The Journal of Supercomputing (2024) 80:1206–1237
https://doi.org/10.1007/s11227-023-05483-x

Deploying warehouse robots with confidence: the BRAIN-IoT framework's functional assurance

Abdelhakim Baouya[1] · Salim Chehida[2] · Saddek Bensalem[2] · Levent Gürgen[3] · Richard Nicholson[4] · Miquel Cantero[5] · Mario Diaznava[6] · Enrico Ferrera[7]

Accepted: 6 June 2023 / Published online: 12 July 2023

Abstract
Our study details the development and validation of an orchestrator-controlled robotic network that effectively organizes and manages the activities of multiple robots. The design workflow is based on a model-driven methodology that allows for the independent specification of robot behaviour, which can be successfully refined regardless of the physical architecture. The main focus of this study involves the verification and analysis of robot orchestration by building formal models in a component–port–connector fashion supported by BIP language (behaviour–interaction–priority). The model also helps to study the automated orchestration with the help of a centralized computer tasks manager. The related functional requirements gathered from industrial partners are specified in temporal logic. Statistical model checking is performed to verify the model's correctness, providing a functional assurance to achieve the deployment. Validation is a carry out using a dedicated robotic platform simulator. We demonstrate the capability of the verification artefact for the Brain-IoT (https://cordis.europa.eu/project/id/780089) platform and ways of applying them to potentially complex case studies.

Keywords Internet of Things · Statistical model checking · Robotics · Scalability

1 Introduction

The use of controlled robots in the realm of the Internet of Things (IoT) has received significant research attention. According to the 2020 World Robotics Report published by the International Federation of Robotics (IFR), there are currently 2.7 million robots deployed in factories and warehouses worldwide, with 1.7 million in Asia, 580,000

✉ Abdelhakim Baouya
abdelhakim.baouya@irit.fr ; abdelhakim.baouya@gmail.com

Extended author information available on the last page of the article

Fig. 1 `Robotnik` robots movement in warehouse [17]

in Europe and 389,000 in the Americas [1]. These robots are equipped with actions that enable control [2–4], as demonstrated by companies such as Amazon, Ocado and Exotec in the Economist issue [5]. They are equipped with arms that embed cameras and bar code readers to identify goods, while AI algorithms monitor their actions to manage pickup and item sorting efficiently.

Robots in networked systems can communicate with each other either directly in a decentralized system or through a master controller in a centralized system [6–8]. Decentralized collaborative robots are designed to execute cooperative missions [9–11], and the success of these missions depends on the quality of communication among the robots [12, 13] and their processing speed [14, 15]. However, decentralized robot deployment can be costly in terms of programming as much of the functionality and orders rely on embedded AI algorithms [16]. The use case scenario developed by `Robotnik` is essential to the success of the Brain-IoT project. The company produces robots that are capable of remote control and self-adaptation in warehouse environments (as shown in Fig. 1). To meet the needs of customers, Robotnik controls the robots from a centralized computer while managing their actions and orders. For further information on the robots and physical structures used, please refer to [17].

Building complex systems at a high level of abstraction is an effective method for predicting system behaviour, as most implementation flaws stem from design imperfections, as mentioned by Taylor Armerding [18]. According to Crnkovic and

Larsson [19], component-based design is the ideal paradigm, from a software architect's perspective, for building a system while effectively segregating business logic into individual components. Moreover, this approach offers various ways of composing and coordinating these components. As a result, the incorporation of the component-based design paradigm prompts the usage of model-driven design (MDD) methodologies [20], especially in embedded systems and IoT [21]. MDD provides assistance in the development of model systems by offering abstract characterization support, with a focus on platform-agnostic implementation and execution [22]. The **BRAIN-IoT** framework offers the fundamental elements required to model IoT systems through a series of refinements. Designers first define the platform-independent model (PIM), which is then translated into one or more platform-specific models (PSM) based on the platform definition model (PDM).

The paper builds upon the research conducted in [23] and [24]. So, to guarantee the correctness of Robots Orchestration, certain requirements must be met. For example, if a cart is detected, the robot should lift it off the ground and transport it to the storage area. `Robotnik` provides the requirements that must be satisfied during experimentation within the warehouse. The BIP[1] language, which has been maintained at Verimag Lab for decades [25–27] and has been utilized in successful projects such as CPS4EU[2] and ERGO,[3] is the language we use to construct formal models. To ensure accuracy, the modelling and verification phases will be conducted at the PIM level. The BIP Statistical Model Checker (SMC) [28] enables us to perform both quantitative and qualitative analyses of requirements that have been formalized in temporal logic [29]. Once the requirements are met, a code is generated for the specific execution platform, followed by validation through simulation at the PSM level.

The paper's structure is as follows: Sect. 2 provides a review of the existing literature on the deployment of robot fleets, while Sect. 3 details the methodology and architecture of the **BRAIN-IoT** framework. Section 4 presents a background of the BIP formalism, and Sect. 5 highlights model transformation and the Java code generator. Section 6 models the orchestration in the robots' warehouse, including verification and validation. Finally, Sect. 7 concludes the paper.

2 Related Work

In this section, we delve into works related to the model-driven approach for orchestrating networked collaborative robots, utilizing formal methods and simulation techniques. Formal modelling and verification techniques have primarily been employed in robotic applications such as controllers, motion planning and fault detection [30]. Among the works in this direction, the work presented in [31] proposes a toolchain that utilizes a UML profile for designing models of human–robot collaboration (HRC). Then, a formal model is expressed in terms of metric temporal logic that specifies the concepts defined in the UML model. Transcoding tools are used to

[1] BIP: https://www-verimag.imag.fr/TOOLS/DCS/bip/doc/latest/html/index.html.

[2] CPS4EU: https://cps4eu.eu/.

[3] ERGO: https://www.h2020-ergo.eu/.

automate the development process of the designed task on the chosen infrastructure after requirements satisfaction is enabled by the Zot tool [32]. The toolchain has experimented on some realistic case studies. The approach is also applied for risk analysis [30] in collaborative robotic applications, and Zot formal verification tool is used to identify and mitigate hazardous situations associated with non-negligible risks. Guiochet [33] proposes a hazard identification approach of human–robot interactions. This approach combines the semiformal UML notation with HAZOP (Hazard Operability) to describe robot manufacturers' scenarios using use case, sequence and state machine UML diagrams, and then identify hazards and analyse their risks based on HAZOP tables. It also produces a list of hazards, recommendations and hypotheses. Analysis tools are not detailed in the article but are mainly based on diagram animations using simulation. The authors in [34] utilize the Brahms language for formal modelling of an autonomous assistant robotic system. The Brahms model is then translated into the PROMELA language to verify safety requirements via the SPIN model checker [35]. Dixon et al. [36] apply the NuSMV model checker [37] to analyse the safety and trustworthiness of robot behaviours in a robotic assistant located in a typical domestic environment. This work aims to prove that given temporal properties are satisfied on all the possible behaviours of the system. Mohammed et al. [38] propose a framework for formal specification and verification of multi-robot systems behaviours using Hybrid Finite State Machines. The framework provides two views (concurrent and hierarchical) to optimize the verification and the constructed models. Walter et al.[39] present a methodology for specifying and verifying the functional properties of autonomous vehicles and robots. The method applies the theorem prover Isabelle for interactive formal proof and verification. As all theorem provers, the engine requires multiple assertions to perform verification compared to Model checking. In the study by Murray et al. [40], system properties are established through the co-verification of interconnected models using platform mappings. These mappings establish the relationship between the inputs and outputs of both software and hardware components, as well as their corresponding sensors and actuators. The software components are modelled in RoboChart [41], while the hardware components are modelled in Simulink. To verify the system, the authors utilized Simulink Design Verifier (SDV) [42] and FDR [43]. The system properties were expressed using the formalism of Communicating Sequential Processes (CSP) [44]. The study by Walter et al. [45] presents a custom domain-specific language (DSL) for modelling scenarios involving robot movement and human intervention, which also accounts for the stochastic nature of human fatigue. The model is further refined to a Network of Stochastic Hybrid Automata and analysed using UPPAAL [46] for statistical model checking. The formal model's adherence to reality is thoroughly validated through experimental scenarios created in the field of healthcare service robotics. In their work, Chowdhary and Chattopadhyay [47] propose algorithms for robot orchestration in warehouse and factory-like environments. These improved algorithms incorporate obstacle avoidance and are validated for fault tolerance through various experiments. In their paper, Delgado et al. [48] propose a new approach called OROS, which optimizes the navigation and sensing of robots, as well as the use of infrastructure resources, to minimize the completion times of mission-critical tasks for 5 G-connected robots and for battery life extension.In their paper, Tahir et al. [49] demonstrate how formal verification using

UPPAAL [46] can be applied to an autonomous firefighting robot system. To model the system, they designed a customized arena and a sensor-equipped robot that throws balls into boxes to simulate ejecting water at the fire location. Multiple properties, including safety and liveness, were identified and validated during the simulation to ensure the system met the necessary requirements.

Not only robots are orchestrated but also automatic trains within MDA vision. Baouya et al. [22] proposed model-driven approach (MDA) to model the AATC systems in AADL [50]. The workflow is based on three levels: the platform-independent model (PIM), the platform-dependent model (PDM) and the platform-specific model (PSM), which highlight different refinement levels. Formal verification targets the PIM level since the architecture is abstracted while the simulation is done after deployment at PDM. Moreover, the specification in AADL is partially mapped into PRISM language [51, 52] to perform safety assessment [52–54] due to the limitation of the PRISM model checker [55]. The **BRAIN-IoT** framework [56] encompasses a set of tools from system modelling to code generation. The modelling aspect of the system is performed using BIP [57], a language that offers greater expressivity in component–port–connector formalism with C/C++ constructs. The dedicated statistical model checker SMC-BIP [28] is capable of verifying BIP models through a dedicated engine. Unlike PRISM [55], NuSMV [37] and UPPAAL [46], SMC-BIP compiles models instead of storing them in memory, resulting in reasonable verification times [58]. Furthermore, the code generator is capable of validating specifications through simulation.

3 Modelling and verification artefact of BRAIN-IoT framework

The **BRAIN-IoT** modelling and verification artefact relies on model-driven development (MDD) principles [59]. The system view is structured into three distinct viewpoints: the platform-independent model (PIM), the platform description model (PDM) and the platform-specific model (PSM). The PIM contains the system functionality in component-oriented architecture. The PDM describes the software resources and the hardware platform, whereas the PSM describes the mapping of the software components to the hardware platform. The PSM view is not established until the functionality assurance is met. Figure 2 portrays the **BRAIN-IoT** modelling and verification artefact. The boxes represent the steps, and the edges show their relationship. Statistical model checker [60] accepts the BIP models defined at the PIM level to perform verification ①. Suppose the requirements are satisfied with a certain probability level. In that case, the deployment is performed while Java code is generated with robots communication libraries to perform simulation ② using the tool developed in [61]. Finally, The generated code is refined to the OSGi Bundles. When the requirements are unmet, as mentioned with a red line in Fig. 2, the architect using the **BRAIN-IoT** modelling and verification artefact has to redesign the software view at the PIM level. Documentation pertaining to the Brain-IoT project and its framework

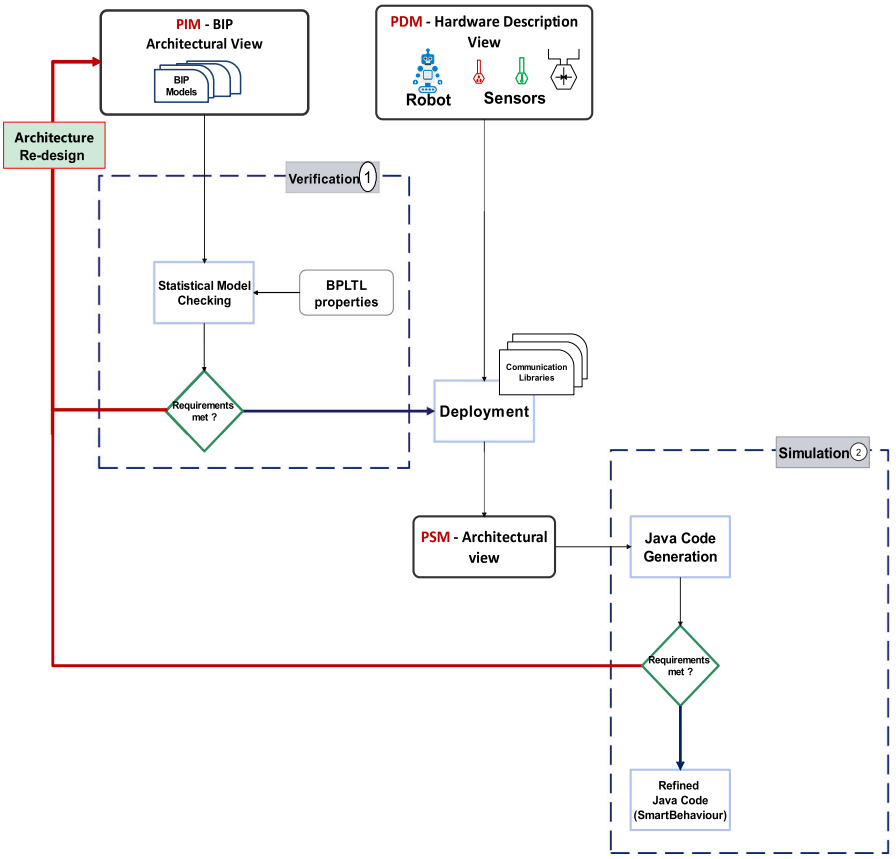

Fig. 2 Functional assurance using modelling and verification artefact of **BRAIN-IoT** framework

can be found on the project's website.[4] Table 1 provides a summary of the main project deliverables that were used in writing this article.

3.1 PIM

The PIM relies on the utilization of BIP[5] constructs that adhere to the component-oriented approach. Two types of components are identified: *atomic* and *compound* components. Components exchange through ports endowed with native (i.e. float, integer and Boolean) and complex data. BIP has been used in multiple projects [62–64], and a formal description related to BIP language and statistical model checker is detailed in Sect. 4.

[4] https://www.brain-iot.eu/resources/public-deliverables/.

[5] BIP: https://www-verimag.imag.fr/TOOLS/DCS/bip/doc/latest/html/index.html.

Table 1 **BRAIN-IoT** stemming deliverables relative to the article

Deliverables reference	Description
http://www.brain-iot.eu/wp-content/uploads/2019/11/D2.4-Updated-Visions-Scenarios-Use-Cases-and-Innovations.pdf D2.4, https://www.brain-iot.eu/wp-content/uploads/2021/05/Brain-IoT_D2.6_Final-Visions-Scenarios-and-Use-Cases-and-Innovations_v1.0_final.pdf D2.6	The documents provide a comprehensive overview of BRAIN-IoT scenarios, including their results and how partners can respond to meet the project requirements.
http://www.brain-iot.eu/wp-content/uploads/2019/11/D2.5-Updated-Architecture-and-Test-Sites-Specifications.pdf D2.5	The document offers insight into a reference architecture that depends on the technology supported by our BRAIN-IoT partners.
https://www.brain-iot.eu/wp-content/uploads/2021/05/Brain-IoT_D4.5_Final-Deployment-and-operation-enablers_v1.0.pdf D4.5, https://www.brain-iot.eu/wp-content/uploads/2021/05/Brain-IoT_D4.4_Final-discovery-search-composition-and-orchestration-enablers_final.pdf D4.4	Robotic Service orchestration task and communication through sensinact gateways and Brain-IoT Service Fabric
https://www.brain-iot.eu/wp-content/uploads/2021/05/Brain-IoT_D6.5_Phase_2_Integration_and_Evaluation_Framework_v1.0.pdf sD6.5	Integration and evaluation of all separated Brain-IoT results of the technical work packages that, once integrated, produce the final framework

3.2 PDM and PSM

Our methodology supports the description of the hardware and software within three views: the hardware platform (PDM), the software platform view and the architectural view.

3.2.1 Hardware platform

This paper describes the hardware platform at a high level of abstraction (Macro architecture) without a specific language to model it. The software platform interacts with the hardware through dedicated services, which are supported by the "sensinact" gateway.[6] The software platform offers IoT service functions such as discovery and look-up services. Within a BRAIN-IoT environment, sensiNact Management service is responsible for the Operational management and monitoring of specific devices via the sensiNact gateway's [65]. Discovery is provided by the generic service discovery mechanisms that return the relevant device based on a set of properties to be matched. The look-up service returns information about one physical entity based on the device ID.

[6] https://projects.eclipse.org/projects/technology.sensinact.

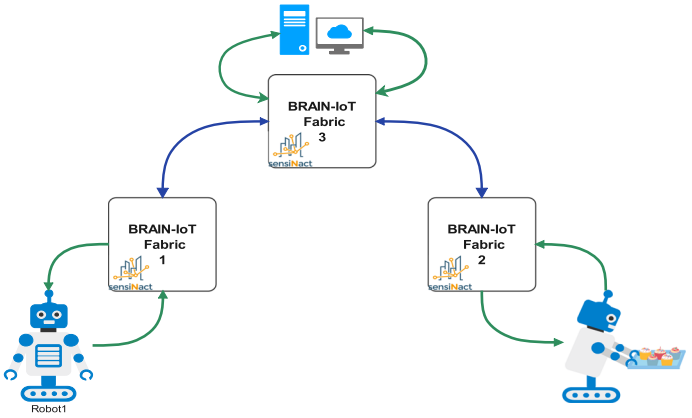

Fig. 3 Federated architecture

3.2.2 Software platform view

The software platform called **BRAIN-IoT** Fabric is based on Paremus Service Fabric[7] which provides a distributed runtime infrastructure for dynamic behaviour expressed in a set of bundles. Each behaviour is a result of Java code generation from the high-level representation in BIP as portrayed in Fig. 2. Each Fabric interacts with one or multiple devices depending on the local or federated architecture as mentioned in Fig. 3. A Fabric is composed of one or more **BRAIN-IoT** nodes, representing bundles. The **BRAIN-IoT** Edge nodes are nodes that provide the connectivity for the software artefacts to communicate with robot devices. The Edge node is played by the Sensinact Gateway [65].

3.2.3 Architectural view

The Architectural view describes:

- The environment into which the **BRAIN-IoT** Fabric will be deployed and
- The dependencies that **BRAIN-IoT** Fabric has on the elements of the environment.

The **BRAIN-IoT** runtime architecture consists of four structural layers and is portrayed in Fig. 4:
Physical layer A set of physical computing resources within the physical environment to be managed (inux Servers and ROS Robots) is selected to run **BRAIN-IoT** nodes. A **BRAIN-IoT** Fabric may be a set of one or more physical resources; the more resources available, the more robust the **BRAIN-IoT** runtime becomes. To participate as a **BRAIN-IoT** node, the compute resource must be capable of running Java and OSGi framework.

[7] https://www.paremus.com/.

Fabric layer To create a Service Fabric, a set of $OSGi^{TM}$/Java agents are installed upon the physical layer. **BRAIN-IoT** nodes are responsible for negotiation, installing/assembling, managing and monitoring software artefacts. The application is written in any Java language.
Systems layer A System is a distributed entity composed of a set of interrelated software components. Meanwhile, it contains the following mandatory **BRAIN-IoT** infrastructure software components: (1) a "Bundle Installer Service"(BIS) responsible for dynamically deploying a specified Smart Behaviour to its local environment, (2) a "Behaviour Management Service" (BMS) responsible for selecting the most appropriate Smart Behaviour from the Marketplace, (3) the "EventBus" substrate that allows asynchronous events to be exchanged between the software components within the same system and (4) the sensiNact node that is responsible for managing communication between federated entities, between smart behaviour and devices.

Smart Behaviours In response to environmental triggers, each **BRAIN-IoT** System is capable of dynamically deploying sets of interrelated Smart Behaviours. **BRAIN-IoT** Smart Behaviours communicate with each other via asynchronous events. A behaviour issues an event which is routed to local or remote endpoints that have registered interest in events of that type: i.e. other Smart Behaviours. If an event cannot be forwarded because no third party has a registered interest, then the Behaviour Management Service consumes the event. The BMS searches the nominated **BRAIN-IoT** repository for an appropriate Smart Behaviour, and if a candidate is found, the BMS instructs a selected Bundle Installer Service (BIS) (local or remote) to instal the selected Smart Behaviour.

4 Background on BIP component formalism

In this section, we provide a background on the modelling and the specification formalism supported by BIP [28, 29, 57, 60].

Atomic components are elementary building blocks for BIP systems. They are described as labelled transition systems extended with *variables*. Transitions between states are labelled by *ports*. A transition is associated with a guard g and an update function $Func(V)$, which are, respectively, a propositional logic formula and a computation defined over local variables V. $eval(V)$ is a function that assigns values to variables V. An atomic component in BIP [28, 29, 57, 60] is formally defined as follows.

Definition 1 (Atomic Components) An atomic component $B{=}\langle S, P, T, s_0\rangle$ is a labelled transition system, where:

- $S = Loc \times Eval(V)$ is a set of states where Loc is a set of component locations and $V = \{v_0, \ldots, v_n\}$ is a set of local variables,
- P a set of communication ports,
- T is a set of transitions of the form (s, p, g, s') where $s, s' \in S$, $p \in P$, $g \in Eval(\vartheta)$ is a guard, and
- $s_0 = \{\langle l_0, X\rangle\} \in S$ is the initial state.

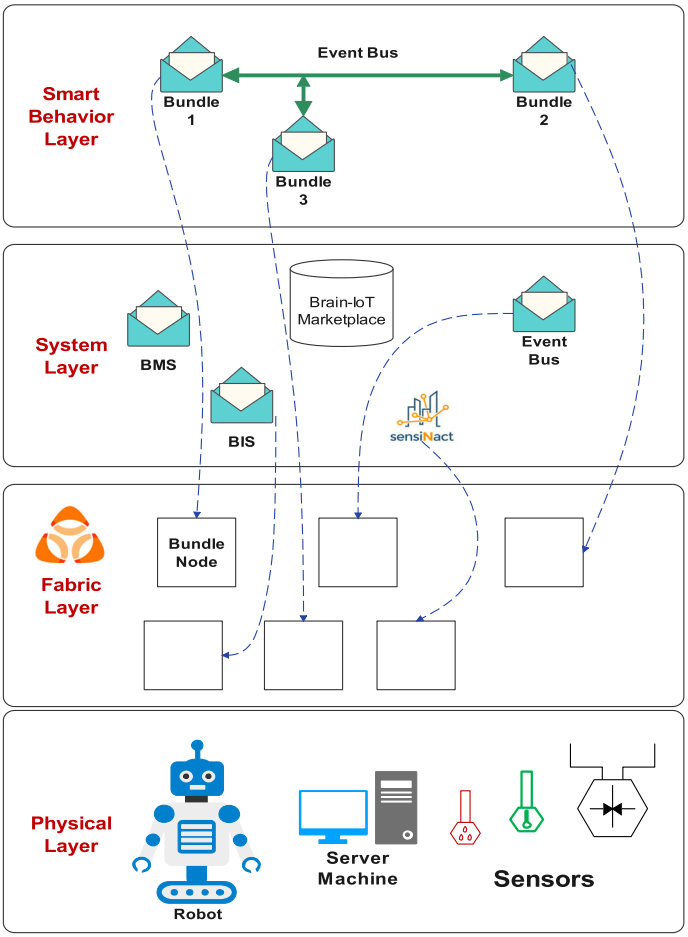

Fig. 4 **BRAIN-IoT** architecture view

Syntactically, A BIP code (i.e. a Program) is composed of a set of "m" components ($m > 0$) where the behaviour of each component is described as a set of statements that take the following form:

$$\texttt{on}\ p\ \texttt{from}\ l\ \texttt{to}\ \mathcal{N}\ \texttt{provided}(g)\ \texttt{do}\{\ v\ =\ eval(v);\ \}$$

The BIP transition can be considered as $l \xrightarrow{g:p} \mathcal{N}$. "p" is the port labelling the transition preceded by the keyword "**on**" and forces components to synchronize and execute actions simultaneously in a lock-step fashion. The current location "l" is preceded by the keyword "**from**", and the next state "$\mathcal{N}$ " is preceded by the keyword "**to**". The transition is enabled when the Boolean expression g evaluates to true within the construct "**provided** ()". Let $\mathbb{D}$ be a finite universal domain. Given

a set of variables ϑ, we define valuations for variables as functions $\vartheta : V \rightarrow \mathbb{D}$ that associate each variable in V with a value in $\mathbb{D}$. To express the evolution of the atomic component, we introduce a token concept as in activity diagrams [66]. So, initially, the initial state is marked using the keyword "**`initial to`**" and formally expressed as follows:

$$\frac{l_0 \rightarrow \mathcal{N}}{\langle l_0, \ldots, \mathcal{N}, \vartheta \rangle \rightarrow \langle \overline{l_0}, \ldots, \mathcal{N}, \vartheta \rangle} \quad (\textit{Initial})$$

where $l_0, \ldots, \mathcal{N} \in Loc$. Also, we express a BIP-triggered transition as an update command:

$$\frac{\overline{l_0} \xrightarrow{g:p} \mathcal{N} \wedge \vartheta \models g}{\langle \overline{l_0}, \ldots, \mathcal{N}, \vartheta \rangle \xrightarrow{p} \langle l_0, \ldots, \overline{\mathcal{N}}, \vartheta' \rangle} \quad (\textit{Update})$$

where $\vartheta' := \vartheta[v_i := eval(v_i)$ and $l_0, \ldots, \mathcal{N} \in Loc$

Within the BIP atomic component, for a given valuation of variables, a transition can be executed if and only if its associated guard evaluates to $true$. Moreover, to deal with the system's interoperability, BIP formalism provides mechanisms for harmonizing and coordinating components' behaviours, namely priorities and connectors [28, 67]. The result of the interaction is a composition of synchronized components obtained by using the component composition operator γ presented in Definition 2.

Definition 2 (*Composition*) The composition of two atomic components denoted by $\gamma(B_1, B_2)$ is a composite component $B = \langle S, P, T, s_0 \rangle$, where:

- $s_1 \xrightarrow{p,g_1} s_1' \wedge s_2 \xrightarrow{p,g_2} s_2'$ such that $s_1, s_1' \in S_1,\ s_2, s_2' \in S_2$ where $p \in P_1 \cap P_2 \wedge X_1 \models g_1 \wedge X_2 \models g_2$,

The SMC[8] implements the main statistical model checking techniques, namely hypothesis testing [68] and probability estimation [69]. Queries/requirements to be verified using SMC-BIP [28] shall be expressed in PBLTL (Probabilistic Bounded Linear Temporal Logic). The syntax of the PBLTL temporal logic is detailed in [28, 29, 57, 60]. Using this query language, it is possible to formulate probabilistic queries in this format:

- Qualitative queries: $\mathtt{P}_{\geq\theta}[\varphi]$, where $\theta \in [0, 1]$.
- Quantitative queries: $\mathtt{P}_{=?}[\varphi]$, where φ is a bounded LTL formula.

Below are two illustrative examples with their natural language translation.

- $\mathtt{P}_{\geq 0.68}[\mathtt{fail\ U}^{\leq 1000}\mathtt{reboot}]$ "*The probability of the system eventually reboots after failure is at least 0.68*". The path formula $\mathtt{F}^{\leq 1000}$ specifies that the length of the considered traces is 1000.

[8] http://www-verimag.imag.fr/BIP-SMC-A-Statistical-Model-Checking.html?lang=en.

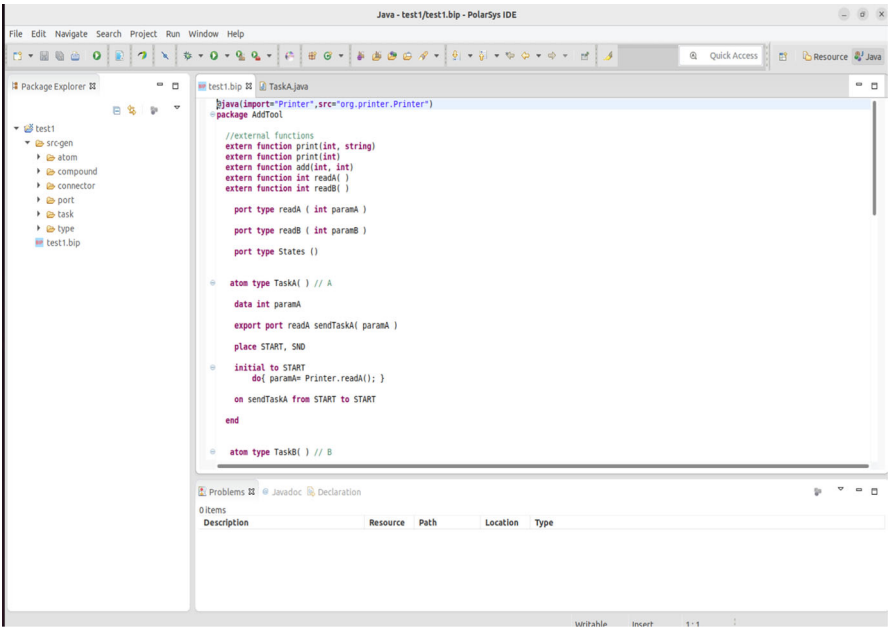

Fig. 5 BIP grammar integration in eclipse tool (supported by the eclipse foundation). The figure shows the BIP project explorer on the left and the BIP editor on the right

- $\mathtt{P}_{=?}[\mathtt{F}^{\leq 1000}\,\mathtt{shutdown}]$ "*What is the probability that the system eventually shutdown?*".

BIP has been implemented in a distributed setting [70, 71], and the components exchange information through asynchronous messaging. This means that each component is capable of sending messages, waiting for notifications or performing internal computations based on harmonizing and coordinating protocols. However, it is important to note that this representation only shows a high-level view and does not reflect the actual software platform structure where the processor manages task scheduling.

5 Model transformation

In this section, we showcase the conventional model transformation process that utilizes Eclipse plug-ins to automatically generate the specific code needed for robot orchestration.

5.1 BIP models to Java code by example

Before integrating the event bus component, the model is fed into the code generator engine. The engine utilizes Xtend to capture the modelled atomic components, made

possible through the Xtext module that enables grammar specification and keyword colouring. The result of such integration is portrayed in Fig. 5. Each atomic component is converted to Java class that extends Java thread and annotates with BIP ports. The example of the atomic component in Listing 1 is mapped to Java code in Listing 2. The initial state `START` and the next state `LOADING` characterize the atomic component. The command in line 9 is enabled if the `START` location is marked and the corresponding condition in the provided construct is satisfied.

Listing 1 Example of BIP Component

```
1  atom type robot()
2   export port Port p
3   data int positionX
4   data int positionY
5   place START, LOADING
6   ...
7   initial to START
8   ...
9   on p from START to LOADING  provided ( positionX
        =1 and positionY = 6) do { initRobot();}
10 end
```

A set of attributes and operations characterizes a Java class. These attributes are related to the data manipulated by atomic components and the locations identified by the "place" keyword. The data types of atomic components are mapped to the respective Java data types, while BIP locations are mapped to Boolean variables. For example, in Listing 2 lines 13–14, the activation of the port *p* means that the function preceded by the annotation "@p()" is called. If the location "`START`" is evaluated to true (line 15), then the guard is checked (line 16), while the *do* actions correspond to Java instructions. Functions that are not preceded by Java annotations are called by *do* actions.

Listing 2 Example of Generated code

```
1  public class robot extends Atom{
2  /*  Variable declaration    */
3      private int positionX
4      private int poisitinY
5  /*  State declaration    */
6      private boolean START
7      private boolean LOADING
8  /*  Labelled transitions declaration    */
9      @initial()
10     public void initialFunction(){
11         START = true;
12     }
13     @p()
14     public void portP(){
15         if(START){
16             if (positionX=1 && positionY = 6){
17             LOADING=true;
18             START=false;
19             initRobot();
20             }
21         }
22     }
23     private void initRobot(){
24     ...
25     }
26 }
```

BIP connectors are managed by a connector scheduler that facilitates communication between Java classes through scheduled send–receive operations. However, it is important to note that our generator only handles a subset of the communication styles supported by BIP Connectors [67].

5.2 Deployment of Java code at EventBus level

The connectors scheduler does not handle communication between generated components; instead, this task is the responsibility of the PAREMUS Event Bus. The bus is responsible for both data communication and eventing. Ports defined in Listing 2 are mapped to Java interfaces. Atomic components are now considered as Java Bundles or "SmartBehaviour", where an additional Java class is created to handle event listening (as shown in Listing 3). The required interface (i.e. BIP Port) is specified as a consumed interface within the "@SmartBehaviourDefinition" annotation (line 2). The "notify" function is sensitive to the event; for example, in line 8, the "RobotImpl" class is lis-

tening for events of type “IresolveCollision” (line 8) and performs the corresponding actions.

Listing 3 Example of SmartBehaviour Definition

```
1  ...
2  @SmartBehaviourDefinition(consumed = {IresolveCollision .class},
3  author = "UGA", name = "Smart RobotA",
4  description = "Implements a remote Smart RobotA.")
5  public class RobotImpl implements SmartBehaviour<BrainIoTEvent>{
6      public void notify(BrainIoTEvent event) {
7          public static IresolveCollision resolveCollision ;
8          if (event instanceof IresolveCollision) {
9              resolveCollision = (IresolveCollision) event;
10             synchronized (this) {
11                 resolveCollision.resolveCollision();
12             }
13         }
14  }
15 }
```

6 From modelling to simulation of Robots Orchestration system

This section describes the scenario provided by Robotnik,[9] followed by the BIP model and architecture related to the case study for analysis.

6.1 Robots Orchestration scenario

A privately owned warehouse houses thousands of carts filled with two or three products, which are transported by Robotnik-manufactured robots programmed to travel in all four cardinal directions to reach their respective destinations (See Fig. 6, 7). Upon reaching the designated cart, the robots (see Fig. 8) execute a corkscrew motion to lift the unit from the ground, transporting it in its entirety to the storage area ② as depicted in Fig. 6 where humans pack the appropriate items. After completing the delivery, the robot proceeds to the unload area ③ and locates a new cart amidst the densely packed shelves. A fog-based brain controller is responsible for coordinating the robot’s movements across the 2D surface (see Fig. 6 and Fig. 7).

Every 0.5 s, the controller sends *position* requests, and the robots (see Fig. 8) will send back a structured response in JSON. The JSON scripts related to robot interactions are available in [72]. The robots read their positions from the QR code tag placed on the square grid (see Fig. 6). Also, the response contains the actual robot state, such as

[9] Robotnik is a company specialized in robot product development and commercialization (mobile robots, robot arms, robotic hands and humanoids).

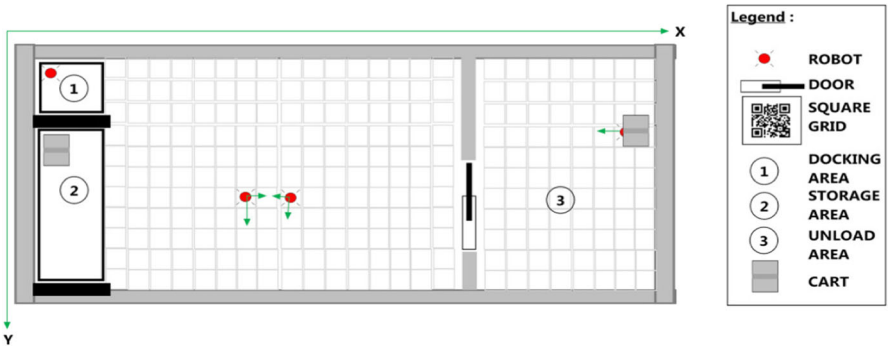

Fig. 6 2D map surface of the warehouse

"stopped" or "running". The robots are endowed with motion sensors in the front to switch to the "stopped" state if obstacles are detected, so the robot response includes an integer attribute "detect" taking values from 0 to 2. If it is 0, then no obstacle is detected else; the obstacle could be a door or robot. When the robot detects a door, the orchestrator will send a request to the automatic door to open, and then, the robot can enter the unload area. In case of collision shall be avoided, the controller will execute a "collision resolve" that orders the robots to update their positions. After retrieving the necessary carts and depositing them in the storage area, the robots return to their original positions in the docking area.

The company[10] responsible for deploying this system would ensure that loading and unloading processes are functioning correctly while also ensuring that collision avoidance measures are properly executed. The central deployed orchestrator must fulfil two requirements:

- *REQ-1* If a cart with densely filled shelves is detected, the robot performs a corkscrew motion to lift the cart off the ground and transport the entire unit to the storage area.
- *REQ-2* If a robot in front is detected, collision avoidance measures shall be taken to ensure safe navigation towards the robot's destination.

6.2 Robots Orchestration model

To demonstrate the practical application of the BIP framework, we have precisely developed BIP models for Robots Orchestration by capturing the control flow of the scenario depicted in Sect. 6.1. First, it starts with a definition of functions that retrieve the position of robots using the reserved word **extern**, e.g.:

```
extern int getPosition ()
```

Some of the variables are used to check the position of the robot according to the grid map in Fig. 6 using the reserved word **const data** as follows:

[10] Robotnik: https://robotnik.eu/.

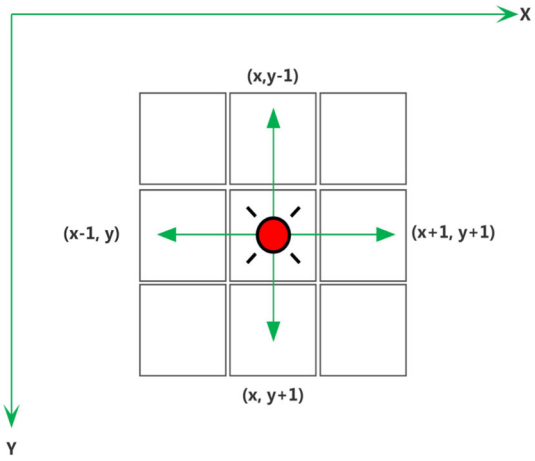

Fig. 7 Robots movement direction

Fig. 8 Robot prototype

```
const data int inDocking = 1
const data int inUnload  = 2
const data int inStorage = 3
```

Other variables are used to check the nature of obstacles in front of the robots declared also as constant (Figs. 7 and 8):

```
const data int isObstacle = 4
const data int isDoor = 5
```

Each transition is labelled with a port that needs to be declared in BIP. Two kinds of ports are declared in the model: "`export port`" and "`internal port`" as mentioned in Sect. 4. The "`export port`" is used to trigger transitions while synchronizing with external atomic component, whereas "`internal port`" are used to trigger internal transitions. So, we have to declare three port type using the reserved word **`port type`**:

```
port type Port_Type_collision (float position)
port type            Port_Type_collision_Three(float
position1,float position2,float position3)
port type Port_Type_No_Param ( )
```

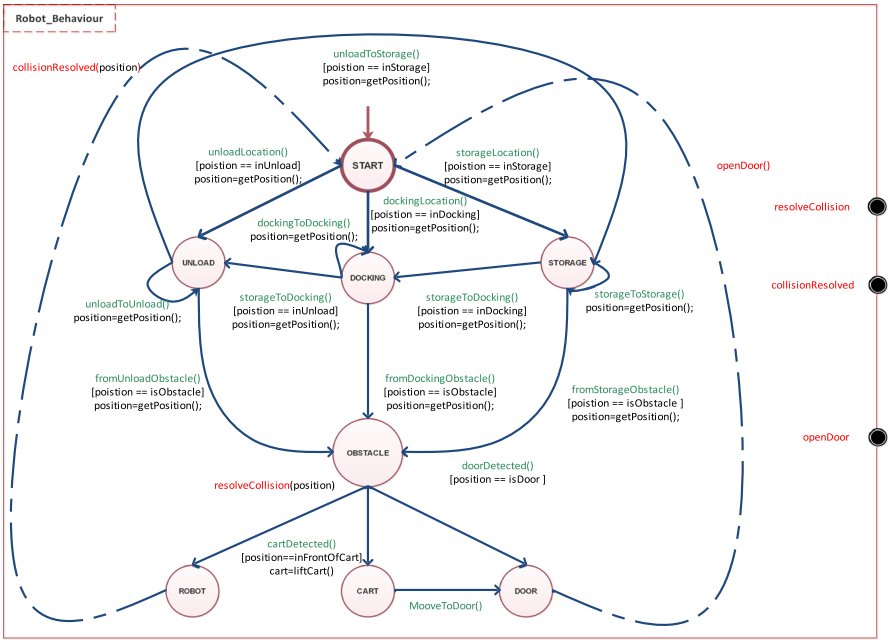

Fig. 9 Graphical BIP representation of individual robot behaviour

The first port type is utilized for receiving and transmitting robot parameters from/to the orchestrator, which enables the resolution of collisions between robots. As mentioned in Fig. 9, the port is instantiated by the atom "`Robot_Behavior`" in "**`resolveCollision`**" and "**`collisionResolved`**". The port instance "**`openDoor`**" referred to the third port definition while no parameters are sent. These kinds of ports are called *silent*. These ports are used to label transitions between different BIP states. In BIP, states are preceded by the reserved word **`place`**:

```
place      START, UNLOAD, DOCKING, STORAGE, OBSTACLE,
ROBOT, DOOR
```

The BIP model shown in Fig. 9, representing a robot, is suitable for all robots. Other robots (i.e. atomic components) are reusing the same model by *instantiating* it with a different name as in Listing 4 lines (4–6). Hence, the robots are instantiated with the names "`Robot1`", "`Robot2`" and "`Robot3`". The same manner will be applied for connectors as in lines 9–11. Connectors have three port parameters since they handle the communication between robots and the orchestrator. Further, some atoms ports are exported (i.e. used for synchronization) in the model of Fig. 9. They are identified on the frame edge with red colour, for instance, "**`resolveCollision`**" and "**`collisionResolved`**", and "**`openDoor`**" ports.

Listing 4 Componenents Instanciation

```
1  compound type Compound ()
2
3  component Door_behavior Door()
4  component Robot_behavior Robot1 (1)
5  component Robot_behavior Robot2 (2)
6  component Robot_behavior Robot3 (3)
7  component Orchestrator_behavior Orchestrator1 ()
8
9  connector connector_type_door Connector1 (
       DoorInstance.openDoor, Robot1.openDoor,
       Robot3.openDoor, Robot2.openDoor)
10 connector connector_resolve_collision Connector2
       (Robot1.resolveCollision, Robot2.
       resolveCollision, Robot3.resolveCollision,
       Orchestrator1.resolveCollision)
11 connector connector_collision_resolved Connector3
        (Robot1.collisionResolved, Robot2.
       collisionResolved, Robot3.collisionResolved,
       Orchestrator1.collisionResolved)
12 end
```

When the robot model is triggered, as portrayed in Listing 5 local variables (i.e. position) are initialized for the first execution. Also, the "id" of the robot in Listing 4 (line 4, id=1) is initialized with the parameter value "VID" of the component as in Listing 5 (line 2). Moreover, the robot retrieves its actual position by calling the function "`getPosition()`" that is declared above.

Listing 5 Train Variables Initialization

```
1  initial to START do {
2  id=VID;
3  position=getPosition();
4  }
```

The BIP model in Fig. 9 relies on multiple phases that are labelling the model transitions, for instance; the port "**`unloadLocation`**" labels `START`→ `UNLOAD`, "**`unloadToUnload`**" labels `UNLOAD`→ `UNLOAD`, "**`unloadToStorage`**" labels `UNLOAD`→ `STORAGE`, "**`fromUnloadObstacle`**" labels `UNLOAD`→ `OBSTACLE`. In case the door is detected while guard "`position==isDoor`" a transition is enabled `OBSTACLE` → `DOOR` labelled with port instance "**`doorDetected`**", else, multiple robots are standing in the same position while they activate the "**`resolveCollision`**" for `OBSTACLE`→`ROBOT`. When the collision is resolved, a transition occurs on `ROBOT`→ `START` labelled with "**`collisionResolved`**".

The red ports identified in Fig. 9 are synchronized with those portrayed in Fig. 10. Two atomic components are modelled: the left one (i.e. Door behaviour) model is the door opening, whereas the right model is the orchestrator that is identified with tree states "`START`", "`COLLISION`" and "`RESOLVED`". The collision is resolved by calling external functions "`resolvedPosition1(position1)`",

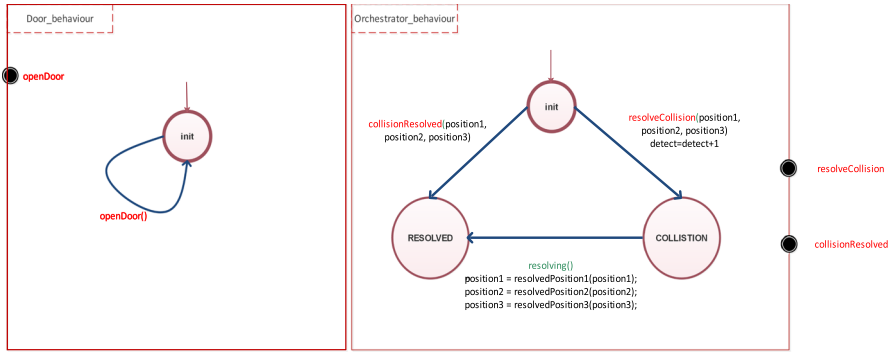

Fig. 10 Graphical BIP representation of door behaviour (left) and orchestrator behaviour (right)

"`resolvedPosition2(position2)`" and "`resolvedPosition3(position3)`". These functions rely on Artificial Intelligence algorithms that return the new position of robots as to move forward/backward or moving left/right.

Three connectors are identified to ensure communication between atomic components, as mentioned in Listing 4. The first connector in line 9 allows the opening of the door following a *Broadcast* manner described in Sect. 4 where the door is triggered when one of the robots is ready on the transition `DOOR→ START` as portrayed in the chronogram of Fig. 11. The connection is handled cyclically, prioritizing the first available port. Listing 6 portrays a textual representation of the *Broadcast* connector to open the door. As the synchronization solely requires a basic notification to trigger the operation, no behaviour is impacted

Listing 6 Openning Door Synchronization

```
1  connector type connector_type_door (Port_Silent p1,
       Port_Silent p2, Port_Silent p3, Port_Silent p4)
2  define p1 p2' p3' p4'
3
4  on p1 p2
5  on p1 p3
6  on p1 p4
7
8  end
```

In addition, robots simultaneously communicate their positions to the orchestrator to perform collision detection and avoidance. In this case, the robots send their positions at the same time to the orchestrator, which is handled by the *Rendez-Vous* connector. Transitions labelled with port "**`resolveCollision`**" in the model do not occur until the Rendez-Vous is satisfied, portrayed in the chronogram of Fig. 12. Listing 7 portrays a textual representation of the *Rendez-Vous* connector. During the synchronization, robots communicate their current positions to the orchestrator in lines (4–7).

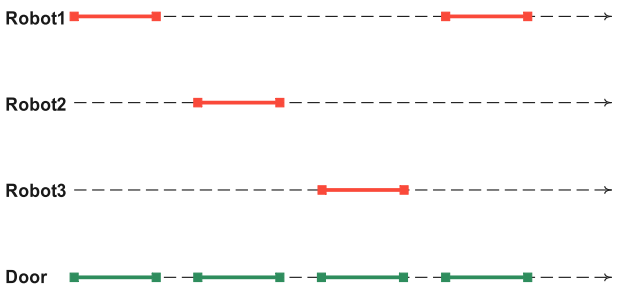

Fig. 11 Broadcast synchronization for connector of Listing 6

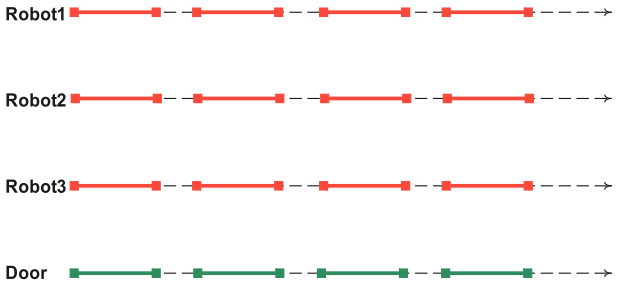

Fig. 12 Rendez-Vous synchronization for connector of Listing 7

Listing 7 Robots Sending their Positions to the Orchestrator

```
1  connector type connector_resolve_collision (
       Port_Robot_Position p1, Port_Robot_Position p2,
       Port_Robot_Position p3, Port_Orchestrator p4)
2  define (p1  p2 p3  p4 )
3
4  on p1 p2 p3  p4 down {
5                          p4.position1 = p1.position;
6                          p4.position2 = p2.position;
7                          p4.position3 = p3.position;}
8  end
```

Figure 13 depicts the global graphical architecture of the complete Robots Orchestration system. It is a graphical interpretation of the textual representation in Listing 4. The blue line links model the connectors, the red boxes model components, and the black circles model the ports.

6.3 Verification and analysis of compliance with requirements

Utilizing the resulting BIP models, we rely on `SMC-BIP` to conduct statistical analysis. SMC-BIP [28] generates runtime traces required to verify probabilistic bounded LTL properties. One of the distinguishing features of `SMC-BIP` is that it can determine the probability of a specified PBLTL property holding based on the generated

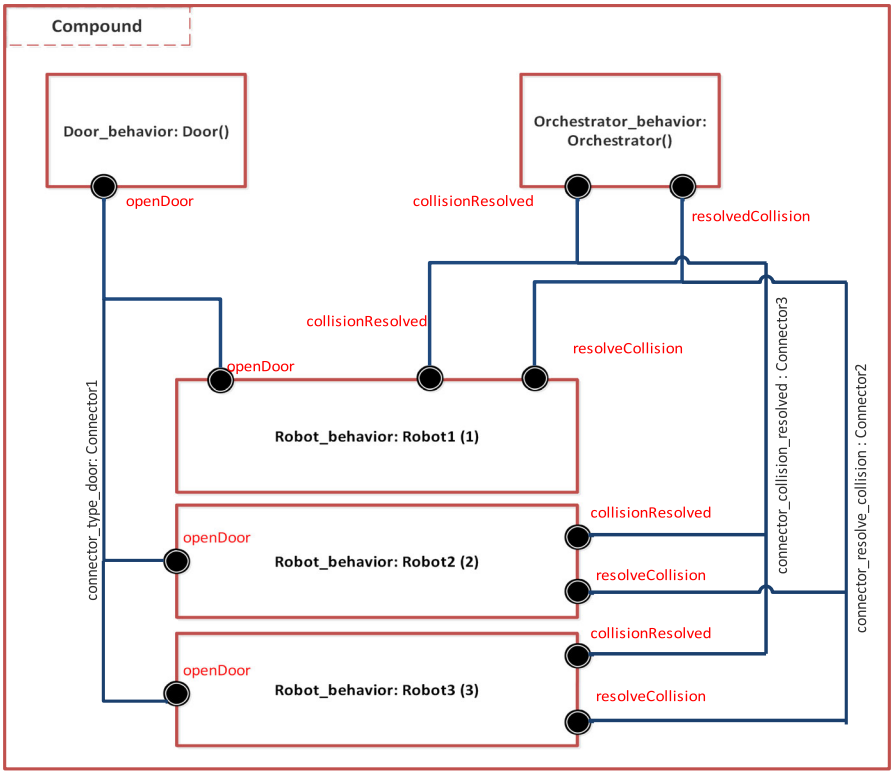

Fig. 13 Graphical BIP representation of Robots Orchestration system

traces. Regarding the requirement expressed in the Robots Orchestration scenario, we formalize it in PBLTL as follows:

$$\varphi_1 : \; P_{=?}[(\texttt{R1.position} == \texttt{inUnload}) \cup^{\leq t} (\texttt{R1.position} == \texttt{inStorage})], \; t = 1000 \qquad (1)$$

The property φ_1 expresses that the robot `R1` is in the `UNLOAD` position for lifting the cart and then returned `STORAGE` position. The resulting probability is equal to 80%.

Moreover, we would check the total carts that have been collected by three robots (in our case we model a system with three carts). In this case, we define a new variable "`cart`" that is initialized to 3. The function "`liftCart()`" sends actions to the robots to perform the operation and returns the remaining cart. Thus, the value of the cart is updated through the transition statement `OBSTACLE`→ `CART` as follows:

```
on cartDetected provided
(position==inFrontOfCart) from OBSTACLE
to CART do {cart=liftCart();}
```

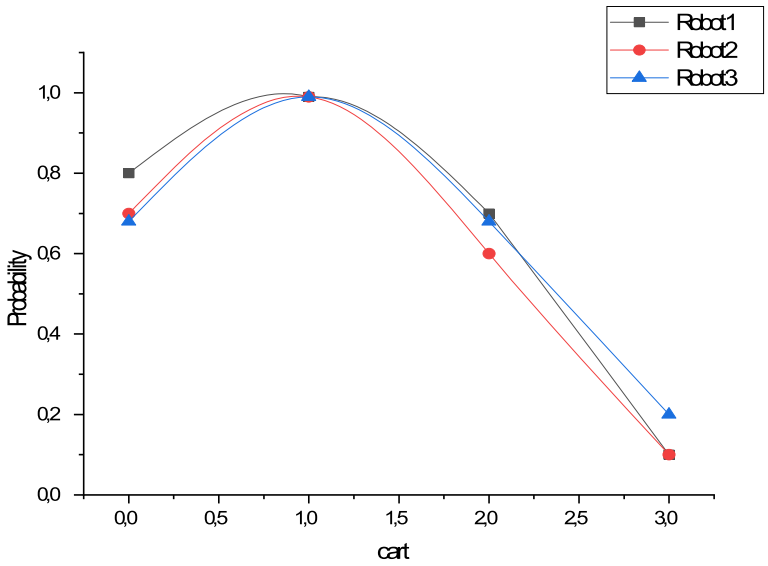

Fig. 14 Checking results of properties φ_2, φ_3 and φ_4

$$\varphi_2 : \; P_{=?}[(\texttt{R1.position} == \texttt{inDocking}) \; \cup^{\leq t} (\texttt{R1.position} == \texttt{inUnload} \;\&\&\; \texttt{R1.cart} == x)], \; t = 1000, \; x = 0:3:1 \tag{2}$$

$$\varphi_3 : \; P_{=?}[(\texttt{R2.position} == \texttt{inDocking}) \; \cup^{\leq t} (\texttt{R2.position} == \texttt{inUnload} \;\&\&\; \texttt{R2.cart} == x)], \; t = 1000, \; x = 0:3:1 \tag{3}$$

$$\varphi_4 : \; P_{=?}[(\texttt{R3.position} == \texttt{inDocking}) \; \cup^{\leq t} (\texttt{R3.position} == \texttt{inUnload} \;\&\&\; \texttt{R3.cart} == x)], \; t = 1000, \; x = 0:3:1 \tag{4}$$

The properties φ_2, φ_3 and φ_4 express that when the robots R1, R2 and R3 are at the docking position, they move to unload position after visiting the storage area. The robot will lift the cart and place it in the unload area. Also, the properties evaluate the number of carts collected by the robots. Checking φ_2 using SMC-BIP results in the graphs portrayed in Fig. 14. The findings indicate that the probability of robots collecting a single cart is high, at approximately 90%. However, there remains a possibility that the robots may not collect any carts. Therefore, the likelihood of robots collecting two carts consecutively is nearly 80%. Due to high levels of concurrency among the robots in the warehouse, collecting three carts successively has a relatively low probability ranging between 10% and 20%.

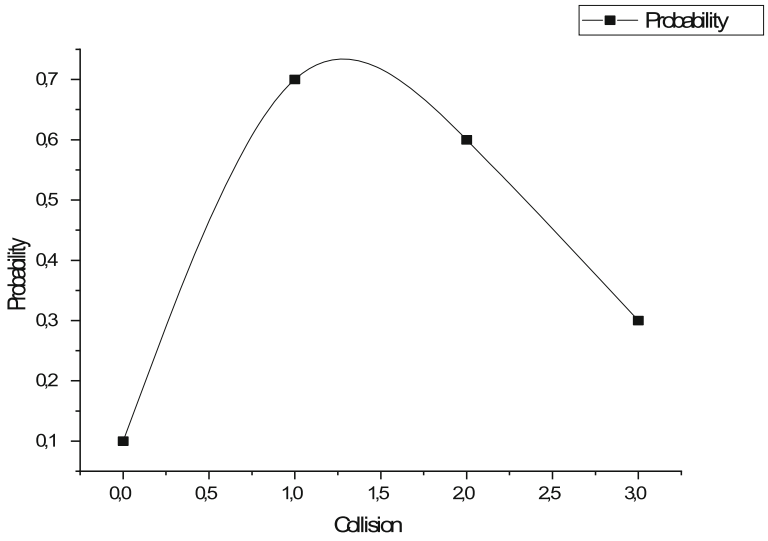

Fig. 15 Checking results of property φ_5

$$\varphi_5 : \mathrm{P}_{=?}[(\mathtt{R1.position} == \mathtt{inDocking} \;\&\&\; \mathtt{R2.position} == \mathtt{inDocking} \;\&\&\; \mathtt{R3.position} == \mathtt{inDocking}) \cup^{\leq t} (\mathtt{R1.cart} \leq \mathtt{C} \;\&\&\; \mathtt{R2.cart} \leq \mathtt{C} \;\&\&\; \mathtt{R2.cart} \leq \mathtt{C} \;\&\&\; \mathtt{O.detect} == \mathtt{x})], \quad \mathtt{t} = 1000,\ \mathtt{x} = 0:10:1,\ \mathtt{C} = 3 \tag{5}$$

Moreover, we want to check how the orchestrator resolved the collision during the robot's movement. The transition START→ COLLISION in the automata model of the orchestrator has for actions to increment the number of detected collisions. The property φ_5 expresses that when the robots are in the docking area, then they collect a set of carts "C" with a certain number of collisions. The symbol "O" refers to the orchestrator component instance. The result of checking property φ_5 is portrayed in Fig. 15. The likelihood of three collisions occurring during the movement of robots is low, as compared to one or two collisions. This trend can be attributed to the fact that when the robots collect carts, they return to their docking position and consequently reduce the number of robots in various warehouse areas, thereby minimizing potential collisions.

6.4 Validation at PSM level

While the **BRAIN-IoT** PIM level is based on statistical model checking, which enables estimation while satisfying requirements, behaviour may differ significantly at the PSM level. Components such as robots are not linked using connectors; instead, they communicate through dedicated libraries that facilitate send/receive operations

with physical units. Connectors can be represented by buses that transport data using send/receive protocols. Furthermore, the quality of data transportation is determined by parameters such as the number of bus access conflicts and bus delays, which are not visible at the PIM level. The paper does not cover the verification procedure at the transaction level.

After verifying that all requirements were satisfied, we performed a model-to-text operation to generate Java code corresponding to the BIP models detailed in this paper. The code generated from this operation is available in [61]. Each atomic component has been translated into a Java class capable of handling all necessary operations for robot movements. Finally, we conducted a simulation using the generated Java code from the initial BIP models. Fortunately, while moving from docking to unloading areas, robots are able to collect available carts for transportation. This confirms that the deployed infrastructure aligns with PIM BIP models. Moreover, we observed that it is impossible for robots to collect two carts due to how communication APIs handle requests during conflict resolution.

Each class is wrapped within OSGi bundles that are accepted by the **BRAIN-IoT** Service Fabric. The OSGi bundles are deployed over a cluster with `Ubuntu−16.04` desktop Intel core i7-950@3.07 GHz and ROS Kinetic with `STAGE` [73] and `rviz` GUI [74].

Also, we use sensinact controllers [65] that implement the mechanic to communicate with the simulation platform called ROS-REST API. We use rviz to plan the intelligent robot's movement within a 3D movement area and STAGE to capture a robot's movement into 2D plan. This simulation is done to validate the requirement REQ-2. The sequence of robot movements is portrayed in Figs. 16, 17 and 18. Figure 16 presents the initial state of the robots in the docking area. Figure 17 presents the state of the blue robot in front of the door, so the door is not visible in the figure. Figure 18 portrays the carts and robots in the unloading area. Through the simulation, no collision is observed due to well orchestration management and competitive access to the unloading area. In addition, we could observe that the door reacts to the robot's demand to open it. These observations help the designer make judgments about the accuracy of the modelled system. Also, these observations validated the requirements expressed formally in LTL.

7 Conclusion

This paper presents the functional assurance artefact of the **BRAIN-IoT** framework, which relates to the orchestration of a fleet of robots within warehouses by Robotnik System Company. The scenario at hand involves utilizing a central computer that is equipped with AI algorithms to detect and manage collisions. Our proposed approach to accomplish this task is through the utilization of a model-driven design (MDD) methodology, which establishes several refinement levels.

This research specifically focuses on the design level of the orchestrator-controlled robotic network, which is developed using the BIP language. To ensure the accuracy of the design, we utilize SMC-BIP to verify a set of properties expressed in PBLTL at the PIM level. By utilizing mathematical reasoning, this formal verification tech-

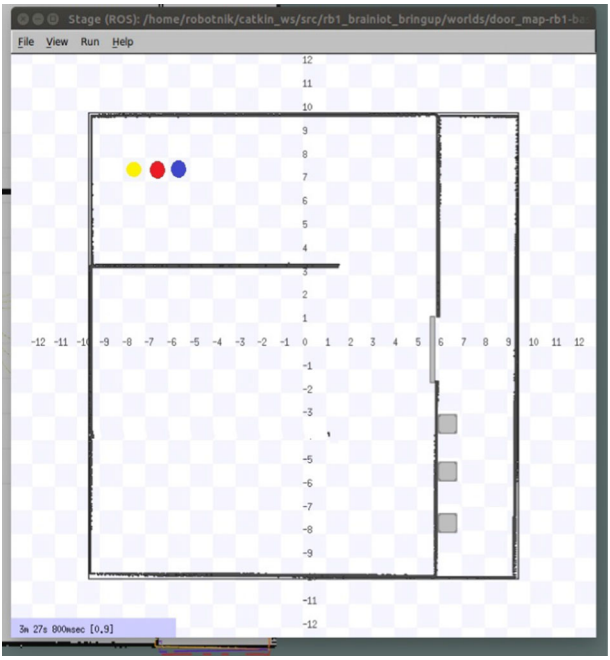

Fig. 16 Robots in docking area

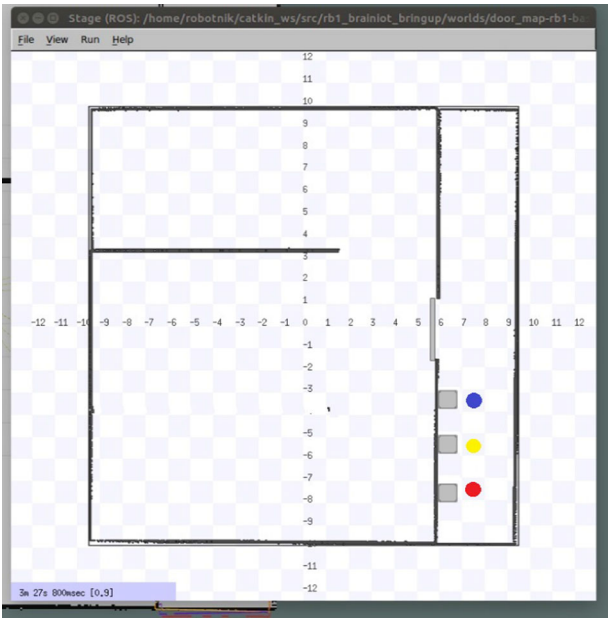

Fig. 17 Robots in front of the door

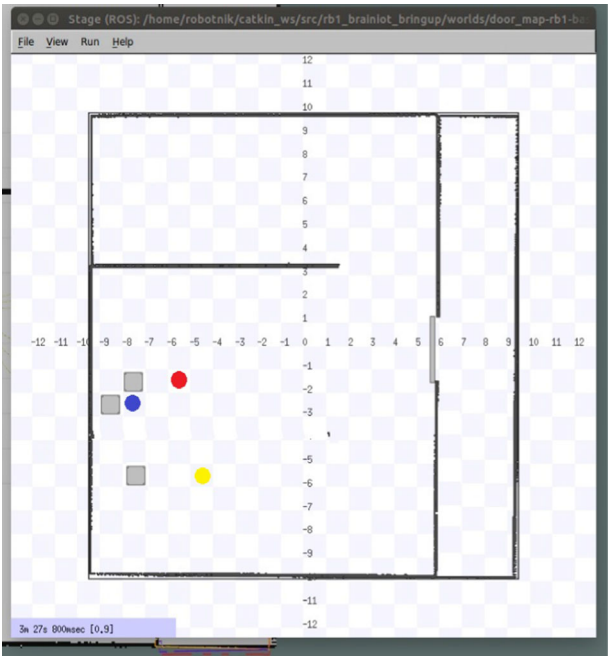

Fig. 18 Robots in storage area

nique provides designers with functional assurance for robotic scenarios. A Java code is generated through model-to-text transformation, enriched with Brain-IoT Fabric annotations and communication libraries, to facilitate the deployment of IoT nodes.

In addition to the high-level analysis, we validate our findings through simulation, which takes into account the features of both the robot's communication services and the **BRAIN-IoT** Service Fabric. In our future work, we will conduct verification at the transaction level by accurately modelling communicating buses. We will also focus on validating a second use case for the water dam system that regulates drained water in Corona, Spain.

Author Contributions AB wrote the main manuscript. SC, SB, RN, LG, MC, MD and EF collaborated in both writing and reviewing the manuscript. SB, RN, LG, MC, MD and EF provided technical and financial support.

Funding The research that led to the presented results was conducted within the research profile of Brain-IoT- model-**B**ased f**R**amework for dependable sensing and **A**ctuation in **IN**telligent decentralized **IoT** systems, funded by the European Union, Grant No.: 780089. The authors wish to extend their gratitude towards the Eclipse Foundation, and specifically Philippe Krief and Maria Teresa Delgado, for providing technical support throughout this project. Additionally, the authors would like to thank them for hosting the Eclipse plug-ins that were developed during this project.

Availability of data and materials The plug-ins used in the experiments are linked to the paper describing those experiments. Models are available upon request.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent Informed consent was obtained from all individual participants included in the study.

References

1. International Federation of Robotics (2020) Ifr annual report. https://ifr.org/downloads/press2018/Presentation_WR_2020.pdf
2. Dourado Carlos MJM et al (2019) A new approach for mobile robot localization based on an online IoT system. Future Gener Comput Syst 100:859–881
3. Dingju Z (2018) IoT and big data based cooperative logistical delivery scheduling method and cloud robot system. Future Gener Comput Syst 86:709–715. https://doi.org/10.1016/j.future.2018.04.081
4. Coquin D, Boukezzoula R, Benoit A, Long NT (2020) Assistance via IoT networking cameras and evidence theory for 3d object instance recognition: application for the nao humanoid robot. Internet Things 9:100128. https://doi.org/10.1016/j.iot.2019.100128
5. The Economist (2014) The bots in the warehouse, new robots–smarter and faster–are taking over warehouses. https://www.economist.com/science-and-technology/a-new-generation-of-smarter-and-faster-robots-are-taking-over-distribution-centres/21807595
6. Rameez C, Manju C (2021) Orchestration of automated guided mobile robots for transportation task in a warehouse like environment. 5:1–7. https://doi.org/10.1109/ETI4.051663.2021.9619354
7. Mello Ricardo C et al (2022) The poundcloud framework for ROS-based cloud robotics: case studies on autonomous navigation and human-robot interaction. Robot Auton Syst 150:103981. https://doi.org/10.1016/j.robot.2021.103981
8. Hiejima T, Kawashima S, Ke M, Kawahara T (2019) Effectiveness of synchronization and cooperative behavior of multiple robots based on swarm AI. In: 2019 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS). pp 341–344. https://doi.org/10.1109/APCCAS47518.2019.8953108
9. Khamis A, ElGindy A (2012) Minefield mapping using cooperative multirobot systems. J Robot 698046:1687. https://doi.org/10.1155/2012/698046
10. Michael N, Zavlanos MM, Kumar V, Pappas GJ (2008) Distributed multi-robot task assignment and formation control. In: 2008 IEEE International Conference on Robotics and Automation. pp 128–133. https://doi.org/10.1109/ROBOT.2008.4543197
11. Ji S-H, Han J-S, Lee S-M, Moon Y-S, Kuc T-Y (2011) Collective searching algorithm for multi-robot system with bounded communication range. In: 2011 8th international conference on ubiquitous robots and ambient intelligence (URAI). pp 180–183. https://doi.org/10.1109/URAI.2011.6145956
12. Lee SG, Diaz-Mercado Y, Egerstedt M (2015) Multirobot control using time-varying density functions. IEEE Trans Robot 31(2):489–493. https://doi.org/10.1109/TRO.2015.2397771
13. Kim K, Park M, Lee S-M, Ji S-H (2012) Development of a dependable network using collective robots with restricted communication range. In: 2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI). pp 408–412. https://doi.org/10.1109/URAI.2012.6463027
14. Papadopoulos Georgios T, Margherita A, Constantine S (2021) Towards open and expandable cognitive ai architectures for large-scale multi-agent human-robot collaborative learning. IEEE Access 9:73890–73909. https://doi.org/10.1109/ACCESS.2021.3080517
15. Nam C, Lee S, Lee J, Cheong SH, Kim DH, Kim C, Kim I, Park S-K (2020) A software architecture for service robots manipulating objects in human environments. IEEE Access 8:117900–117920. https://doi.org/10.1109/ACCESS.2020.3003991
16. Zhang D, Pee LG, Cui L (2021) Artificial intelligence in e-commerce fulfillment: a case study of resource orchestration at Alibaba's smart warehouse. Int J Inf Manag 57:102304. https://doi.org/10.1016/j.ijinfomgt.2020.102304
17. Systems Robotnik (2020) Robotnik in Brain-IoT. https://robotnik.eu/projects/brain-iot-en

18. Armerding T (2020) Security bugs and flaws: both bad, but in different ways. https://www.synopsys.com/blogs/software-security/security-flaws-vs-bugs
19. Crnkovic I, Larsson M (2002) Building reliable component-based software systems. Artech House Inc., USA
20. Selic B (2003) The pragmatics of model-driven development. IEEE Softw 20(5):19–25. https://doi.org/10.1109/MS.2003.1231146
21. Costa B, Pires PF, Delicato FC (2020) Towards the adoption of omg standards in the development of Soa-based IoT systems. J Syst Softw 169:110720. https://doi.org/10.1016/j.jss.2020.110720
22. Abdelhakim B, Otmane AM, Djamal B, Samir O (2019) Safety analysis of train control system based on model-driven design methodology. Comput Indus 105:1–16. https://doi.org/10.1016/j.compind.2018.10.007
23. Baouya A et al (2021) Formal modeling and simulation of collaborative intelligent robots. In: Christian Z et al (eds) Advances in service-oriented and cloud computing. Springer, Cham, pp 41–52
24. Lech J, Radovan S (2021) Proceedings of the 2nd Summer School on Cyber- Physical Systems and Internet-of-Things, vol II. https://doi.org/10.5281/zenodo.5086365
25. El Ballouli R, Bensalem S, Bozga M, Sifakis J (2021) Programming dynamic reconfigurable systems. Int J Softw Tools Technol Transf 23(5):701–719. https://doi.org/10.1007/s10009-020-00596-7
26. Baouya A, Chehida S, et al (2020) A formal modeling and verification of blockchain consensus protocol for IoT systems. In: Hamido F, Ali S (eds) Knowledge innovation through intelligent software methodologies, tools and techniques (SoMeT_20), Kitakyushu, Japan, 20–22 September 2020, vol 327 of frontiers in artificial intelligence and applications. IOS Press, pp 330–342. https://doi.org/10.3233/FAIA200578
27. Baouya A, Chehida S, Ouchani S, Bensalem S, Bozga M (2022) Generation and verification of learned stochastic automata using k-nn and statistical model checking. Appl Intell 52(8):8874–8894. https://doi.org/10.1007/s10489-021-02884-4
28. Mediouni BL et al (2018) Bensalem Saddek S-BIP 2.0: statistical model checking stochastic real-time systems. In: Shuvendu KL, Chao W (eds) Automated technology for verification and analysis. Lecture notes in computer science. Springer, Cham, pp 536–542
29. Nouri A, Mediouni BL, Bozga M, Combaz J, Bensalem S, Legay A (2018) Performance evaluation of stochastic real-time systems with the SBIP framework. Int J Critic Computer-Based Syst 1–33
30. Vicentini F, Askarpour M, Rossi MG, Mandrioli D (2020) Safety assessment of collaborative robotics through automated formal verification. IEEE Trans Robot 36(1):42–61. https://doi.org/10.1109/TRO.2019.2937471
31. Mehrnoosh A, Livia L, Samuele L, Niccolò I, Matteo R, Federico V (2021) Formally-based model-driven development of collaborative robotic applications. J Intell Robot Syst 102(3):59. https://doi.org/10.1007/s10846-021-01386-2
32. Zot (2012) A bounded satisfiability checker. http://github.com/fm-polimi/zot
33. Guiochet J (2016) Hazard analysis of human-robot interactions with hazop-uml. Saf Sci 84:225–237. https://doi.org/10.1016/j.ssci.2015.12.017
34. Matt W, Clare D, Michael F, Maha S, Joe S, Lee KK, Kerstin D, Joan S-P (2016) Toward reliable autonomous robotic assistants through formal verification: a case study. IEEE Trans Human-Mach Syst 46(2):186–196. https://doi.org/10.1109/THMS.2015.2425139
35. Ben-Ari M (2008) Principles of the spin model checker, 1 edn. ISBN 1846287693
36. Dixon C, et al (2014) "The fridge door is open"-temporal verification of a robotic assistant's behaviours. In: TAROS
37. Cimatti A et al (1999) A new symbolic model verifier. In: Nicolas H, Doron P (eds) Computer aided verification. Springer, Berlin, pp 495–499
38. Mohammed A, Furbach U, Stolzenburg F (2010) Multi-robot systems: modeling, specification, and model checking. 01. ISBN 978-953-307-036-0. https://doi.org/10.5772/7349
39. Walter D, Täubig H, Lüth C (2010) Experiences in applying formal verification in robotics. In: Proceedings of the 29th International Conference on Computer Safety, Reliability, and Security. SAFECOMP'10. Springer-Verlag, Berlin, pp 347–360
40. Murray Y, Sirevåg M, Ribeiro P, Anisi DA, Mossige M (2022) Safety assurance of an industrial robotic control system using hardware/software co-verification. Sci Comput Programm 216:102766. https://doi.org/10.1016/j.scico.2021.102766

41. Miyazawa A, Ribeiro P, Li W et al (2019) Robochart: modelling and verification of the functional behaviour of robotic applications. Softw Syst Model 18(5):3097–3149. https://doi.org/10.1007/s10270-018-00710-z
42. MathWorks (2021) Simulink design verifier. Accessed 1 Oct from https://www.mathworks.com/products/simulink-design-verifier.html
43. Gibson-Robinson T, Armstrong P, Boulgakov A, Roscoe AW (2014) Fdr3—a modern refinement checker for csp. In: Erika Á, Klaus H (eds) Tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 187–201
44. Baxter J, Ribeiro P, Cavalcanti A (2022) Sound reasoning in tock-csp. Acta Inf 59(2):125–162. https://doi.org/10.1007/s00236-020-00394-3
45. Livia L, Davide Z, Bersani Marcello M, Matteo R (2023) Specification, stochastic modeling and analysis of interactive service robotic applications. Robot Auton Syst. 163:104387. https://doi.org/10.1016/j.robot.2023.104387
46. David A, Larsen KG, Legay A, Mikučionis M, Poulsen DB (2015) Uppaal smc tutorial. Int J Softw Tools Technol Trans 17(4):397–415
47. Chowdhary RR, Chattopadhyay MK (2021) Orchestration of automated guided mobile robots for transportation task in a warehouse like environment. In: 2021 Emerging trends in industry 4.0 (ETI 4.0), pp 1–7. https://doi.org/10.1109/ETI4.051663.2021.9619354
48. Delgado C, Zanzi L, Li X, Costa-Pérez X (2022) Oros: orchestrating ros-driven collaborative connected robots in mission-critical operations. In: 2022 IEEE 23rd international symposium on a world of wireless, mobile and multimedia networks (WoWMoM), pp 147–156. https://doi.org/10.1109/WoWMoM54355.2022.00026
49. Tahir A, Saghar K, Khalid HB, Shadab BU, Khan US, Asad U (2019) Formal verification and development of an autonomous firefighting robotic model. In 2019 International Conference on Robotics and Automation in Industry (ICRAI), pp 1–6. https://doi.org/10.1109/ICRAI47710.2019.8967388
50. Danielle S et al (2021) AADL-based safety analysis using formal methods applied to aircraft digital systems. Reliab Eng Syst Saf 213:107649. https://doi.org/10.1016/j.ress.2021.107649
51. Simonds D (2017) Prism. Prism statistical model checker. http://www.prismmodelchecker.org/manual/RunningPRISM/StatisticalModelChecking
52. Baouya A, Mohamed OA, Ouchani S, Bennouar D (2021) Reliability-driven automotive software deployment based on a parametrizable probabilistic model checking. In: Expert Systems with Applications, pp 114572. https://doi.org/10.1016/j.eswa.2021.114572
53. Baouya A, Mohamed OA, Ouchani S (2023) Toward a context-driven deployment optimization for embedded systems: a product line approach. J Supercomput 79(2):2180–2211. https://doi.org/10.1007/s11227-022-04741-8
54. Baouya A et al (2016) A formal approach for maintainability and availability assessment using probabilistic model checking. In: Salim C et al (eds) Modelling and implementation of complex systems. Springer, Cham, pp 295–309
55. Kwiatkowska M, Norman G, Parker D (2011) Prism 4.0: verification of probabilistic real-time systems. In: Ganesh G, Shaz Q (eds) Computer Aided Verification, vol 6806. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp 585–591
56. Chehida S et al (2022) Brain-iot architecture and platform for building iot systems. In: Proceedings of the 7th International Conference on Internet of Things, Big Data and Security - IoTBDS, pp 67–77. INSTICC, SciTePress. https://doi.org/10.5220/0011086000003194
57. Basu A, Bensalem S, Bozga M, Combaz J, Jaber M, Nguyen T-H, Sifakis J (2011) Rigorous component-based system design using the bip framework. IEEE Softw 28(3):41–48
58. Agha G, Palmskog K (2018) A survey of statistical model checking. ACM Trans Model Comput Simul 28(1):1–39
59. Brambilla M, Cabot J, Wimmer M (2012) Model-Driven Software Engineering in Practice. 1:9. https://doi.org/10.2200/S00441ED1V01Y201208SWE001
60. Mediouni BL, Nouri A, Bozga M, Dellabani M, Combaz J, Legay A, Bensalem S (2018) SBIP 2.0: statistical model checking stochastic real-time systems. https://www-verimag.imag.fr/TR/TR-2018-5.pdf
61. Baouya A (2021) Java code generator. https://github.com/hakimuga/Resulted_Orchestration_Bundles
62. CPS4EU (2019–2022) Cyber physical systems for Europe. https://cps4eu.eu
63. FOCETA (2021–2023) Foundations for continuous engineering of trustworthy autonomy. http://www.foceta-project.eu

64. CITADEL (2021–2023) Critical infrastructure protection using adaptive MILS. http://www.citadel-project.org
65. CEA List (2019) SensiNact Gateway. Accessed 17 Jan 2020 from https://wiki.eclipse.org/SensiNact
66. Abdelhakim Baouya, Djamal Bennouar, Ait Mohamed Otmane, Samir Ouchani (2015) A quantitative verification framework of sysml activity diagrams under time constraints. Exp Syst Appl 42(21):7493–7510
67. Bliudze S, Sifakis J (2008) The algebra of connectors-structuring interaction in BIP. IEEE Trans Comput 57(10):1315–1330. https://doi.org/10.1109/TC.2008.26
68. Younes HLS, Simmons RG (2002) Probabilistic verification of discrete event systems using acceptance sampling. In: Ed B, Kim GL (eds) Computer aided verification. Springer, Berlin, pp 223–235
69. Hérault T, Lassaigne R, Magniette F, Peyronnet S (2004) Approximate probabilistic model checking. In: Verification, model checking, and abstract interpretation. Springer, Berlin, pp 73–84
70. Dellabani M, Combaz J, Bensalem S, Bozga M (2019). Local planning semantics: a semantics for distributed real-time systems. https://doi.org/10.4230/LITES-v006-i001-a001
71. Giannopoulou G et al DOL-BIP-critical: a toolchain for rigorous design and implementation of mixed-criticality multi-core systems. http://link.springer.com/10.1007/s10617-018-9206-3
72. Robotnik (2020) Json file libraries for robot communication. https://github.com/hakimuga/Robotnik-JSON-Files
73. ROS.org. Ros - stage. 2012. http://wiki.ros.org/stage
74. ROS.org. Ros - rviz. 2012. http://wiki.ros.org/rviz

Authors and Affiliations

Abdelhakim Baouya[1] · Salim Chehida[2] · Saddek Bensalem[2] · Levent Gürgen[3] · Richard Nicholson[4] · Miquel Cantero[5] · Mario Diaznava[6] · Enrico Ferrera[7]

Salim Chehida
salim.chehida@univ-grenoble-alpes.fr

Saddek Bensalem
saddek.bensalem@univ-grenoble-alpes.fr

Levent Gürgen
levent@kentyou.com

Richard Nicholson
Richard.Nicholson64@protonmail.com

Miquel Cantero
mcantero@robotnik.es

Mario Diaznava
mario.diaznava@st.com

Enrico Ferrera
enrico.ferrera@linksfoundation.com

1 CNRS, UT2, IRIT, Université de Toulouse, Toulouse, France

2 VERIMAG, Université Grenoble Alpes, Grenoble, France

3 Kentyou, Grenoble, France

4 Amazon Web Services (AWS), London, UK

5 Robotnik Automation, Valencia, Spain

6 STMicroelectronics, Grenoble, France

7 LINKS Foundation, Turin, Italy