on Reo ports.

For example in **Fig.6 (B)** a SysReo message between the component SB and RTU can be defined as SysReoMes= (SB, sendTempData, RTU, {A, V}, {[A],$|V|$}).

Definition 3 *(SysReo SD).* A SysReo SD is defined as tuple B = (IM, SysReoMes, ReoSeq, ReoLoop, ReoAlt) is composed of:

- IM:is the initial message.
- $SysReoMes$: the set of messages in SysReo SD.
- $ReoSeq$ = ($ReoSeq_1$, ...,$ReoSeq_i$, ... ,$ReoSeq_n$) is the list of reo sequencer combined fragments. $ReoSeq_i$=(obj_1, ... , obj_i, ...,obj_n), obj_i is message or a fragment and card($ReoSeq_i$) ≥ 2.
- $ReoLoop$ = ($ReoLoop_1$, ...,$ReoLoop_i$, ... ,$ReoLoop_n$) is the list of reo loop combined fragments. $ReoLoop_i$=(obj_1, ... , obj_i, ...,obj_n), obj_i is message or a fragment and card($ReoLoop_i$) ≥ 1.
- $ReoAlt$= ($ReoAlt_1$, ...,$ReoAlt_i$, ... ,$ReoAlt_n$) is the list of reo alternator combined fragments. $ReoAlt_i$=(obj_1, ... , obj_i, ...,obj_n), obj_i is message or a fragment and card($ReoAlt_i$) ≥ 2.

In this paper, we focus on the Reo sequencer from the SysReo SD fragments (ReoSeq, ReoLoop, and ReoAlt) mentioned above. Our main objective is to translate SysReoSD=($IM, SysReoMes, ReoSeq$) into reo constraint automata

using our algorithm. This automaton will serve as input to vereofy tool where we can formally verify the correctness and interoperability of SMB system through LTL properties.

4.4 Verification process: CA, Vereofy, LTL

This section outlines the verification process of SysReo. First, we start by proposing an algorithm to construct CA from SysReo SD diagram. Then, the resulting automaton is used as input for the vereofy tool to assess the accuracy and interoperability of the SMB system through the application of LTL properties.

Algorithm SysReo SD is a visual representation of the SMB system, which consists of components and their interactions. However, to analyze and verify the system formally, it is necessary to transform this visual representation into a more structured and formal representation. Therefore through our algorithm, we provide a systematic and automated approach to convert the SysReo SD into constraint automata, which are well-suited for formal analysis.

Algorithm 1 is developed to transform a SysReo SD into a Constraint Automaton (CA), using a SysReo message and fragment list as inputs. The resulting CA comprises a set of states (S), an initial state (S_0), a set of port names (N), and a set of transitions (δ). The algorithm begins by initializing the set of states, checking for emptiness, and creating a new state as the initial state (cf. Algorithm 1, lines 2-7). It then iterates through each object in the list. If the object is part of a Reo sequencer fragment, the algorithm recursively calls itself with the corresponding sub-list representing that fragment (cf. Algorithm 1, lines 8-13). When encountering a SysReo message object ($SysReoMes$), the algorithm creates a new state s' if the message is not the last object in the list (cf. Algorithm 1, lines 14-18). If it is the last object, s' is designated as the initial state s_0. The algorithm then populates the sets S and N in the CA with the newly created state and the ports of the $SysReoMes$, respectively. A new transition is created from the previously added state s to s', incorporating the port name and constraints from the $SysReoMes$, and it is subsequently added to the set of transitions δ. This process continues until there are no remaining objects in the list to be processed (cf. Algorithm 1 line 19→30).

The algorithm, SysReoSDtoCA, has a linear complexity dependent on the size of the objects set l in the SysReoSD diagram specification. When we apply this algorithm to the formal model of the SysReoSD diagram example, as presented in **Fig.6 B**, we obtain the constraint automata CA=(S,S_0,N,δ) described in **Fig.6 step2**. The CA is characterized by the following components:

1. Set of states: S=$\{S_0$, S$\}$.
2. Initial state: $S_0 = S_0$.
3. Set of port names: N= {(A,V), (B,W)}.
4. Set of transitions: δ= S x $2^{A,V}$ x DC x S_0 where DC = [A], $|V|$, and the second δ= S_0 x $2^{B,W}$ x DC x S where DC = [B], $|W|$.

Algorithm 1: Mapping SysReoSD to CA algorithm

```
1  Function SysReoSDtoCA(SysReoSD, l, CA)
   Input: SysReoSD = (IM, SysReoMes, ReoSeq);
   l: a list of objects in SysReoSD;
   l = (obj_1, ..., obj_i, ..., obj_n),
   obj_i is a message or a fragment in SysReoSD.
   Output: CA=(S,S_0, N, δ)
2  Begin
3  if S = ∅ then
4  |  s = createNewState()
5  |  S = S ∪ {s} // add s to S (the set of states in CA)
6  |  S_0 = s // set the initial state S_0 to s
7  end
8  while (l ≠ ∅) do
9  |  Let obj the first element in l
10 |  if obj ∈ ReoSeq then
11 |  |  Let l' be a list of objects composing the Reo sequencer fragment
12 |  |  SysReoSDtoCA(SysReoSD, l', CA)
13 |  end
14 |  else
15 |  |  if obj ∈ SysReoMes then
16 |  |  |  if card(l) > 1 then
17 |  |  |  |  s'= createNewState()
18 |  |  |  end
19 |  |  |  else
20 |  |  |  |  s'= S_0
21 |  |  |  end
22 |  |  |  S=S ∪ {s, s'} // the set of states
23 |  |  |  N=N ∪ obj.P // the set of port names
24 |  |  |  δ= δ ∪ (s, obj.P, obj.Σ, s') // the set of transitions
25 |  |  |  s=s' // initial state
26 |  |  end
27 |  end
28 |  l=l'- {obj}
29 end
30 End SysReoSDtoCA
```

Upon analyzing the resulting constraint automaton, we can observe the system's behavior when component SB sends the SysReoMes "sendTempData" to component RTU. This communication is coordinated by the Reo sequencer and specified by the transition labeled with reo ports {A, V} and input/output action interfaces [A], $|V|$. As a response, component RTU sends the SysReoMes "ack" back to component SB through the reo sequencer ports {B, W} and input/output action interfaces [B], $|W|$.

Vereofy tool Vereofy [12,14], developed at the University of Dresden, is a powerful model checking tool specifically designed for analyzing and verifying Reo connectors. It supports two input languages: the Reo Scripting Language (RSL) for specifying coordination protocols, and the Constraint Automata Reactive Module Language (CARML), a textual representation of constraint automata used to define component behavior. With vereofy, one can verify temporal properties expressed in LTL [11] and CTL-like logics [30]. Distinguishing itself from other model checkers [2,20,26], vereofy places a primary focus on verifying coordination aspects, communication, and interactions at the behavioral interface level. It employs a symbolic representation based on binary decision diagrams (BDDs) to facilitate efficient verification algorithms. For a detailed understanding of the modeling languages and verification techniques employed by vereofy, refer to [13,12].

Next, we present the CARML code corresponding to the generated Constraint Automaton (CA) from SysReoSD (cf. Appendix A Fig.7 to see the execution of the CARML code in vereofy followed by the its BDD output in Fig.8). The code module provided below specifically defines a sysreoCA. Initially, the sysreoCA is in an empty state denoted as S_0. When the sysreoCA is not in a full state S_1, and a data value is written to its input port A, the data is stored in the 'sendTempData' variable, and the internal state changes from S_0 to S_1. Another component reading data from the output port V resets the internal state back to S_0. The data domain has been locally set to the integer range (0,1). Although it is possible to set the data domain to any other available datatype, it can only be done once across the included files or as a runtime argument. The data flow within the system may depend on the value of a variable of type 'Data', as illustrated in the second transition, where the 'sendTempData' stored is written to the output port V, represented as: $\#V == sendTempData$.

```
#Vereofy CARML code:
TYPE Data = int(0,1);

MODULE sysreoCA{
// initializing the I/O ports {A,V} and { B,W}
 in: A;
 out: V;
 in: B;
 out: W;

//defining the set of states that are S0 and S1
 var : enum {S0, S1} state := S0;

//defining the messages that should be exchanged
 var : Data sendTempData := 0;
 var : Data ack  := 0;

//drawing the transitions from So to S1 and from S1 to S0
```

```
 state == S0  -[ {A} ]-> state:=S1 & sendTempData:=#A;
 state == S1 -[{V} & #V == sendTempData ]-> state:=S0;

 state == S0  -[ {B} ]-> state:=S1 & ack:=#B;
 state == S1  -[{W} & #W == ack ]-> state:=S0;
}
```

Verification of LTL properties LTL-based model checking rigorously verifies system properties, boosts confidence in correctness and reliability, and identifies design flaws for improvements. To verify the correctness of the specified protocol and the interoperability between the smart bed and RTU, we propose to check the following LTL properties in vereofy tool:

```
p1:LTL<<G(("{A}" & "#A==1")->X("state==S1"& "sendTempData==1"))>>
/*PASSED*/

p2:LTL<< G(("{W}" & "#W==1")-> X("state==S0" & "ack==1"))>>
/*PASSED*/
```

The evaluated LTL property in "p1" ensures that when port A is active once, the subsequent state must satisfy two conditions: the state variable should be S_1 (indicating a transition from S_0 to S_1) and the 'sendTempData' variable should be 1 (indicating successful message transmission). The result, "PASSED", confirms that this property holds for all execution traces (cf. Appendix B Fig.9). The same thing applies for the evaluated LTL property in "p2" where it indicates the successful transmission of the "ack" message (cf. Appendix B Fig.10).

5 Conclusion

Our paper introduces a novel diagram called "SysReo SD" that enhances CPS modeling and analysis. By extending SysML with Reo, we create a powerful "semi-formal-formal" model that effectively captures the behavior and coordination of CPS components using an exogenous protocol. This allows us to ensure CPS interoperability, meet specific design requirements, and validate the correctness of the system behavior. Furthermore, we illustrate the applicability of our approach through a case study in the medical CPS domain, showcasing the potential benefits of employing SysReo.

Looking ahead, we are planning to explore the application of SysReo in Digital Twins (DT) [28] where we can accurately capture the interactions and behaviors of the components of a physical system in a virtual environment. With the use of SysReo models, we can continuously monitor and optimize the performance of the digital twin.

References

1. Al-Jaroodi, J., Mohamed, N., Abukhousa, E.: Health 4.0: on the way to realizing the healthcare of the future. Ieee Access **8**, 211189–211210 (2020)
2. Alur, R., De Alfaro, L., Grosu, R., Henzinger, T.A., Kang, M., Kirsch, C.M., Majumdar, R., Mang, F., Wang, B.Y.: jmocha: A model checking tool that exploits design structure. In: Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001. pp. 835–836. IEEE (2001)
3. Amálio, N., Payne, R., Cavalcanti, A., Brosse, E.: Foundations of the sysml profile for cps modelling. Deliverable D2. 1a, version **1** (2015)
4. André, C.: Syntax and semantics of the clock constraint specification language (CCSL). Ph.D. thesis, INRIA (2009)
5. Arbab, F.: Reo: a channel-based coordination model for component composition. Mathematical Structures in Computer Science **14**(3), 329–366 (2004). https://doi.org/10.1017/S0960129504004153
6. Arbab, F., Baier, C., de Boer, F., Rutten, J.: Models and temporal logical specifications for timed component connectors. Software & Systems Modeling **6**, 59–82 (2007)
7. Arbab, F., Baier, C., de Boer, F., Rutten, J., Sirjani, M.: Synthesis of reo circuits for implementation of component-connector automata specifications. In: Jacquet, J.M., Picco, G.P. (eds.) Coordination Models and Languages. pp. 236–251. Lecture Notes in Computer Science, vol 3454. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11417019˙16
8. Arbab, F., Baier, C., Rutten, J., Sirjani, M.: Modeling component connectors in reo by constraint automata. Electronic Notes in Theoretical Computer Science **97**, 25–46 (2004)
9. Arbab, F., Meng, S.: Synthesis of connectors from scenario-based interaction specifications. In: CBSE. vol. 8, pp. 114–129. Lecture Notes in Computer Science, vol 5282. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87891-9˙8
10. Arbab, F., Rutten, J.: A coinductive calculus of component connectors. In: WADT. vol. 2755, pp. 34–55. Lecture Notes in Computer Science, vol 2755. Springer, Berlin, Heidelberg (2002). https://doi.org/10.1007/978-3-540-40020-2˙2
11. Babenyshev, S., Rybakov, V.: Linear temporal logic ltl: basis for admissible rules. Journal of Logic and Computation **21**(2), 157–177 (2011)
12. Baier, C., Blechmann, T., Klein, J., Klüppelholz, S.: Formal verification for components and connectors. In: Formal Methods for Components and Objects: 7th International Symposium, FMCO 2008, Sophia Antipolis, France, October 21-23, 2008, Revised Lectures 7. pp. 82–101. Lecture Notes in Computer Science, vol 5751. Springer, Berlin, Heidelberg. (2009). https://doi.org/10.1007/978-3-642-04167-9˙5
13. Baier, C., Blechmann, T., Klein, J., Klüppelholz, S.: A uniform framework for modeling and verifying components and connectors. In: Coordination Models and Languages: 11th International Conference, COORDINATION 2009, Lisboa, Portugal, June 9-12, 2009. Proceedings 11. pp. 247–267. Lecture Notes in Computer Science, vol 5521. Springer, Berlin, Heidelberg. (2009). https://doi.org/10.1007/978-3-642-02053-7˙13
14. Baier, C., Blechmann, T., Klein, J., Klüppelholz, S., Leister, W.: Design and verification of systems with exogenous coordination using vereofy. In: Leveraging Applications of Formal Methods, Verification, and Validation: 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October

18-21, 2010, Proceedings, Part II 4. pp. 97–111. Lecture Notes in Computer Science, vol 6416. Springer, Berlin, Heidelberg. (2010)
15. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in reo by constraint automata. Science of computer programming **61**(2), 75–113 (2006)
16. Bouaziz, H., Chouali, S., Hammad, A., Mountassir, H.: Sysml model-driven approach to verify blocks compatibility. International Journal of Computer Aided Engineering and Technology **11**(2), 206–231 (2019). https://doi.org/10.1504/IJCAET.2019.098134, https://www.inderscienceonline.com/doi/abs/10.1504/IJCAET.2019.098134
17. Bouskela, D., Falcone, A., Garro, A., Jardin, A., Otter, M., Thuy, N., Tundis, A.: Formal requirements modeling for cyber-physical systems engineering: An integrated solution based on form-l and modelica. Requirements Engineering **27**(1), 1–30 (2022)
18. Chen, X., Liu, Q., Mallet, F., Li, Q., Cai, S., Jin, Z.: Formally verifying consistency of sequence diagrams for safety critical systems. Science of Computer Programming **216**, 102777 (2022)
19. Chen, X., Mallet, F., Liu, X.: Formally verifying sequence diagrams for safety critical systems. In: 2020 International Symposium on Theoretical Aspects of Software Engineering (TASE). pp. 217–224. IEEE (2020)
20. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: A new symbolic model verifier. In: Computer Aided Verification: 11th International Conference, CAV'99 Trento, Italy, July 6–10, 1999 Proceedings 11. pp. 495–499. Lecture Notes in Computer Science, vol 1633. Springer, Berlin, Heidelberg. (1999). https://doi.org/10.1007/3-540-48683-6˙44
21. Clarke, D., Costa, D., Arbab, F.: Connector colouring i: Synchronisation and context dependency. Science of Computer Programming **66**(3), 205–225 (2007)
22. DeTommasi, G., Vitelli, R., Boncagni, L., Neto, A.C.: Modeling of marte-based real-time applications with sysml. IEEE Transactions on Industrial Informatics **9**(4), 2407–2415 (2012)
23. Genius, D., Apvrille, L.: Hierarchical design of cyber-physical systems. In: Modelsward (2023)
24. Graphivz: Graphivz. https://shorturl.at/guvX2
25. Hause, M., et al.: The sysml modelling language. In: Fifteenth European Systems Engineering Conference. vol. 9, pp. 1–12 (2006)
26. Holzmann, G.J.: The model checker spin. IEEE Transactions on software engineering **23**(5), 279–295 (1997)
27. Huang, P., Jiang, K., Guan, C., Du, D.: Towards modeling cyber-physical systems with sysml/marte/pccsl. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). vol. 1, pp. 264–269. IEEE (2018)
28. Juarez, M.G., Botti, V.J., Giret, A.S.: Digital twins: Review and challenges. Journal of Computing and Information Science in Engineering **21**(3), 030802 (2021)
29. Khosravi, R., Sirjani, M., Asoudeh, N., Sahebi, S., Iravanchi, H.: Modeling and analysis of reo connectors using alloy. In: Coordination Models and Languages: 10th International Conference, COORDINATION 2008, Oslo, Norway, June 4-6, 2008. Proceedings 10. pp. 169–183. Lecture Notes in Computer Science, vol 5052. Springer, Berlin, Heidelberg. (2008). https://doi.org/10.1007/978-3-540-68265-3˙11
30. Kokash, N., Arbab, F.: Formal design and verification of long-running transactions with extensible coordination tools. IEEE Transactions on Services Computing **6**(2), 186–200 (2011)

31. Kokash, N., Jaghoori, M.M., Arbab, F.: From timed reo networks to networks of timed automata. Electronic Notes in Theoretical Computer Science **295**, 11–29 (2013)
32. Kokash, N., Krause, C., De Vink, E.: Reo+ mcrl2: A framework for model-checking dataflow in service compositions. Formal Aspects of Computing **24**(2), 187–216 (2012)
33. Larsen, P.G., Fitzgerald, J., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., et al.: Integrated tool chain for model-based design of cyber-physical systems: The into-cps project. In: 2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data). pp. 1–6. IEEE (2016)
34. Lin, J., Sedigh, S., Miller, A.: Modeling cyber-physical systems with semantic agents. In: 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops. pp. 13–18. IEEE (2010)
35. Mallet, F.: Marte/ccsl for modeling cyber-physical systems. Formal Modeling and Verification of Cyber-Physical Systems: 1st International Summer School on Methods and Tools for the Design of Digital Systems, Bremen, Germany, September 2015 pp. 26–49 (2015)
36. Meng, S., Arbab, F., Baier, C.: Synthesis of reo circuits from scenario-based interaction specifications. Science of Computer Programming **76**(8), 651–680 (2011)
37. OMG: OMG System Modeling Languag. https://www.omg.org/spec/SysML/, accessed: 10-02-2023
38. Panahi, V., Kargahi, M., Faghih, F.: Control performance analysis of automotive cyber-physical systems: A study on efficient formal verification. ACM Transactions on Cyber-Physical Systems (2022)
39. Pundir, A., Singh, S., Kumar, M., Bafila, A., Saxena, G.J.: Cyber-physical systems enabled transport networks in smart cities: Challenges and enabling technologies of the new mobility era. IEEE Access **10**, 16350–16364 (2022)
40. Sickert, S., Esparza, J., Jaax, S., Křetínskỳ, J.: Limit-deterministic büchi automata for linear temporal logic. In: International Conference on Computer Aided Verification. pp. 312–332. Lecture Notes in Computer Science, vol 9780. Springer, Cham. (2016). https://doi.org/10.1007/978-3-319-41540-6˙17
41. Tannoury, P.: An Incremental Model-Based Design Methodology to Develop CPS with SysML/OCL/Reo. In: Journées du GDR GPL. Vannes, France (Jun 2022), https://hal.science/hal-03893454
42. Tannoury, P., Chouali, S., Hammad, A.: Model driven approach to design an automotive cps with sysreo language. In: Proceedings of the 20th ACM International Symposium on Mobility Management and Wireless Access. pp. 97–104 (2022)

A Vereofy and BDD

In this section, we provide screenshots of how our CARML code runs in the vereofy tool and produces a BDD diagram.

Fig.7 illustrates the process of running a CARML file in vereofy, generating a Binary Decision Diagram (BDD) as a dot file. The resulting BDD diagram, depicted in **Fig.8**, represents the various possible paths or options within the system. In contrast, the constraint automata (CA) provides a comprehensive overview of system behavior and component interactions, demonstrating how

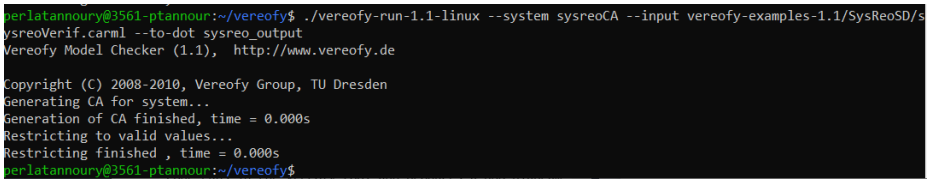


Fig. 7. running CARML code in Vereofy.

all the pieces fit together. The BDD, on the other hand, serves as a map that showcases the different possible routes that the system can take. To visualize the BDD graphically, one can test the dot file using Graphviz online [24].

B LTL properties

Vereofy relies on the external tool LTL2BA for translating LTL formulas into Buchi automata [40]. To use LTL model checking in vereofy, we need to ensure the availability of a functional LTL2BA executable. Vereofy expects the LTL2BA executable to be in the same directory or accessible through the system's path. we can specify the location of LTL2BA using the $--ltl2ba$ command line option. To pass an LTL formula to vereofy via the command line, we use the $--formula = LTL << formula >>$ parameter. This seamless integration allows for effective LTL model checking with vereofy. For example in **Fig.9** and **Fig.10** we formally verify the two LTL properties defined in section 30.

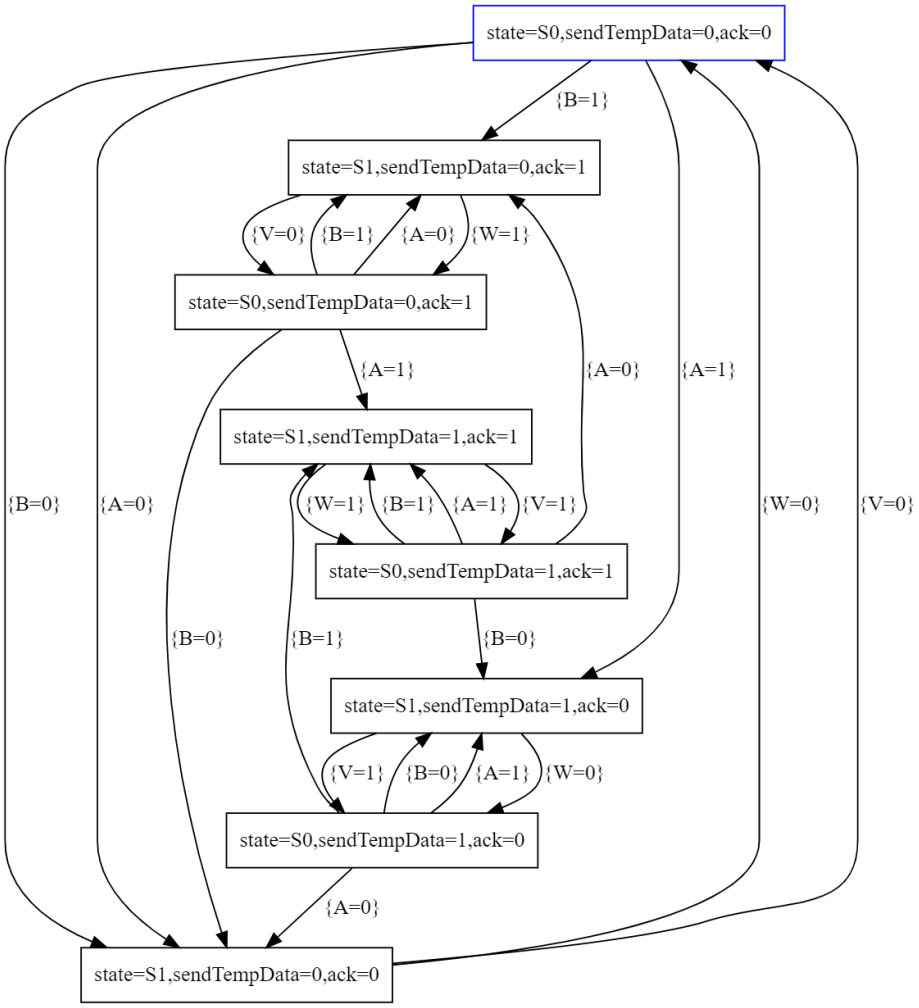


Fig. 8. Binary decision diagram.

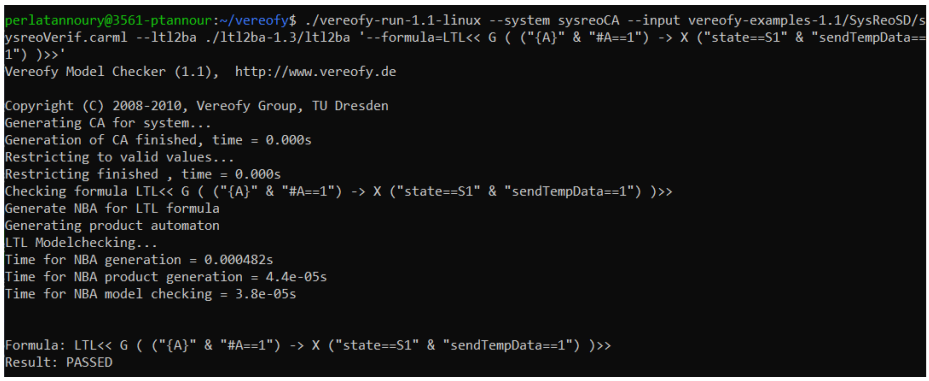


Fig. 9. P1 LTL property.

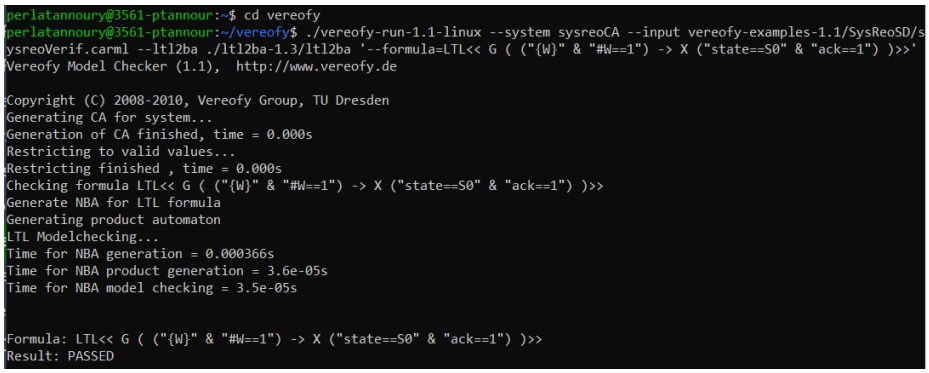


Fig. 10. P2 LTL property.