University of Camerino

School of Advanced Studies

Doctor of Philosophy in Computer Science

Process-driven Development and Analysis of Multi-Robot Systems

Supervisor

Prof. Francesco Tiezzi

Co-Supervisor

Dr. Lorenzo Rossi

PhD Candidate

Sara Pettinari

XXXVI Cycle

"Just when you think you know something,
you have to look at it in another way."

Abstract of the Dissertation

Robots capable of cooperating to accomplish a common mission, i.e., multi-robot systems, are emerging to perform complex tasks in different application domains, by supporting or automatizing human tasks. The behavioral workflow of these systems, referred to as mission, can be seen as a sequence of tasks enabling both robots' actions and inter-robot interactions. Hence, considering robots' behavior at a high level allows for conceptualizing their missions as process models. In the context of process models, Business Process Management (BPM) is the reference discipline that enables their usage to describe the overall workflow of a system, including its design, enactment, monitoring, analysis, and refinement.

Using process models in the robotics domain can leverage the techniques belonging to the BPM discipline. Specifically, two primary approaches can be employed. A top-down approach involves designing processes to model the robotic mission, facilitating its execution according to the planned sequence. This concept belongs to model-driven approaches, utilizing high-level modeling languages to facilitate both the modeling and enactment phases. Differently, a bottom-up approach leverages data generated during robot operations to discover and analyze mission executions. This approach exploits process mining techniques to automatically extract insights related to the multi-robot execution.

Although BPM is a well-established discipline, its integration with the robotics domain has not been sufficiently investigated. Nevertheless, combining process models with robotic systems holds significant potential, lightening the effort to program the behavior of robots and their interactions, and enabling the analysis of different aspects impacting multi-robot executions. Indeed, process models can provide a robust solution for modeling and en-

acting the system. Whereas, data captured by these systems can enhance process models with insights from the physical world.

This thesis focuses on investigating the application of BPM techniques in the robotics domain, aiming to outline the advantages and challenges of this integration. The thesis proposes a top-down and a bottom-up approach to leverage process models for multi-robot systems development and analysis. The top-down approach consists of a process-driven development of multi-robot systems by employing business processes to design the system's behavior and integrating direct process enactment into the robotic architecture. In the opposite direction, the bottom-up approach offers a methodology to facilitate the extraction of event logs, which represent robotic activities, during the execution of robotic systems. These event logs enable the mining of the process model depicting the system workflow and system analysis from various perspectives.

List of Publications

[J3]. F. Corradini, **S. Pettinari**, B. Re, L. Rossi, F. Tiezzi. *A Technique for Discovering BPMN Collaboration Diagrams.* Software and Systems Modeling, (2024): 1-21.

[J2]. F. Corradini, **S. Pettinari**, B. Re, L. Rossi, F. Tiezzi. *Executable Digital Process Twins: Towards the Enhancement of Process-Driven Systems.* Big Data and Cognitive Computing 7 (3), (2023): 139.

[J1]. F. Corradini, **S. Pettinari**, B. Re, L. Rossi, F. Tiezzi. *A BPMN-driven framework for Multi-Robot System development.* Robotics and Autonomous Systems 160, (2023): 104322.

[C4]. F. Corradini, **S. Pettinari**, B. Re, L. Ruschioni, F. Tiezzi. *Enhancing Compatibility in QoS Communication for the Internet of Robotic Things.* ER Forum/Posters/Demos, vol. 3618. (2023)

[C3]. F. Corradini, **S. Pettinari**, B. Re, L. Rossi, F. Tiezzi. *A Methodology for the Analysis of Robotic Systems via Process Mining.* Enterprise Design, Operations, and Computing 2023: 117-133.

[C2]. F. Corradini, **S. Pettinari**, B. Re, L. Rossi, F. Tiezzi. *An Approach to Support Digital Process Twin.* DASC/PiCom/CBDCom/CyberSciTech 2022: 1-4.

[C1]. K. Bourr, F. Corradini, **S. Pettinari**, B. Re, L. Rossi, F. Tiezzi. *Disciplined use of BPMN for mission modeling of Multi-Robot Systems.* Forum at Practice of Enterprise Modeling, vol. 3045, pp. 1–10. (2021)

Contents

List of Figures

List of Tables

Listings

Part I

Introduction and Background

CHAPTER 1

Introduction

Robots are systems capable of sensing, processing, and physically reacting to information from the real world. Due to their abilities, they can substitute humans in many fields, from industrial to safety-critical ones [77]. They help to minimize human errors in repetitive tasks and can improve productivity and service quality in tasks that involve human interaction [39]. Additionally, robots offer benefits by reducing the need for humans to perform potentially dangerous tasks like handling toxic materials or participating in rescue missions during disasters [29].

In recent years, robots' capabilities have been combined to perform complex tasks that would otherwise be impossible for one single robot to accomplish. This concept is based on using a Multi-Robot System (MRS) where smaller sub-tasks are distributed to individual robots capable of interacting with each other to achieve a common objective [39]. From a high-level point of view, the sequence of tasks that the robots have to perform to accomplish the objective is referred to as mission [63].

The rest of the chapter introduces the motivations behind the research presented in this thesis and the challenges of existing methodologies. After that, it presents the research objectives it aims to achieve and the corresponding research outcomes. Finally, the structure of the thesis is provided.

1.1 Motivations

An MRS is a group of heterogeneous robots that act on the environment and interact to accomplish a common mission. A single entity has few functionalities as the real power lies in the cooperation among robots. Unlike single-robot systems, MRSs can fulfill tasks faster and cheaper, guaranteeing scalability, reliability, flexibility, and versatility [39]. Therefore, the cooperative mission of the entire system is generated through the behavior of individual robots and their interactions, eliminating the necessity for centralized control. Indeed, depending on the robots' behavior, how they coordinate with each other, and different aspects impacting the system, the mission of the robots may succeed or fail.

From a behavioral point of view, the mission is the core element that should be taken into account during the specification, execution, analysis, and enhancement of the overall MRS. This focus, combined with the ability of an MRS to autonomously execute tasks and interact with the physical environment in a similar way as human users, enables the mission to be conceptualized as a process model [64]. Specifically, a process model serves as an abstract representation of the flow of tasks executed by a system over time. In the context of process models depicting a system workflow, Business Process Management (BPM) is the reference discipline. Indeed, BPM is a well-established discipline that considers business process models as a core element to refer to the work an organization performs to achieve its objectives. It provides a lifecycle to circularly support the identification, discovery, analysis, (re-)design, implementation, execution, monitoring, and evolution of business process models [32]. Following a **top-down approach**, business processes permit modeling the referenced system and enacting its execution [45]. This concept belongs to model-driven approaches, utilizing high-level modeling languages like Business Process Modeling and Notation (BPMN)[68] to facilitate both the modeling and enactment phases. In contrast, a **bottom-up approach** enables the analysis of process-driven and process-agnostic systems, where data generated from system execution is used to discover and analyze related processes [91]. This approach exploits process mining techniques [92] to extract and provide insights into the process execution automatically.

1.2 Challenges

Considering an MRS mission as a process model, the application of a BPM-based top-down approach in the robotics domain is in line with the need for model-driven solutions that support efficient and flexible methodologies for modeling and enacting robotic systems [67]. The current state of the art em-

phasizes the necessity of such solutions to track the challenges of specifying high-level robots' behavior and interactions [26]. The main motivation for applying model-driven approaches lies in existing robotic frameworks, which are used to avoid programming robots from scratch by providing hardware and application abstraction layers [16]. However, the definition of the MRS mission using these frameworks demands expertise in low-level programming and intricate decision-making algorithms from robotic experts. This hinders MRS developers from **specifying high-level robots' behavior and inter-robot interactions** to enable a cooperative MRS. Existing model-driven approaches proposed to overcome these challenges often lack high-level abstraction of the entire mission or discard scenarios involving multiple robots. Utilizing BPMN can address this gap by enabling high-level mission modeling and enactment [27], thereby facilitating the development of cooperative MRSs by identifying robots' behaviors and interactions through process models.

At the same time, during an MRS execution, relevant data should be collected and analyzed to determine if the mission is performing as expected. Applying a BPM-based bottom-up approach in the robotics domain aligns with the need to propose analysis methodologies to ensure a proper system workflow [2]. Indeed, the analysis of an MRS should take into account that the execution of a robot strictly depends on the capability to control several sensors and actuators in real-time, and on its interaction with the environment. With the increasing adoption of MRSs, many methodologies have been developed to enable the analysis of these systems. The most used ones presented in the literature range from formal verification [57] to quantitative analysis [2]. Formal verification approaches are mainly exploited to analyze specific aspects of a mission, such as identifying deadlocks, state reachability, or monitoring a particular property during system execution. Nevertheless, considering the complex nature of MRSs, the necessity of combining the analysis of different system perspectives can lead to the state-space-explosion problem [89]. Differently, quantitative analysis seeks to deploy the system to observe its behavior. However, performing these approaches on real robots involves both risks to human life and economic risks to robot hardware, which could get damaged due to the deployment of incorrect behaviors. Whereas, running them in simulation environments requires constant monitoring of system execution, thereby consuming considerable time, and leads to subjective analysis results based on human observations and evaluations. To overcome these limitations in the analysis of an MRS, the application of process mining techniques aims at **addressing the complexity of understanding the behavior of the robots and their interactions** [75] while **checking the correctness of the whole system execution** [66]. These would enable the reconstruction of an MRS mission through process models

extracted from event logs generated during the execution (in a physical or simulation environment) of the system, and the identification of recurrent errors, or deviations with respect to the intended mission.

1.3 Research Objectives

The purpose of this thesis is to provide novel solutions for supporting the modeling and enactment of the MRS mission as well as the analysis of the MRS executions. To this aim, the thesis considers the MRS mission as a process model aimed at enhancing system execution [64]. Indeed, adopting a top-down approach, process models can provide a robust foundation for developing the MRS in a model-driven fashion [45]. Whereas, following a bottom-up approach, data collected by an MRS can enable the analysis of its mission and derive process models enhanced with insights from real-world observations.

The **top-down approach** consists of the specification of the high-level behavior and interactions of MRSs, thus facilitating the development of these systems via process models. In this approach, the following research objectives have been identified.

(i) Support the representation of the MRS cooperative behavior using BPMN as a modeling language.

(ii) Enrich the BPMN process model with the information required for the MRS implementation.

(iii) Assess the usage of BPMN for driving the MRS execution.

The **bottom-up approach** aims to provide techniques capable of analyzing, via process models, the behavior performed during an MRS execution, and the perspectives impacting the system. In this approach, the following research objectives have been identified.

(i) Support the extraction of robotic data in a process mining-compatible manner.

(ii) Assess the usage of process mining techniques to discover and analyze the behavior of the MRS.

(iii) Enhance process mining techniques to deal with multiple perspectives impacting MRS execution.

1.4 Research Outcomes

To meet the research objectives for the top-down and bottom-up approaches, the thesis respectively proposes FAME, a BPMN-driven framework for MRS development, and TALE, a tag-based multi-perspective methodology.

The FAME framework achieves the **top-down approach** objectives as follows.

(i) Support the usage of BPMN by providing a set of guidelines capable of guiding the MRS designer in the high-level system modeling of the MRS cooperative behavior.

(ii) Enrich the designed process model by facilitating the enhancement of its elements with information related to the robotics domain and required for MRS execution.

(iii) Assess the usage of the process model to directly enact the system by providing BPMN-engines capable of driving the MRS execution.

The TALE methodology achieves the **bottom-up approach** objectives as follows.

(i) Support the extraction of event logs from MRS execution by providing a methodology for preparing activity-centric event logs.

(ii) Assess the usage of process mining to analyze the MRS by discovering the executed behavior in the form of a process model.

(iii) Enhance process mining techniques by integrating additional visualizations that support the analysis of multiple MRS perspectives impacting system execution.

1.5 Thesis Structure

The thesis is organized into four parts and eight chapters. The structure is detailed as follows.

Part I - Introduction and Background. The first part introduces the motivation of the thesis, the background concepts, and the running scenario used throughout the thesis.

- Chapter 2 provides all the fundamental concepts required for the thesis. It includes a detailed description of the robot operating system framework, the BPM discipline, and process mining techniques.

- Chapter 3 introduces the multi-robot smart agriculture system designed to serve as a running scenario.

Part II - Top-Down Approach. The second part describes the top-down approach for developing an MRS via BPMN collaboration diagrams, which is based on the FAME framework.

- Chapter 4 presents the FAME framework's phases that support the entire MRS development, from the mission modeling to the system enactment.

- Chapter 5 focuses on the implementation of the tool supporting the FAME phases, its evaluation in the running scenario, and its evaluation in a small-scale physical scenario.

Part III - Bottom-Up Approach. The third part describes the bottom-up approach designed to ease data extraction from MRS execution to perform process mining-based analysis, which relies on the TALE methodology.

- Chapter 6 illustrates current challenges in extracting robotic data in an activity-centric manner. After that, it presents the TALE methodology and the steps composing it.

- Chapter 7 focuses on the implementation of the tool supporting the TALE steps and its evaluation in the running scenario.

Part IV - Concluding Remarks. The last part concludes the work.

- Chapter 8 concludes the thesis resuming the contributions, discussing the obtained results, and providing an overview of future works.

CHAPTER 2

Background

This chapter introduces all the concepts used to fully understand the contribution of this thesis. Firstly, the Robot Operating System (ROS) framework, its architecture, and the advantages of its application are presented. Subsequently, the BPM discipline is presented, with a focus on the BPMN standard and collaboration diagrams. Finally, the process mining discipline and its main concepts are introduced.

2.1 Robot Operating System

Over the years many middleware architectures have been proposed to make robot development easier and to provide programming abstractions that help in managing the complexity and heterogeneity of hardware and applications. Among the existing frameworks, ROS is the most prominent one, which boasts a big and active community and focuses on the support of MRSs [78].

ROS is an open-source and flexible software framework for programming robots. It provides an abstraction layer in which developers can build robotics applications without worrying about the underlying hardware [52]. The core of this framework is a message-passing middleware in which processes can communicate and exchange data with each other, even when they are running on different machines.

The primary goal of ROS is to support code reuse in robotics research and development. So, it is designed as a distributed framework of executable processes that can be individually designed and loosely coupled at run-time.

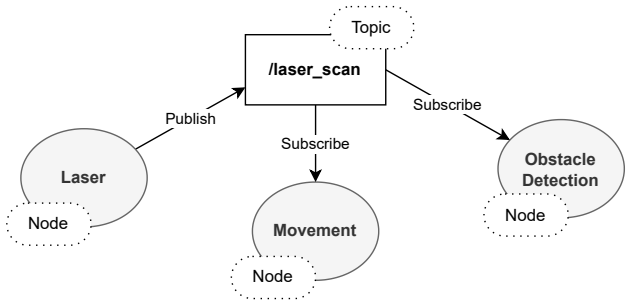


Figure 2.1: ROS nodes interaction

These processes can be grouped into packages and stacks, which can be easily shared and distributed [38]. Moreover, by exploiting the hardware abstraction layer, almost all ROS packages can be reused for different robots. This enables collaborative development, in which developers can join the ROS community to contribute or find a package for a specific purpose.

More in detail, the independent executable processes composing a ROS application are called **nodes**. Each node is designed to fulfill specific functionality, such as collecting data from the distance sensor, which promotes fault isolation, faster development, and modular and reusable code. The communication between nodes is based on a *publish/subscribe* model, in which nodes communicate by passing messages via a **topic** related to specific data structures. The nodes use the topic name to identify the content of the message. In detail, when a node publishes a message on a topic, a subscriber to that topic can utilize it. Figure 2.1 shows the interactions between three nodes. Indeed, the *laser* node publishes laser data over the */laser_scan* topic, whereas, the *movement* and the *obstacle detection* nodes subscribe to it so that they can access laser data. Nodes can be created by leveraging *client libraries*, facilitating their development through widely adopted programming languages. Although C++ and Python are common choices, these libraries are also accessible for languages such as Java, JavaScript, and C#.

In recent years, the Open Source Robotics Foundation developed the next generation of ROS, to fully support the development of MRSs. The new version, called ROS2, has been designed to support new functionalities. First of all, it natively supports the development of MRSs, by improving the network performance of communication. Additionally, ROS2 not only runs on Linux systems, but also adds support for Windows, MacOS, RTOS, and other systems, giving developers more choices. It supports real-time control, which can improve the timeliness of control and the performance of the overall robot. Moreover, ROS2 aims to bridge the gap between prototypes and products, shifting from the scientific research field to the application of robots, thus enabling to carry ROS2 systems directly to the market.

Specifically, the multi-robot support is enabled by the integration of the

OMG Data Distribution Service (DDS)[1] standard. DDS enables reliable, high-performance, interoperable, real-time, and scalable data exchanges using a publish/subscribe pattern. It provides a default distributed discovery system, which is required to use DDS's publish-subscribe transport. This allows DDS programs to communicate without the need for a central orchestrator. In this way, a system becomes more fault tolerant and flexible [61]. Many vendors provide DDS and have several implementation types, so developers can select an appropriate DDS implementation from various vendors.

In ROS2, DDS is implemented through a special layer, i.e., the ROS middleware interface layer, in which all DDS-specific APIs and message definitions are hidden. The core of DDS is a Data-Centric Publish-Subscribe (DCPS) model, depicted in Figure 2.2, that creates a global data space accessible by any independent applications. In this model, several entities can be identified:

- **Participant** is a publisher or subscriber and corresponds to a ROS node.
- **Publisher** supports the publishing of multiple data types and can be associated with multiple Data Writers to publish messages on one or more topics.
- **Subscriber** receives published data and can be associated with multiple Data Readers and subscribes to messages on one or more topics. It is responsible for receiving published data and making the data available.
- **Data Writer** is an object used by a Participant to publish data through a Publisher. Each Data Writer corresponds to a specific topic.
- **Data Reader** is an object attached to a Subscriber. A Participant can receive and access data such that the type corresponds to the one of the Data Writer. Each Data Reader corresponds to a specific topic.
- **Topic** defines a name and a data structure. It is used to identify each data object exchanged between a Data Writer and a Data Reader.
- **Quality of Service (QoS) Policy** controls all aspects of the communication mechanism with the underlying layer, mainly from the aspects of time limit, reliability, and history to meet user data requirements for different scenarios.

In the DCPS model, data of a given type is published from one or several Data Writers to a topic, uniquely identified by its name. One or more Data

[1] https://www.omg.org/omg-dds-portal

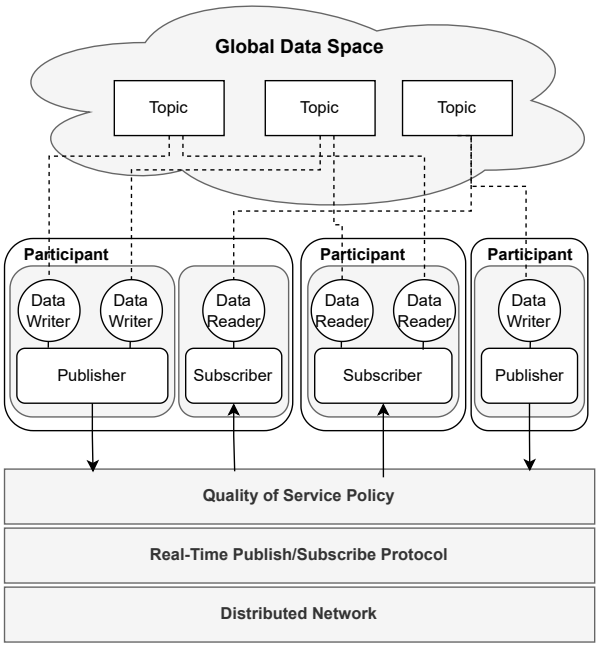


Figure 2.2: Data-centric publish-subscribe model [61]

Readers identify a data object by topic name to subscribe to the topic. After this transaction, a DataWriter connects to a DataReader using the Real-Time Publish/Subscribe (RTPS) protocol[2]. The RTPS protocol is flexible and enables communication among publishers and subscribers based on the QoS policy. Indeed, a QoS policy is used to define the communication requirements a system must provide [83]. Exploiting the DCPS model enables DDS users to generate code as a Domain Participant, including QoS Policies using the DDS APIs. Thus, users can focus solely on their purpose and determine ways to satisfy real-time constraints easily. Figure 2.3 shows an example of DDS data transport following a QoS policy. The deadline period, history depth, and communication reliability parameters are configured by a QoS policy. Focusing on QoS policy, the DDS protocol provides a rich set of QoS policies for controlling data distribution. These policies refer to various communication parameters, such as data availability, resource usage, reliability, and timing. Specifically, ROS implements a subset of the QoS policies provided by the DDS protocol, as figured out in the feature diagram in Figure 2.4. This representation describes the policies considered by ROS with the different values they can assume, as well as the dependencies that may occur between them.

Finally, robot development goes hand in hand with simulation environments. Indeed, simulators have played a critical role in robotics research

[2]https://www.omg.org/spec/DDSI-RTPS/2.2/

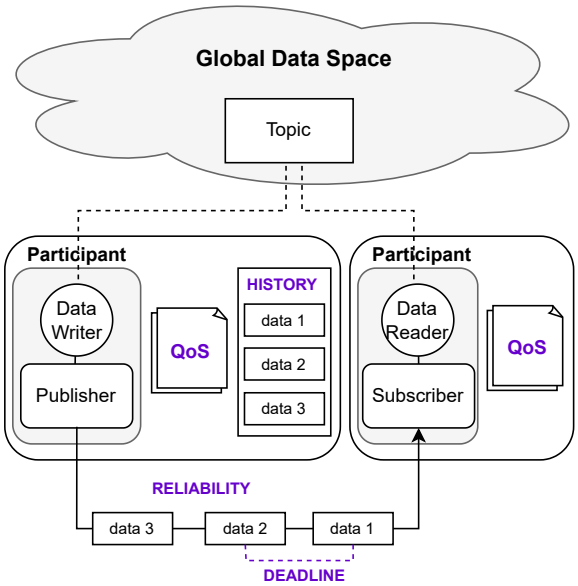


Figure 2.3: DDS QoS policy [61]

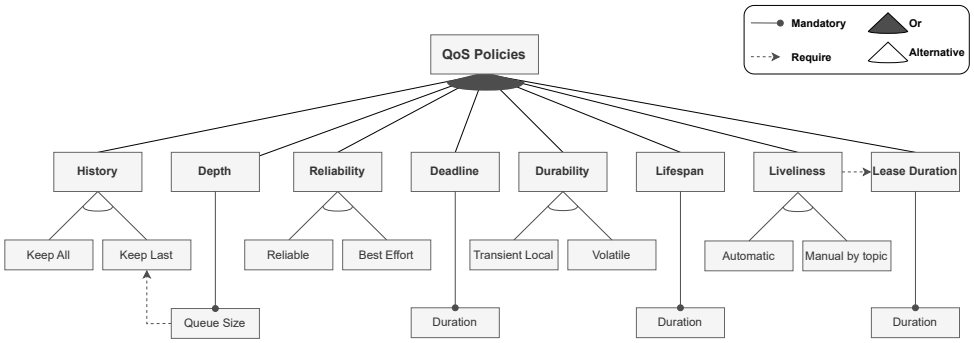


Figure 2.4: Feature model of ROS QoS policies

as tools to quickly and efficiently test new concepts, strategies, and algorithms [51]. ROS is capable of interaction with many open-source simulators and 3D environments, in particular, the most used is Gazebo[3]. The Gazebo simulator is designed to faithfully simulate the physics and behavior of its physical counterpart. As a result, a developer can monitor not only the behavior of the robot but also the data perceived by sensors and the actions performed by actuators.

From now on, for the sake of presentation, the framework will be referred to simply as ROS.

[3]https://gazebosim.org/

2.2 Business Process Management

BPM is the discipline that supervises the work conducted by organizations to ensure consistent outcomes and to take advantage of improvement opportunities [32]. In particular, BPM aims to manage the entire chain of events, activities, and decisions connected to an organization. These chains are called business processes and BPM includes concepts, methods, and techniques to support their design, administration, configuration, enactment, and analysis [104].

Business processes derive from the observation of products or services provided by a company. Business processes are the outcome of several activities and are the key instrument for organizing these activities and improving the understanding of their interrelationships. BPM is characterized by a set of steps that occur cyclically to adapt and improve the model. Hence, BPM involves a continuous cycle, comprising the following phases: modeling, analysis, execution, and monitoring.

Over the years, different languages and graphical notations have been proposed to represent business process models, differing both in the possibility to express aspects related to the organization's perspectives and in the level of formality used to define the elements composing the notation. **BPMN 2.0** [68] is currently acquiring a clear predominance. It has been standardized by the OMG organization and it is now widely accepted both in industry and academia. Its first goal is to provide a notation that is readily understandable by all business users. This includes the business analysts in the creation of the initial drafts of the processes to the technical developers responsible for implementing the technology that will perform those processes. BPMN's success comes from its versatility and capability to represent business processes with different levels of detail and for different purposes. Business process models are expressed in business process diagrams and each diagram consists of a set of modeling elements. To foster usage and interchangeability of BPMN between different tools, the OMG permits sharing diagrams in a standard manner. Indeed, it defines unique XML-based notation in which a business process is described in a tree-structured way, bringing all the information required for reproducing the elements composing the diagram, and their style. Indeed, each BPMN element can be mapped to an XML fragment containing semantic and visual information. The first part of the fragment depicts the semantic information (e.g., the element id, the connected sequence flows), while the second part graphically locates the element in the diagram.

BPMN allows to design of different kinds of diagrams: process, collaboration, choreography, and conversation diagrams. Specifically, **collaboration diagrams** can be employed for depicting processes in a distributed system.

Within these diagrams, various BPMN elements are utilized to model the intended behavior of the referenced system. The following is a description of a subset of BPMN elements that will aid in comprehending the work presented in the subsequent chapters.

- **Pools** (Figure 2.5) are used to represent participants or organizations involved in the collaboration and include details on internal process specifications and related elements. Pools are drawn as rectangles, and they usually have a name. BPMN allows the assignment of a multi-instance marker (three vertical lines) to a pool, representing multiple instances playing the same role.

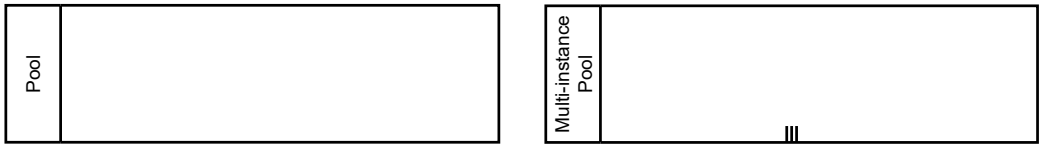


Figure 2.5: BPMN pools

- **Activities** (Figure 2.6) are used to represent a specific work to be performed within a process. They are drawn as rectangles with rounded corners. A *Script Task* is an automated activity that contains a script. When a process execution arrives at this element, the corresponding script is executed. A *Call Activity* references a process that is external to the process definition. Therefore, the call activity enables the reusability of a process that can be called from multiple other process definitions. An *Event Sub-Process* is integrated within a process and is activated when its start event is triggered. An event sub-process may be interrupting or non-interrupting. An interrupting sub-process cancels any executions in the current scope. A non-interrupting one creates a new concurrent execution. While an interrupting event sub-process can only be triggered once for each activation of the scope hosting it, a non-interrupting one can be triggered multiple times.

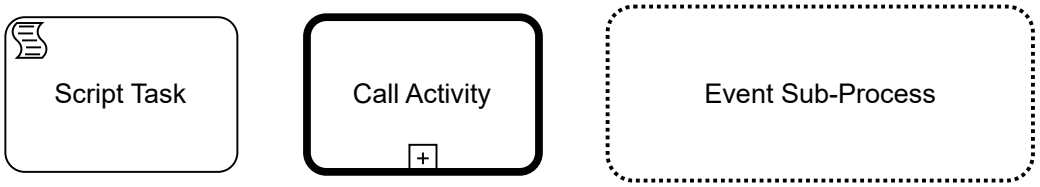


Figure 2.6: BPMN activities

- **Events**(Figure 2.7) are used to represent something that can happen. An event can be a *Start Event* representing the point from which a pro-

cess starts, an *Intermediate Event* indicates that something has happened (i.e. intermediate throw events), or waits and reacts to certain events (i.e. intermediate catch events), or an *End Event* representing the process termination. Moreover, *Terminate* end event triggers the termination of all the active instances in the process. Events are drawn as circles. Specifically, events can be of different types.

- *None* events are unspecified events, also called "blank" events.
- *Timer* events are events triggered by a defined timer.
- *Conditional* events define an event that is triggered if a given condition is evaluated as true.
- *Error* events model the deviations of a process workflow to react to errors.
- *Signal* events reference a signal. Broadcasting a signal triggers all signal events with a matching name. The broadcast iterates over subscriptions, creating a process instance if the broadcasted signal's name matches the name of the signal start event.

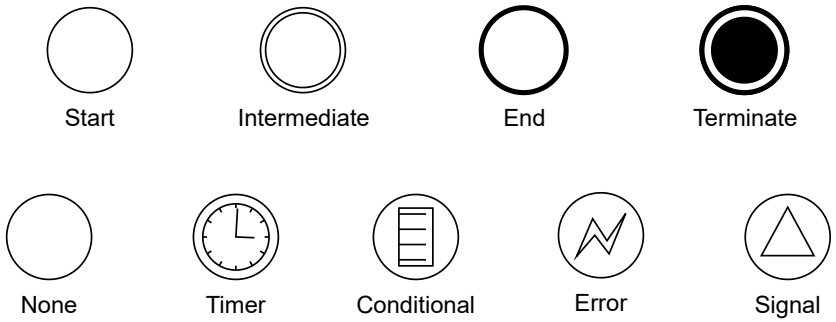


Figure 2.7: BPMN signals

- **Gateways** (Figure 2.8) are used to manage the flow of a process. Gateways are drawn as diamonds. Different types of gateways are available.
  - *Exclusive (or XOR)* gateway is used to create alternative paths within a process flow. For a given instance of the process, only one of the paths can be taken.
  - *Parallel (or AND)* gateway is used to represent two tasks in a business flow. A parallel gateway is used to visualize the concurrent execution of activities.
  - *Event-based* gateway allows making decisions based on events. It must have at least two outgoing sequence flows. Each sequence flow must be connected to an intermediate catch event. When an

event-based gateway is entered, the process instance waits at the gateway until one of the events is triggered. When the first event is triggered, the outgoing sequence flow of this event is taken. No other events of the gateway can be triggered afterward.

Figure 2.8: BPMN gateways

- **Connecting Edges** (Figure 2.9) are used to connect process elements inside different pools. *Sequence Flow* are solid connectors used to specify the internal flow of the process, thus ordering elements in the same pool. Differently, *Data Association* is used to associate data elements to activities.

Figure 2.9: BPMN connecting edges

- **Data Objects**(Figure 2.10) allow showing and storing data flowing through a process by passing information into or out of an activity. They are depicted as a document with the upper-right corner folded over, and linked to activities with a data association arrow. The direction of the data association is used to establish whether a data object is an input or output for a given activity.

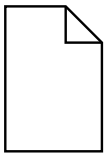


Figure 2.10: BPMN data object

The execution semantics of BPMN is **token**-based [68, Sec.7.1.1]. A token traverses, from a start event, the sequence edges of the process and passes through its elements enabling their execution, and finally, an end event consumes it when it terminates. Process elements acquire one or more tokens

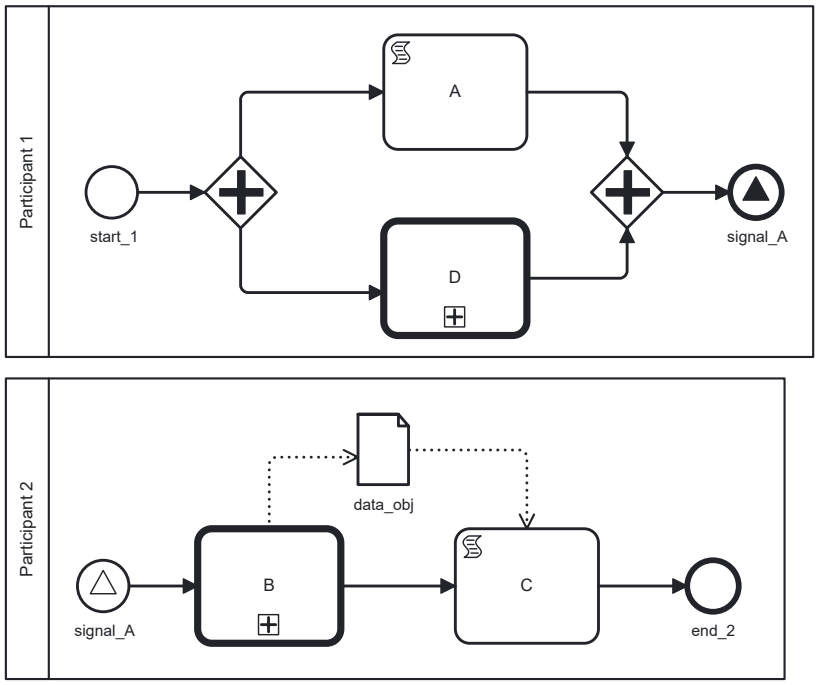


Figure 2.11: BPMN collaboration diagram example

from incoming sequence flows for execution. Once finished, they may produce one or more tokens on outgoing sequence flows, depending on their behavior. Referring to the example depicted in Figure 2.11, the collaboration diagram contains two pools, i.e., *Participant 1* and *Participant 2*. Specifically, the collaboration execution starts with two tokens in both the start events of the pools. The token in the *Participant 1* can immediately start the execution, whereas the token in *Participant 2* is in an idle state waiting to receive the corresponding signal. Therefore, the token in the *start_ 1* event of *Participant 1* executes in parallel the script task A and the call activity D and terminates by throwing signal end event *signal_ A*. This signal is caught by the start event *signal_ A* in *Participant 2* that executes in sequence the call activity B and the script task C and ends once it reaches the end event *end_ 2*. Notably, the activity B produces the data object *data_ obj* that is taken as input by the activity C.

Following the execution semantics, BPMN process models can be directly executed by BPMN **engines**. These engines can consume and execute processes provided in the correct format. Standardization of the format and semantics by BPMN ensures that the execution behavior remains consistent across different engines [40].

2.3 Process Mining

The process mining research field is spreading to analyze processes using event logs generated by information systems, to discover, monitor, and improve the overall system [90]. Indeed, by combining event data and process models, process mining techniques provide insights, identify bottlenecks and deviations, anticipate and diagnose performance and compliance problems, and support the automation or removal of repetitive work [95]. The key input for process mining is an *event log*, i.e., the set of events representing the executed process. Each event in the log records data coming from systems' executions in an activity-centric manner, reporting at least the *activity* name and the *timestamp*. Further attributes can be included in the events to provide context data that provides information about other perspectives. Events referring to the same system execution are grouped into traces of events ordered by time. Within the event log, the system execution identifier is referred to as *case*.

As depicted in Figure 2.12, event logs can be firstly used to conduct three basic types of process mining analysis.

- **Process discovery** techniques take an event log and create a process model that adequately describes the underlying process. Nevertheless, the correctness of the result depends both on the log completeness and on the used discovery algorithm. Usually, process discovery techniques resort to many notations to describe processes, ranging from Directly-Follows Graphs (DFGs) to BPMN and Petri nets [95].

- **Conformance checking** relates events in the event log to activities in the process model and compares both. The goal is to enable the analysis of the quality of a process model discovered from event data, the identification of potential deviations, and the projection of real traces onto process models [14].

- **Process enhancement** aims at changing or extending an existing process model. Specifically, process extension techniques aim to incorporate different perspectives to achieve a high-precision model. Whereas, process improvement techniques are a supervised way to modify an existing model, by producing a model that properly reflects reality [28].

Recently, **object-centric process mining** has been proposed to group the event logs not based on the case notion, but to explore and filter the behavior contained in the logs considering different classes of objects and their interaction [8]. In this context, Event Knowledge Graphs (EKGs) are a flexible and expressive event data model that captures and connects different aspects of the distributed behavior of the system [33]. EKGs are built on

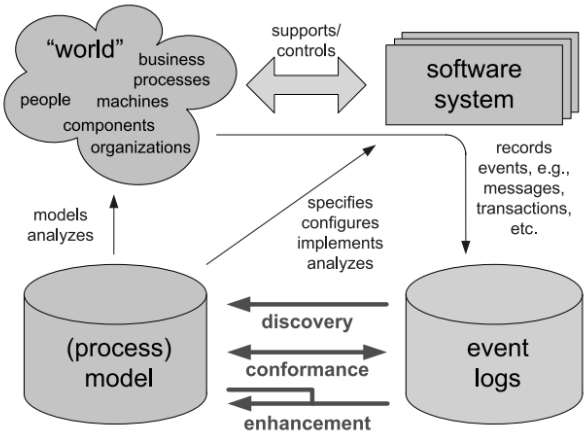


Figure 2.12: The three basic types of process mining: (a) discovery, (b) conformance, and (c) enhancement [90].

the concepts of events, entities (i.e., objects), and relations, which are interconnected to represent the analyzed system accurately. Moreover, EKGs can be queried to manipulate and navigate the resulting graph, facilitating the extraction of desired insights.

Specifically, an EKG is a data structure designed to represent event tables that encompass multiple entity types and maintain the order of events for the correlated entities [34]. EKGs are built upon labeled property graphs, but they have a limited set of node and relationship labels. In an EKG, each event is represented as a node with the label *Event* containing at least the identifier of the performed activity and at which time it has been performed, and each entity is represented as a node with the label *Entity* and a property defining its entity type. The directly-follows relationship denotes the sequential order between events associated with the same entity and is labeled as *df*. Whereas, the relationship between an event and its correlated entity is labeled as *corr*. More in detail, an event e_1 directly follows an event e_2 if (i) they are both correlated to the same entity n, (ii) e_1 occurred before e_2 and (iii) there is no other event occurred between e_1 and e_2 correlated with n. Considering all the df-relationships between events correlated to the same entity, the path along them corresponds to a trace of a traditional event log [34], i.e., an entity identifier serves as a case to create a group of events. Figure 2.13 depicts an example of an EKG, where events are represented with a rectangle, entities as circles, corr-relationships as dashed edges, and df-relationships as solid edges. For instance, the df-path for entity $r1$, with type *resource*, is $\sigma_{r1} = \langle (e1, e2), (e2, e6) \rangle$.

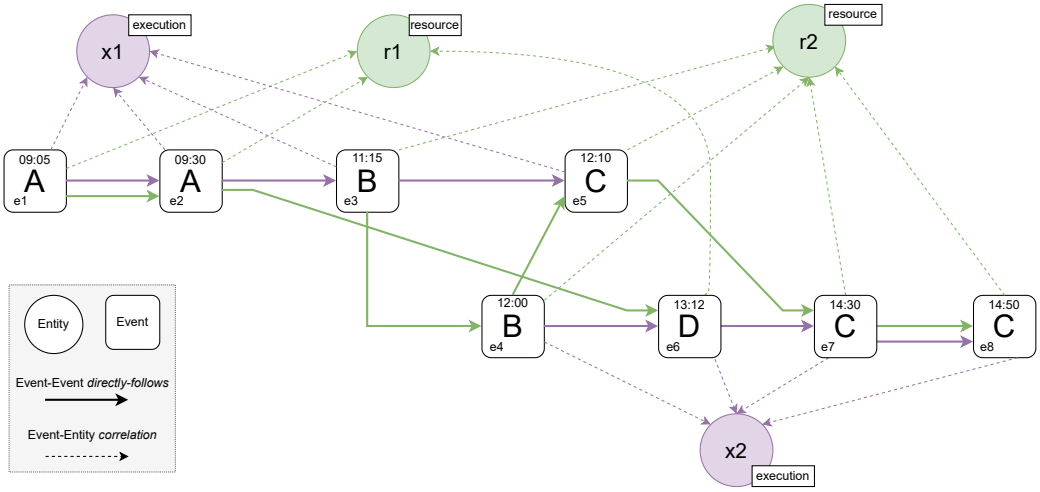


Figure 2.13: EKG example

CHAPTER 3

Running Scenario

In the continuously advancing agricultural domain, the integration of robotic technologies has become essential for improving efficiency and minimizing operational costs. Indeed in this domain, autonomous mobile robots can be used in a variety of operations. Their applications range from the facilitation of capturing and processing high quantities of data to the capabilities of operating both at crop level and at field level [74]. Considering the promising and emerging application of robotic systems, this thesis considers as a running scenario the application of an MRS in an agricultural field aiming to improve crop productivity. Notably, ROS is the reference framework for the implementation of robots in the running scenario. However, the concepts and proposals presented in this thesis can be adapted for MRSs built upon other frameworks.

This chapter illustrates all the concepts useful for understanding the running scenario and its capabilities. First of all, it describes the MRS running scenario applied in an agricultural context and its components. After that, the ROS-based architecture of the involved robots is presented.

3.1 System Description

The multi-robot agriculture scenario, designed as a running scenario, describes an autonomous system where a drone collaborates with two Automated Guided Vehicles (AGVs) to automate the identification and removal of weed grass. To better decline the AGVs in the agricultural domain, from

now on they will be referred to as tractors.

The proposed scenario consists of two tractors and one drone that cooperate to identify and remove weed grass in farmland to increase field productivity. The drone is the first robot that can start its behavior. At the system start-up, it receives the field's boundaries to inspect and starts the exploration. During the overflight of the area, the drone can recognize weed grasses and, when found, it sends to the tractors the coordinates. This triggers the tractors, which store the weed grass coordinates and send their distance to the weed grass area back to the drone. The drone can hence elect the closest tractor and notify it. At this point, the selected tractor starts moving towards the field. Once it reaches the weed, it activates the blade to cut the weed and stops its process until it receives a new position from the drone.

More in detail, the robots composing the scenario are composed as follows.

- The **drone** is equipped with a battery and several sensors to identify the weed grass, and can autonomously navigate the agricultural field. Its primary task is to explore the field and communicate the position of the identified weeds to the tractors.

- The **tractors** are equipped with a battery and several sensors for navigation and obstacle avoidance in the field. Moreover, each tractor is equipped with a blade enabling weed grass removal. To properly perform the operations, tractors can communicate with the drone that shares the position of the identified weeds and manages the coordination between them.

Within the field, distinct areas serve specific purposes. The cultivated area is dedicated to the operation of the robots. The drone base station serves as a taking-off, landing, and charging area. Differently, the tractors' base station contains the charging area. A high-level system representation is depicted in Figure 3.1.

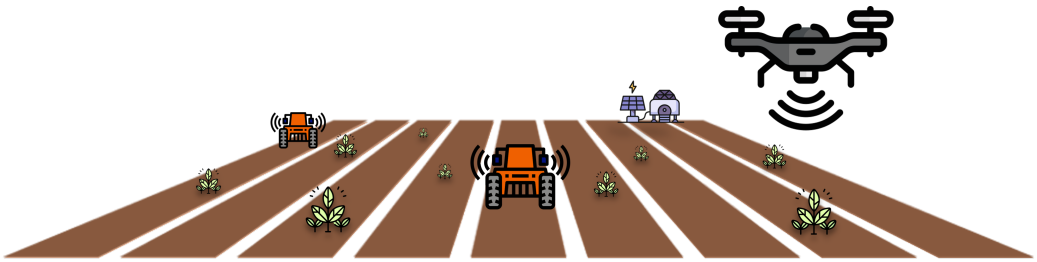


Figure 3.1: Running scenario

3.2 Robots Configuration

Robots participating in the running scenario run the ROS framework and expose several high-level functionalities that ease the development of their behavior.

The implementation of the **drone** exploits the *tello-ros2*[1] model that exposes the following topics for implementing the running scenario. The data published by the drone and useful for continuous monitoring are the */battery* topic for drone battery status, the */odom* topic for retrieving drone position, orientation, and speed, and */range* topic that publishes the distance of the drone from the ground. Differently, other topics enable the control of the drone. Specifically */takeoff* and */land* trigger the taking off and landing respectively, and */cmd_vel* controls horizontal the movement of the drone

The implementation of the **tractor** model exposes the following topics useful for implementing the running scenario. The data published by the tractor and useful for continuous monitoring are the */battery* topic for battery status, the */odom* topic for retrieving current position, orientation, and speed, */laser* topic that publishes data read by a 360° lidar sensor, and */range* topic that publishes data read by the ultrasonic sensor. Differently, other topics enable the control of the tractor. Specifically */cmd_vel* controls the movement of the tractor, while */blade_force* controls the activation of the blade enabling weed removal. Moreover, tractor functionalities are enriched with autonomous navigation capabilities.

The ROS architecture of the running scenario is illustrated in Figure Figure 3.2. Here, nodes are represented by rectangles labeled as s, a, or c to denote whether they function as sensor, actuator, or controller nodes, respectively. Whereas topics flowing across nodes are represented by dotted rectangles. Specifically, sensor nodes manage robots' low-level functionalities to retrieve perceived data, such as positioning data. The actuator nodes define the operation of robots' physical functionalities, such as the activation of propellers. Controller nodes compute the information received from the sensor nodes to trigger the actuator nodes, such as autonomous navigation, or manage the communication between robots, such as weed handling. Notably, while the functionalities of sensor and actuator nodes are exposed by robots' models, the operations of the controller nodes must be defined by an MRS developer.

[1] https://github.com/tentone/tello-ros2

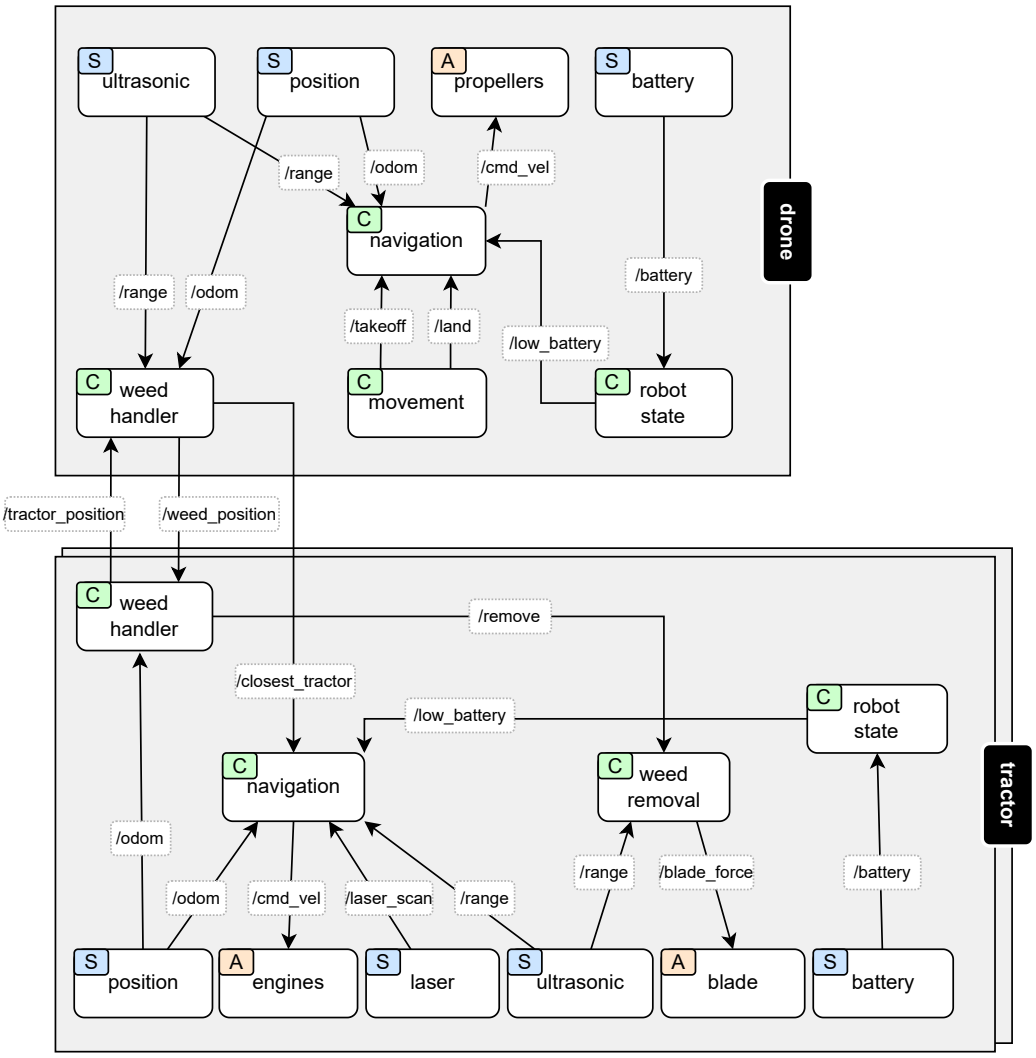


Figure 3.2: ROS architecture of the running scenario

Part II

Top-Down Approach

CHAPTER 4

The FAME Framework

Following a top-down approach, this chapter presents the usage of BPMN collaboration diagrams to enable the modeling and enactment of an MRS mission. The proposed solution is the FAME (BPMN-driven FrAmework for Multi-robot systEms development) framework which relies on a model-driven approach and comprises three phases enabling an MRS designer to develop a BPMN-driven MRS. In particular, the chapter introduces the phases composing the framework, from the system behavior modeling to the direct enactment of the process models inside each robot. After that, an additional module to enhance multi-robot communication is presented. Finally, a comparison with already existing model-driven approaches for robotic systems is provided.

4.1 The Framework

This section presents the FAME framework development phases defined for developing a BPMN-driven MRS. Figure 4.1 depicts the framework and highlights the supported development phases: modeling, configuration, and enactment.

The **modeling phase** corresponds to the first step in developing an MRS using the FAME framework. It disciplines the use of BPMN for the definition of collaboration diagrams to represent the behaviors and interactions of an MRS at a high level of abstraction. The **configuration phase** enriches the modeled collaboration with all the information needed for its execution. It

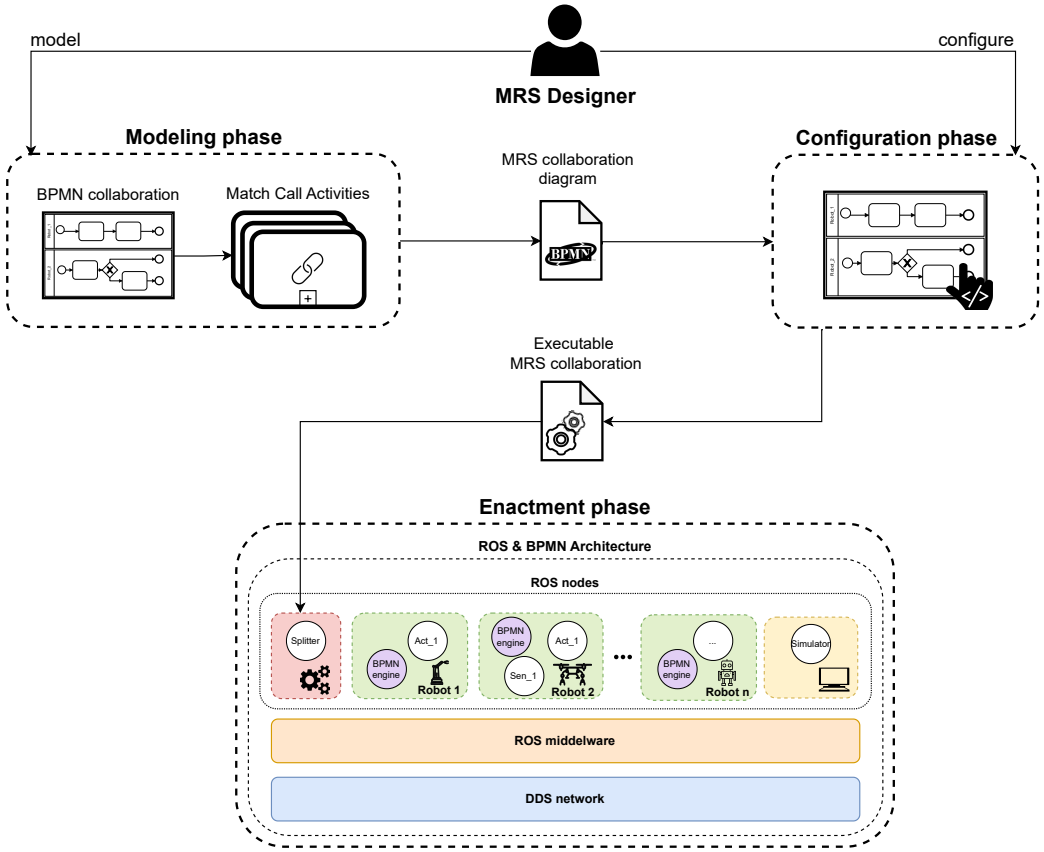


Figure 4.1: The FAME framework

produces as output an executable MRS collaboration diagram used in the following phase to command each robotic behavior, thus guaranteeing a distributed execution of the MRS. The last phase is the **enactment phase** which consists of the execution of the BPMN collaboration directly on each involved robot, without the need for any direct translation into code. This is made possible through BPMN engines, deployed in each robot, that enact only the process associated with each considered robot. As a result of this phase, the framework enables a distributed BPMN-driven MRS mission execution.

4.1.1 Modeling

To enable the modeling of a robotic system via BPMN collaboration diagrams, a set of guidelines has been provided to guide an MRS designer in the modeling of the desired system mission [10]. Initially, a subset of BPMN elements suitable for describing robotic missions is identified. The subset of BPMN elements utilized in this approach is illustrated in Figure 4.2. The selection of these elements resulted from extensive discussions aimed at choos-

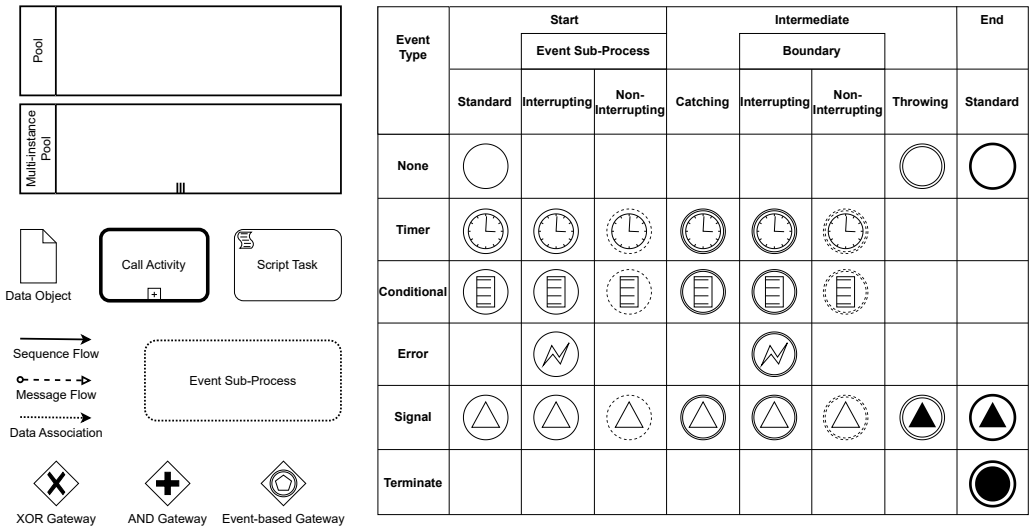


Figure 4.2: Selected BPMN elements

ing the most suitable ones from a list of over 85 elements, aligning them with the requirements of MRSs. The decision-making process was influenced both by considerations driven by the modeling activities conducted in various application scenarios and by considerations that emerged from experiments involving ROS implementations through the Gazebo simulator.

The BPMN elements subset selection is matched with a list of guidelines driving their disciplined use to compose a collaboration diagram that specifies the actions and interactions of each robot in an MRS.

G1. **Robots as pools.** Robots involved in an MRS are abstracted by pools, representing the participants in the collaboration.

G1.1 **Heterogeneous robots as single-instance pools.** Heterogeneous robots, i.e., robots of a different kind or robots with different missions, are abstracted by single-instance pools.

G1.2 **Homogeneous robots as multi-instance pools.** Homogeneous robots, i.e., robots of the same kind with the same mission, are abstracted by multi-instance pools. This simplifies the resulting diagram, as a multi-instance pool represents many robots in terms of several instantiations of the same process, avoiding repeating the same robot process into different pools.

G2. **Mission as a process.** The mission to be performed by a robot is abstracted by a process diagram within the pool of the considered robot. The process diagram expresses the control flow of the mission.

G2.1 **Actions as activities.** A robot mission is mainly made up of a set of actions; these actions are abstracted by activities within the mission process diagram. Based on the complexity of the action, an activity in the model can be either a call activity or a script task.

G2.1.1 **Complex actions as call activities.** Complex actions, e.g., navigation, perception, and control, that can be decomposed into several steps and/or reuse already modeled procedures, are abstracted by call activities. Indeed, a call activity can be used for referencing another process diagram or other existing activities. This enables the modularisation of diagrams and reduces their size, speeding up the modeling of the MRS through the reuse of already modeled behaviors.

G2.1.2 **Simple actions as script tasks.** Simple actions, which do not require any further decomposition, are abstracted by script tasks. This element refers to a piece of code[1] to be executed by the considered robot.

G2.2 **Event handlers as event sub-processes.** Procedures handling an event (such as the expiration of a timer, the satisfaction of a condition, the occurrence of an error, or the reception of a signal) are abstracted by event sub-processes. Indeed, this element triggers a handling process concurrent to the main process describing the robot mission. Based on the event type, the main process can be interrupted or not.

G2.3 **Concurrent behaviors via AND gateways.** Concurrent behaviors in a robot mission are rendered by means of AND split gateways.

G2.4 **Conditional choices as XOR gateways**. Conditional choices in a robot mission are rendered by XOR split gateways.

G2.5 **Event-based choices as event-based gateways.** Choices driven by events in a robot mission are abstracted using event-based gateways.

G2.6 **Missions activation as start events.** The beginning of a mission is abstracted by a start event. In more detail, a none start event fires immediately the robot mission. The other event types activate the mission when specific circumstances occur.

G2.7 **Missions shutdown as end events**. The end of a mission is abstracted by an end event. The none end event stops only the

[1]The code can be of any programming language supported by the robot's software.

incoming execution flow, while the terminate end event stops any execution flow still active[2].

G3. **Communication via signal events**. Intra- and inter-robot communications, even in the presence of a payload, are abstracted by signal events. The sending of a message corresponds to a throwing signal event, and the receiving of a message is modeled by a catching signal event. The correlation between one or more senders and one or more receivers is established through the event name.

G4. **Execution errors via error events**. Errors that can occur during the execution of the robotic mission are abstracted using error events. These events enable the designer to consider and react to potential errors that may arise during execution.

G5. **Delays and timers via timer events**. The evolution of the mission, driven by time-based data, is abstracted with timer events. Timer events introduce a time-based dimension to the execution, enabling the introduction of a time delay within the process.

G6. **Conditions via conditional events**. Boolean conditions that need to be evaluated during the execution are abstracted with conditional events. Conditional events are related to certain conditions being met or specific data values being observed.

G7. **Data as data objects**. The data used during the execution of the robot's mission are abstracted by data objects. They provide storage in which activities and signal events can read or write information. The model should explicitly represent in terms of data objects, only the information used to drive the decision-making and the data exchange in the mission execution. Of course, at the implementation level, other data will be required. However, since they are confined within low-level pieces of code and do not play any role at the abstraction level of the model, they are omitted. This permits reducing the complexity of the diagram.

The modeling phase also includes the possibility to access and leverage pre-defined and reusable processes, i.e., building blocks. Indeed, following the BPMN standard, a call activity serves as a reference to another process defined in a different BPMN diagram. This element facilitates the structuring of models, particularly those that are large and complex, in terms of decoupled, reusable processes. This element enables the modularity of

[2]Notably, more than one execution flow in a model can be active due to an AND gateway (see G2.3).

a process, breaking down intricate processes into manageable, independent components. These modular building blocks offer an advantage in terms of reusability. Indeed, once created, they become versatile tools that can be easily referred to in various processes. This not only saves time but also ensures consistency and reliability among MRS designers. The efficiency of this approach lies in its capacity to simplify processes, minimize redundancy, and reduce the possibility of errors. By integrating the concept of building blocks, the FAME framework keeps consistency with the characteristics of modularity and reusability of the ROS.

4.1.2 Configuration

The output of the modeling phase is a collaboration diagram that can be executed through engines that implement the execution semantics of BPMN. Such engines reproduce the marking evolution of the model. However, to create a collaboration diagram compliant with the ROS framework, an MRS designer needs to enrich the diagram with additional information. This information is mapped in each configured BPMN element as *extension elements*. Indeed, the extension element is part of the BPMN metamodel [68, 8.2.3] and is designed to store additional attributes while ensuring compliance with the standard. This enables the integration of domain-specific XML tags inside the collaboration diagram.

The elements involved in the configuration phase are data objects, events, script tasks, and XOR gateway, and are extended to include the following information.

- **Data Objects**. As prescribed in guideline G7, data objects are containers for information used by the process elements. Therefore, the designer has to use such elements to instantiate the necessary variable names. Variables can be associated with predefined values or with dynamically updated values in the form of $\${variable\_id}$. Other process elements can refer to a variable value using the variable name.

- **Signal Events**. In ROS, the communication, i.e., the publication or subscription of messages over a topic, is specified using: a topic name, a type, and a payload. Concerning the topic name, it is inferred by looking at the signal event name defined by the designer during the modeling phase. While the types of the message (e.g., string, float, bool) and its payload need to be manually selected.

- **Error Events**. Similarly to signal catch events, an error event initializes a ROS subscription. The topic name is inferred by looking at the

error event name, whereas the type of the message is automatically set to the Empty message type[3].

- **Timer Events**. The designer needs to incorporate in a timer event the definition that specifies the delay to be applied to the execution. Consequently, a timer includes the specification that instructs the flow to wait for the desired amount of time. As prescribed by the BPMN standard [68, p.274], the configuration of a timer event can be based on the time date (e.g., *2023-10-01T12:00:00Z*: trigger on 1 October 2023), time duration (e.g., *PT1H30M*: wait 1 hour and 30 minutes), and time cycle (e.g., *R5/PT10S*: repeat every 10 seconds, up to five times).

- **Conditional Events**. The configuration of a conditional event outlines the specific conditions managing its activation. Whenever the condition definition is assessed as true, the conditional event is consumed. Therefore, the designer must add the conditional expression evaluated during the execution. The condition expression is in the form of $\langle expression \rangle$ (e.g., $processVariable > 0$).

- **Script Tasks**. Depending on the desired outcome, the designer has to include in each script task the lines of code to be executed at runtime by the corresponding robot. Therefore, a task contains the code that makes the robot perform the desired computation. It is worth noting that, although the integration of handwritten code can complicate the system design and may lead to some errors, this is significantly less impactful than developing the entire MRS behavior from scratch.

- **XOR Conditions**. The sequence flows leaving the XOR gateway should be configured to guide the control flow along a choice. Specifically, the designer must add a conditional expression evaluated during the execution. The condition is in the form of $next(null, \langle expression \rangle)$.

Once these configurations have been done, the BPMN collaboration can be deployed and the MRS executed.

4.1.3 Enactment

The enactment phase consists of the deployment of the BPMN collaboration in the robotic system and of its execution on real or simulated systems. The first step in the deployment is the splitting of the BPMN collaboration into several process models, one for each robot. This is because the framework is defined to realize a distributed execution of the MRS. The *splitter* component extracts the pools from the collaboration diagram and associates them with

[3]https://docs.ros2.org/foxy/api/std_msgs/msg/Empty.html

the corresponding robot. Notably, to bind a pool to a robot, the same name is utilized for the robot namespace in ROS and the pool in the diagram. In the case of a multiple instance pool, the splitter component searches for namespaces starting with the pool name, followed by the symbol "_", and ending with some digits. This permits automatically determining the number of robots corresponding to the multi-instance pool, in this way, the same model can be used in scenarios with a different number of robots.

Following the distributed architecture of ROS, where a robot is a collection of nodes, a node implementing a *BPMN engine* has been incorporated into the robots. Each of these nodes receives the pool to execute from the splitter component and exploits a BPMN execution engine on top of the robotic environment. Thus, the engines enable the robots to perform computations and communicate with each other. Since the splitter deploys every single process separately, the designer can change at run-time the behavior of all robots or a subset of them, even of those that have been previously instantiated with a multi-instance pool. The enactment of the modeled collaboration can be done in the same way in real scenarios or in the Gazebo simulator, thanks to the ROS infrastructure which considers Gazebo as a node in the DDS network. Therefore, the MRS developer can decide to execute the collaboration directly on the real robots or simulate the scenario to spot potential issues.

At the startup of the MRS system, each engine fires the execution of its process model following the BPMN execution semantics. Every time an engine triggers a BPMN element that involves the performing of an action by the robot (i.e., a script task or a signal event), the code embedded in the element is evaluated at run-time by the robotic framework. The other elements instead are entirely interpreted by the engine; for example, when an engine executes a XOR split gateway, it drives the control flow toward the sequence flow with the true condition without sending any command to the ROS framework.

In the case a process or an activity is interrupted during its action, like when a terminate end event is fired or there is the need to deploy a new process at run-time, the engine forces the termination of any action the robot is performing. This, of course, may provoke risky situations. Therefore, the MRS designer must include in the model an event sub-process that brings the robot to a safe state before the engine kills the execution. As an example, a safe state sub-process may include a set of activities executed to bring the robot back to the base station.

Notably, the FAME framework identifies a minimal subset of elements necessary for describing MRS scenarios. However, this does not imply that other elements in the BPMN standard can not be useful in other scenarios. The benefit of using an engine allows for the addition of elements to the

model that are not indicated in the guidelines, as long as they are supported by the engine. In contrast, a model-to-code approach would require mapping new elements to the corresponding code whenever a new element is used.

4.2 Communication Compatibility

In addition to the proper development of the behavioral model of robots, communication is one of the main features that enable cooperation in an MRS system. Indeed, the interconnections created inside an MRS generate a dynamic and heterogeneous system where robots can share information, coordinate actions, and provide intelligent services. Therefore, efficient and reliable communication within an MRS is crucial to achieving the full potential of these systems. Robots' communication should be guaranteed by integrating QoS policies that allow the specification of the communication needs, supporting time-aware, context-aware, and content-aware communications [102], thus impacting the overall system performance. However, while QoS policy integration enhances communication, ensuring effective communication requires devices to integrate these policies in a compatible manner [72].

This section tackles the integration of QoS policies in the FAME framework to ensure communication compatibility among the system's devices, thus obtaining a BPMN-driven approach that can handle the complexity of selecting and guaranteeing compatible QoS policies.

4.2.1 QoS Requirements

In the context of distributed applications, QoS policies are used for controlling data distribution. These policies refer to various communication parameters, such as data availability, resource usage, reliability, and timing [69]. During the development of ROS-based systems, QoS policies can be configured by associating within each publisher and subscriber a QoS profile, i.e., a set of predefined QoS policies, or by manually choosing them. However, the manual setup implies the developer must properly configure QoS policies to ensure compatibility. More in detail, the communication between ROS publishers and subscribers is established only if all the QoS policies are compatible, following the compatibility rules defined in the ROS documentation[4]. Notably, multiple subscriptions can be connected to a single publisher simultaneously even if their policies are different. An example of compatibility between QoS policies for the reliability parameter is shown in Table 4.1.

[4]https://docs.ros.org/en/rolling/About-Quality-of-Service-Settings.html

Publisher	Subscriber	Compatible
Best effort	Best effort	Yes
Best effort	Reliable	No
Reliable	Best effort	Yes
Reliable	Reliable	Yes

Table 4.1: Policy compatibility for the reliability parameter

4.2.2 QoS Compatibility Approach

The proposed approach, depicted in Figure 4.3, is directly tied with the FAME configuration phase. Indeed, thanks to the modular structure of FAME, additional modules that extend framework functionalities can be easily integrated. During the *configuration phase* the MRS designer configures the signal events (see Section 4.1.2) by relating publishers and subscribers. In this phase, the designer can select and specify the set of QoS policies that guide the communication.

The configuration of the policies triggers the compatibility checker. Indeed, the **C-QoS module** ensures that the chosen policies are compatible with each other. This phase integrates a checker capable of assessing communication compatibility. It ensures that for each matched publisher and subscriber, the corresponding policies comply with the standards set by ROS for effective communication. The output of this phase is feedback returned to the MRS designer stating if the checking has been successful or not. Specifically, if incompatible policies have been detected, the designer receives feedback highlighting the need to fix the communication issues in the model.

This approach enhances the communication among the various robots, contributing to the overall performance of the system. Additionally, the module implementing the approach enables the configuration of the system communication requirements, the identification of communication incompatibilities, and ensures feedback to the MRS designer during the system's design phase.

4.3 Related Works

In the literature, many works face the model-driven development of robotic systems considering the different stages that lead from systems modeling to execution in real contexts. In this regard, these works show different levels of maturity and provide contributions mainly focused on: (*i*) modeling robotics missions, and enacting the obtained models via a central control unit, (*ii*) deploying and enacting the models inside the robots, and (*iii*) developing model-to-code translations leading to the execution.

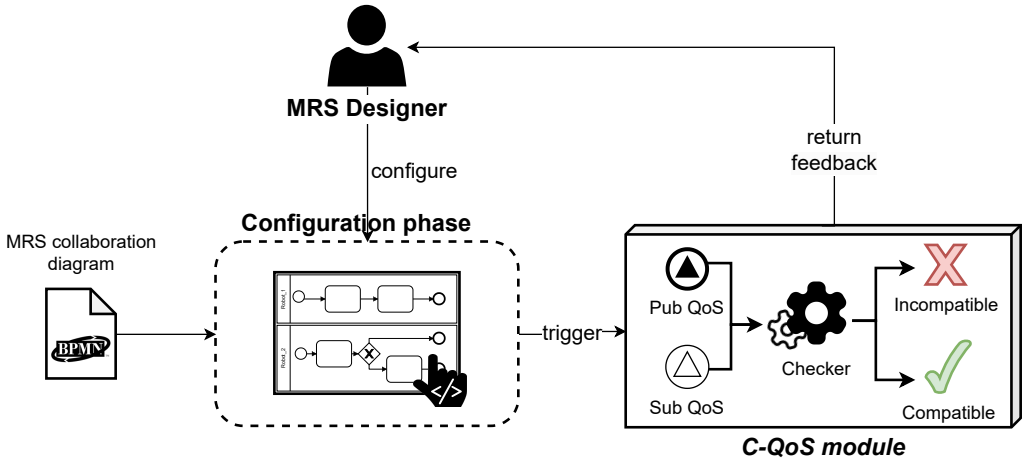


Figure 4.3: QoS compatibility check approach

Considering the approaches focused on the model execution managed by a central control unit, in [11] the authors describe a tool for the mission specification for a team of multicopters and the generation of a detailed flight plan, using a DSL that is translated into an intermediate language representing the basic actions of an aerial vehicle. These actions are combined to define the drones' mission using a finite state transition system where each transition corresponds to an action. Exploiting the same tool, in [17] the authors define a set of DSLs to specify the civilian missions for unmanned aerial vehicles. The mission specification and enactment are done through a web-based graphical interface, that communicates with some ROS-based controllers sending commands to the vehicles. Notably, using custom DSLs may require time to learn them. This effort could be mitigated using a standard modeling language, like the BPMN notation used in FaMe. Indeed, BPMN is a well-accepted standard applied to fit a wide variety of process modeling purposes [50], thus resulting familiar to many users. Its notation is easy to understand and learn [53], especially considering that FaMe considers only a subset of elements. On the other hand, the modeling guidelines support the application of BPMN for modeling the mission of an MRS shift BPMN from a general-purpose notation to a domain-specific one. Even if this means that the designers should learn the guidelines, the effort to learn them is not high. Indeed, they do not introduce any new element to the notation and do not associate different semantics with the BPMN elements. The guidelines just provide disciplined arrangements of BPMN elements, linking them to the corresponding MRS concepts. In [105] the authors propose a laboratory survey to enable the co-creation of DSL solutions for robots. It aims to reduce the complexity and the necessary technological background that is required to develop a robotic application. The proposed tool running

externally from the robot can use BPMN models to abstract the implementation of one single robot, and to generate and execute the related mission. Another integration of the BPMN standard with an autonomous robot is presented in [73]. The authors develop and automatize a warehouse process by implementing a human-robot cooperation system managed by a web interface able to control the entire process in real-time. These approaches exploit an external device to enact the process and send commands to the robot, thus making this central device a single point of failure for the robotic system. The FAME approach, instead, is fully distributed; hence, it is more suitable in a multi-robot context. Nevertheless, in cooperative scenarios, the behavior of the robots can be interrelated. Thus, it may happen that if a robot fails, the entire mission can not be carried on. In those cases, the MRS designer should prevent this situation by modeling an alternative behavior that reacts to a robot failure.

Moving on to the interpretation of the model executed directly by a robot, the authors in [27] describe the TRACE tool to tailor BPMN to the robotics domain to model a sequence of robotic activities. This approach aims to understand what will happen after an unplanned event and check if it will compromise the mission. The authors applied their proposal to a single robot equipped with ROS1, which can execute its tasks and autonomously react to unexpected events. In the same direction, in [54], the authors present an application of the TRACE tool to a multi-robot scenario. The mission is specified in a BPMN file, uploaded inside all the vehicles, and each block of the model corresponds to a robot's behavior. The system uses the ROS1 framework with an open-source package that allows custom message-passing among multiple master nodes. Such message passing creates a decentralized communication among ROS nodes implementing an ad-hoc communication mechanism that, unlike FAME, fails to exploit the distribution brought by the powerful DDS standard. Moreover, since the robots share the same mission, but cannot communicate with each other, the authors highlight that a human operator must be involved in assigning robots to different areas. Exploiting the FAME tool, it would be easier to add an automatic coordinator to manage area coverage. At the same time, robots could handle this problem independently by leveraging distributed DDS communication.

Other approaches develop model-to-code translations for system execution. In [56] the authors provide an automatic method to verify the task-level control in autonomous robotic systems using Petri Nets as input language for a model-checking tool. A DSL is used to define the robot's behavior and all the constraints that must be satisfied. Each global action is then translated into a Petri Net and verified to check system fairness and efficiency. The approach uses a single-vehicle scenario, equipped with ROS1. In [35] the authors present a methodology for the description of a multi-robot mission

and the related application to a ROS-based system, using Hierarchical Petri Nets. The proposal is extended in [36], in which a 6-layer Petri Net is used to describe the activity of the robotic system at a different level of abstraction. The resulting model is used to verify the system's safety and liveness, and finally to generate ROS-executable code for controllers. In [44] the authors present a model-driven approach to support the systematic engineering of MRSs thus facilitating the definition, implementation, and analysis of these systems. The proposed framework is based on the ATLAS DSL, for the team structure and mission objectives specification, and on a code generator engine. The latter enables communication and coordination among system components, creates an interface for direct interaction with a robotic simulator, and produces the related configuration files. Finally, in [65] the authors propose a platform to develop multiple robots using a user-friendly model editor. The editor is based on ROS1 so that users can graphically represent the system mission, automatically translated into an executable Python source file. The works described above are based on translations of models into code, which limits dynamic changes to the mission of an MRS. On the other hand, FAME relies on the interpretation of models by a BPMN engine, which is more suitable to support change at run-time to the MRS mission.

Paper	Purpose[1]	Modeling Language	ROS Version	#Robot	Architecture
[11]	M	MML	ROS1	Multi	Centralized
[17]	M, E	MML	ROS1	Multi	Centralized
[105]	M, E	BPMN	ROS1	Single	Centralized
[73]	M, E	BPMN	ROS1	Single	Centralized
[27]	M	BPMN	ROS1	Single	Centralized
[54]	M, E	BPMN	ROS1	Multi	Decentralized
[56]	M, E	TDL	ROS1	Single	Centralized
[35] [36]	M, E	Petri Net	ROS1	Multi	Centralized
[44]	M, E	ATLAS	ROS1	Multi	Centralized
[65]	M, E	Domain-Specific	ROS1	Single	Centralized
FAME	**M, E**	**BPMN**	**ROS2**	**Multi**	**Distributed**

Table 4.2: Features comparison of the related works
(1) M: Modeling E: Execution

The analysis of the literature review has been summarized in Table 4.2. The comparison shows that all the approaches exploit version 1 of ROS, therefore the communication is managed by a master node. This creates a centralized architecture with a single point of failure, making it unsuitable for the development of MRS applications. Indeed, many of these works do not consider MRSs at all. Furthermore, many approaches execute the system by exploiting the model-to-code translation instead of the execution of the model. This implies that the mission of the robots has to be uploaded offline and any modification of the model leads to the need to interrupt the system and load the new code generated by the translation. All these shortcomings are considered in the design of the FAME framework. Specifically,

distributed communication among robots is achieved with the adoption of the ROS2 middleware, and a direct enactment of the model is obtained with the deployment of a BPMN engine inside each robot.

CHAPTER 5

FAME at Work

This chapter presents the tools implemented to support the FAME framework phases. After that, the application of the FAME phases to the running scenario of Chapter 3 is presented. Additionally, FAME is evaluated in a physical environment to analyze framework performances in relation to direct code execution.

5.1 FAME Implementation

The FAME framework is supported by a toolchain composed of software and artifacts, developed for experimenting with the BPMN-driven development approach for ROS-based MRSs. The modeling, configuration, and enactment phases can be fully automated through the use of the provided toolchain.

Specifically, two packages support the FAME framework. The **FAME-modeler** is a web application facilitating the modeling and configuration phases, whereas **FAME-ROS** is a ROS package that contains the scripts for deploying the splitter node and the engine node. This design enables the reuse and integration of these packages with different robots and scenarios. Communication between packages is enabled by the *ros2-web-bridge*[1], which provides a JSON interface to ROS, allowing any client to publish or subscribe to ROS topics. The FAME toolchain is depicted in Figure 5.1 and the source code of software tools is available on the project page[2].

[1] https://github.com/RobotWebTools/rosbridge_suite
[2] https://pros.unicam.it/fame

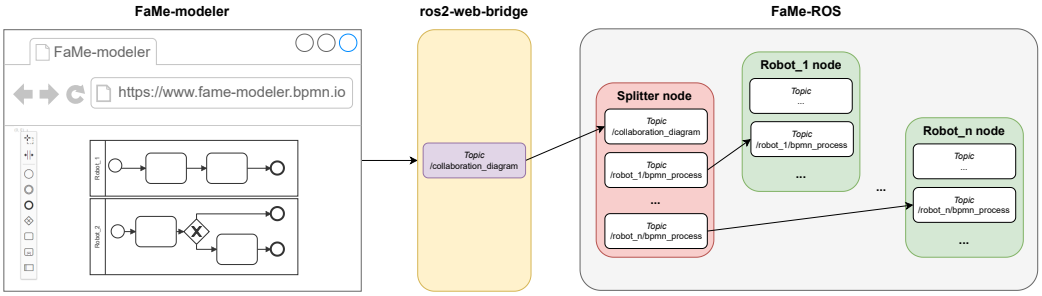


Figure 5.1: The FAME toolchain

5.1.1 FAME-modeler

For the modeling and configuration phases, the designer can exploit a customized BPMN modeler built upon the *bpmn-js* toolkit[3]. Indeed, the FAME-modeler interface, depicted in Figure 5.2, embeds the BPMN modeling environment and the following functionalities aimed at supporting the MRS designer.

- The **palette** contains the BPMN graphical elements used to design a process model.
- The **canvas** is where BPMN elements are placed to form the process model.
- The **property panel** provides a space for viewing and modifying properties or attributes of selected elements, i.e., for configuring them.
- The **C-QoS module** checkbox indicates whether or not activating the QoS compatibility checker.
- The **ROS connection** button initializes the connection with the ROS network.

Notably, the FAME-modeler exploits the BPMN extension element tag to include additional data and message exchange information in the .bpmn file. This approach ensures compatibility with other modeling tools while allowing FAME to enrich the BPMN representation with specific details. In this regard, the BPMN property panel plays a key role in enriching the model with this information. Therefore, the FAME-modeler implements all the functionalities of a BPMN modeler while supporting the configuration of the elements presented in Section 4.1.2 through an extended and customized property panel.

[3]https://bpmn.io/toolkit/bpmn-js

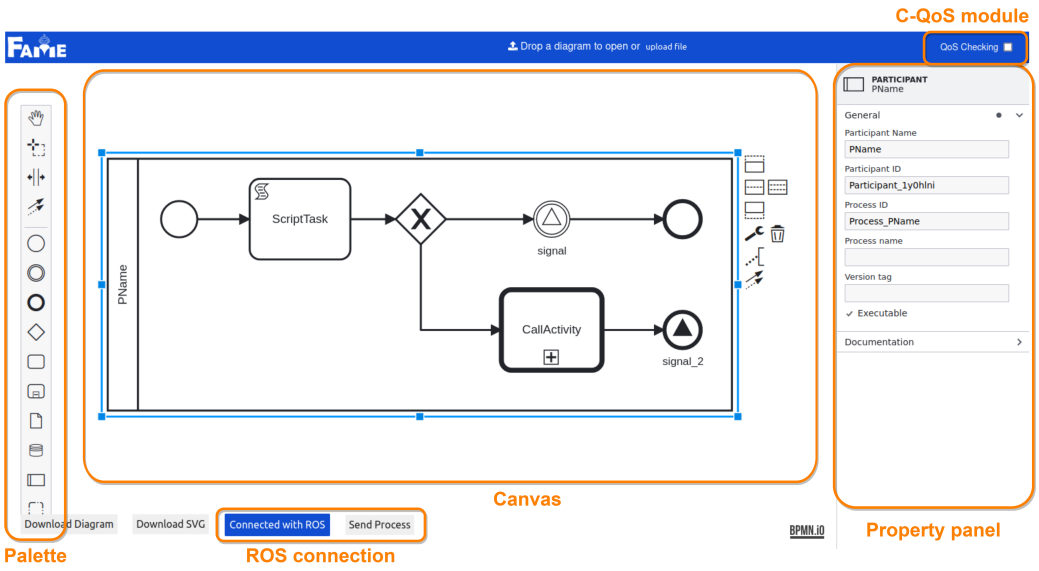


Figure 5.2: FAME-modeler property panel

Property panel. The configuration of the collaboration diagram compliant with the FAME framework is facilitated by the extension of the property panel for the elements that require specific configurations. More in detail, ***data objects*** property panel provides the flexibility to select an arbitrary number of variables in the format *name-value*. If the value is undefined, i.e., the value is generated by the execution of one element in the process, the system automatically assigns the variable value equal to *${name}*. The extension of a data object is in the form of Listing 5.1.

```
<bpmn:dataObjectReference id="DataObjectReference_17784gh"
    dataObjectRef="DataObject_0wt07gj">
    <bpmn:extensionElements>
        <data:parameters>
          <data:parameter name="variable_1"
              value="${variable_1}" />
          <data:parameter name="variable_2" value="50" />
        </data:parameters>
    </bpmn:extensionElements>
</bpmn:dataObjectReference>
```

Listing 5.1: Data object schema

Signal events definition is facilitated by the incorporation of a *ROS tab* in the property panel that enables the selection of predefined ROS message types. Additionally, for throwing signals, i.e., ROS publishers, the tab is enriched with auto-generated payload fields associated with the chosen message type. For instance, if the message is of type *Vector3*, three fields, i.e., x, y,

and z, are added to the property panel and can be associated with predefined values or dynamic variables. A throwing signal XML schema is in the form of Listing 5.2, and a catching signal is in the form of Listing 5.3.

```
<bpmn:intermediateThrowEvent id="Event_00tbq5g"
    name="topic_name">
    <bpmn:extensionElements>
        <ros:message type="geometry_msgs/msg/Vector3" />
        <ros:payload>
          <ros:parameter name="x" value="5.0" />
          <ros:parameter name="y" value="2.1" />
          <ros:parameter name="z" value="0.0" />
        </ros:payload>
    </bpmn:extensionElements>
</bpmn:intermediateThrowEvent>
```

Listing 5.2: Signal throw schema

```
<bpmn:intermediateCatchEvent id="Event_1orgyuv"
    name="topic_name">
    <bpmn:extensionElements>
        <ros:message type="geometry_msgs/msg/Vector3" />
    </bpmn:extensionElements>
</bpmn:intermediateCatchEvent>
```

Listing 5.3: Signal catch schema

For ***script tasks*** and ***conditional choice***, the property panel does not need additional extensions. However, to facilitate the designer, the property panel defines *JavaScript* as the predefined language for script tasks, while allowing language customization to meet designer requirements. Whereas, the condition configuration is supported by the automatic generation of the needed structure. A script task is in the form of Listing 5.4 and a condition is in the form of Listing 5.5.

```
<bpmn:scriptTask id="Activity_0cwtqkh" name="Script"
    scriptFormat="JavaScript">
    <bpmn:script>
        var text = "hello!";
        console.log(text);
        next();
    </bpmn:script>
</bpmn:scriptTask>
```

Listing 5.4: Script task schema

```
<bpmn:sequenceFlow id="Flow_1f1eb5r" name="yes"
    sourceRef="Gateway_08xvyud" targetRef="Activity_0cwtqkh">
      <bpmn:conditionExpression
          xsi:type="bpmn:tFormalExpression">
        next(null, variable_2 > 10);
      </bpmn:conditionExpression>
</bpmn:sequenceFlow>
```

Listing 5.5: XOR condition schema

Finally, the property panel for ***call activities*** has been extended to leverage a database with reusable processes and provides the flexibility to create new ones as needed. Indeed, the process referenced by the call activity can be chosen from a database of processes or the designer can define a new process and save it in the call activities database. A call activity is in the form of Listing 5.6.

```
<bpmn:callActivity id="Activity_1rktq6o" name="GoTo"
    calledElement="GoTo">
      <bpmn:extensionElements>
        <callActivity:process process="<bpmn:process
            id='Process_GoTo' isExecutable='true' ..." />
      </bpmn:extensionElements>
</bpmn:callActivity>
```

Listing 5.6: Call activity schema

C-QoS module. This module has been integrated into the framework to ease the specification of communication requirements and to ensure their compatibility. It consists of two main components embedded in the FAME-modeler: a property panel extension and a rule checker. The property panel extension allows the association of a set of QoS policies with each signal element. The policy values selected are stored in the diagram, thus enriching the behavioral model with additional information driving robots' communication. An excerpt of the QoS-enhanced BPMN signal is shown in Listing 5.7.

The rule checker is integrated within the modeler using the *node-rules*[4] library, a rule-engine library that enables real-time control of the inserted QoS policies. Following the ROS-based QoS compatibility specification, a specific set of rules is defined to take as input each paired publisher and subscriber and the respective QoS policies to assess their compatibility. An event-based function monitors changes in signal nodes that can occur either when the communication topic changes or when QoS parameter values are modified. Then, the rule engine takes the changed signal node information

[4]https://github.com/mithunsatheesh/node-rules

as input and assesses whether the publisher policy parameters align with the one required by the corresponding subscriber. Specifically, as prescribed by the ROS framework, the compatibility is determined based on the *request vs offered* model, which allows connections only when the publisher's policies align with the subscriber's policies. Whenever there is a change in the QoS policy of a signal node, the modeler triggers the rule engine mechanism to evaluate the updated parameters.

```
<bpmn:intermediateCatchEvent id="Event_1orgyuv"
    name="topic_name">
    <bpmn:extensionElements>
        <ros:message type="geometry_msgs/msg/Vector3" />
        <qos:policy name="history" value = "keep_last" />
        <qos:policy name="depth" value = "5" />
        <qos:policy name="reliability" value = "reliable" />
        <qos:policy name="durability" value = "transient_local"
            />
        <qos:policy name="deadline" value = "2" />
        <qos:policy name="lifespan" value = "10" />
        <qos:policy name="liveliness" value = "automatic" />
        <qos:policy name="lease_duration" value = "10" />
    </bpmn:extensionElements>
</bpmn:intermediateCatchEvent>
```

Listing 5.7: QoS policy schema

ROS connection. This component facilitates the establishment of a connection with the *ros2-web-bridge*, enabling the integration of the BPMN collaboration within the ROS network. The connector plays a key role in sharing the executable MRS collaboration, allowing it to be published across the ROS network. Indeed, this component creates a ROS publisher over the */collaboration_diagram* topic and can extract the process model in a string format. Therefore, the designer can exploit the facilities of the FAME-modeler to easily establish communication within the ROS network and share the collaboration diagram with other ROS nodes. It is worth noticing, that this component necessitates an active instance of the *ros2-web-bridge* package to function properly.

5.1.2 FAME-ROS

The automation of the enactment phase is supported by the FAME-ROS package implementing the splitter and the BPMN engine as ROS nodes. Firstly, the splitter node implements a submission to the ROS topic named */collaboration_diagram* for receiving from the modeler the collaboration diagram. Subsequently, it can extract from the received diagram the involved

processes and send them in the form of a string over ROS topics named */robot_name/bpmn_process*. Notably, if there are pools in the collaboration marked as *multi-instance pool*, the splitter node scans the ROS network to extract the number of robots' occurrences matching the pool name. Therefore it publishes the related process as many times as the number of identified homogeneous robots.

The BPMN engine node is a customized version of the JavaScript-based *bpmn-engine*[5] library. It is integrated into the ROS architecture, making possible the interpretation of BPMN models configured as prescribed in the FAME approach. The integration of the engine within ROS has been done without any limitation by exploiting the *rclnodejs* client library[6] for interpreting nodes written in JavaScript. Notably, some of the engine features have been extended to support the FAME framework as follows. The initialization of the engine creates a scope of variables, named *environment*, containing the model and the ROS engine node itself. Moreover, the activation of a data object adds the data it stores in the environment. In line with the discussion in Section 4.1.1, communication is handled through signals mapped into ROS topics. To align the engine's functionalities with the ROS communication model, triggering a throw signal creates a ROS publisher characterized by the information retrieved from the model. On the other hand, a catch signal and an error signal result in a subscription to the related ROS topic. The message exchange on ROS topics facilitates communication between signals executed in different engines, by producing the termination of the catch signal when the subscriber receives a message.

5.2 Evaluation in the Running Scenario

This section presents the application of the FAME phases to the running scenario presented in Chapter 3 and executed in the Gazebo simulator. Notably, a detailed system setup and the configured collaboration diagram are illustrated in the online documentation.

5.2.1 Modeling

Starting from the system description, the BPMN collaboration depicted in Figure 5.3 is a possible result of the application of the guidelines proposed in Section 4.1.1.

As prescribed by guideline G1, the scenario is modeled using two pools: a single-instance pool for representing the drone (G1.1), and a multi-instance pool for representing the tractors (G1.2). In turn, these pools contain the

[5]https://github.com/paed01/bpmn-engine
[6]https://github.com/RobotWebTools/rclnodejs

robots' mission specifications in the form of processes (G2). For instance, the process contained in the drone pool depicts the sequence of activities that the robot has to perform, from the taking off to the landing. The other guidelines are then used in the modeling of each process which finally results in a composition of process elements (activities, gateways, and events) connected via sequence flows. As prescribed by G2.1.1, call activities are used for representing complex actions that are specified in external BPMN diagrams, and that can be possibly reused several times in different parts of the collaboration, for instance, the *GoTo* call activity appears four times in the collaboration. Specifically, this call activity refers to the process depicted in Figure 5.4 and is composed of a script task implementing the desired algorithm that implements autonomous navigation, continuously checking for robot position, and when the desired destination is reached, stops the movement.

Differently, simple actions that cannot be further decomposed are rendered as script tasks (G2.1.2), for instance, the script task *Update Closest* performs simple mathematical operations to calculate the tractor closest to the weed. Finally, as indicated in G2.2, an event sub-processes is used for representing the procedure to handle a specific situation. For instance, *End Handler* is an interrupting event sub-process that is performed by the robots whenever a *low_ battery* error or *field_ cleaned* signal is caught. Concerning the gateways, in the tractor's pool, an example of conditional choice is the decision taken by the tractor to determine whether or not to stop its execution, this is rendered as a XOR gateway with two possible outcomes (G2.4). Notably, the label of the XOR gateway is used to improve the readability of the model. As prescribed by the BPMN standard, the conditions of XOR gateways are provided as boolean expressions specified as attributes of the elements during the configuration phase. Whereas, an event-based choice happens when a robot waits for an event, for instance when a tractor waits for a *closest_ tractor* signal. Indeed, if the signal is not received within 30 seconds, the timer event is triggered (G5), routing the execution to another path in the model. This is rendered through an event-based gateway (G2.5) followed by two catching events: the signal and the timer. Regarding the events, guideline G2.6 guides the use of a none start event at the system start-up in the main process of the drone, while a signal start event is used for the tractors, which indeed are enacted only when they receive a *weed_ position* signal. This happens when a signal with the same name is thrown, like for the signal event in the drone pool (G3). Similarly, the *low_ battery* error event has been used to trigger the event sub-process of a robot that spots the end of the battery charge (G4). Additionally, data-driven workflow is defined using conditional events (G6). For instance, the tractor event sub-process handling the low battery, integrates a *battery_ full*

signal to pause the process workflow until the battery is fully charged. In conclusion, as prescribed in G7, the data used along the robot's process are represented through data objects, as for data object *Weed Position* which contains the coordinates of the grass to be removed.

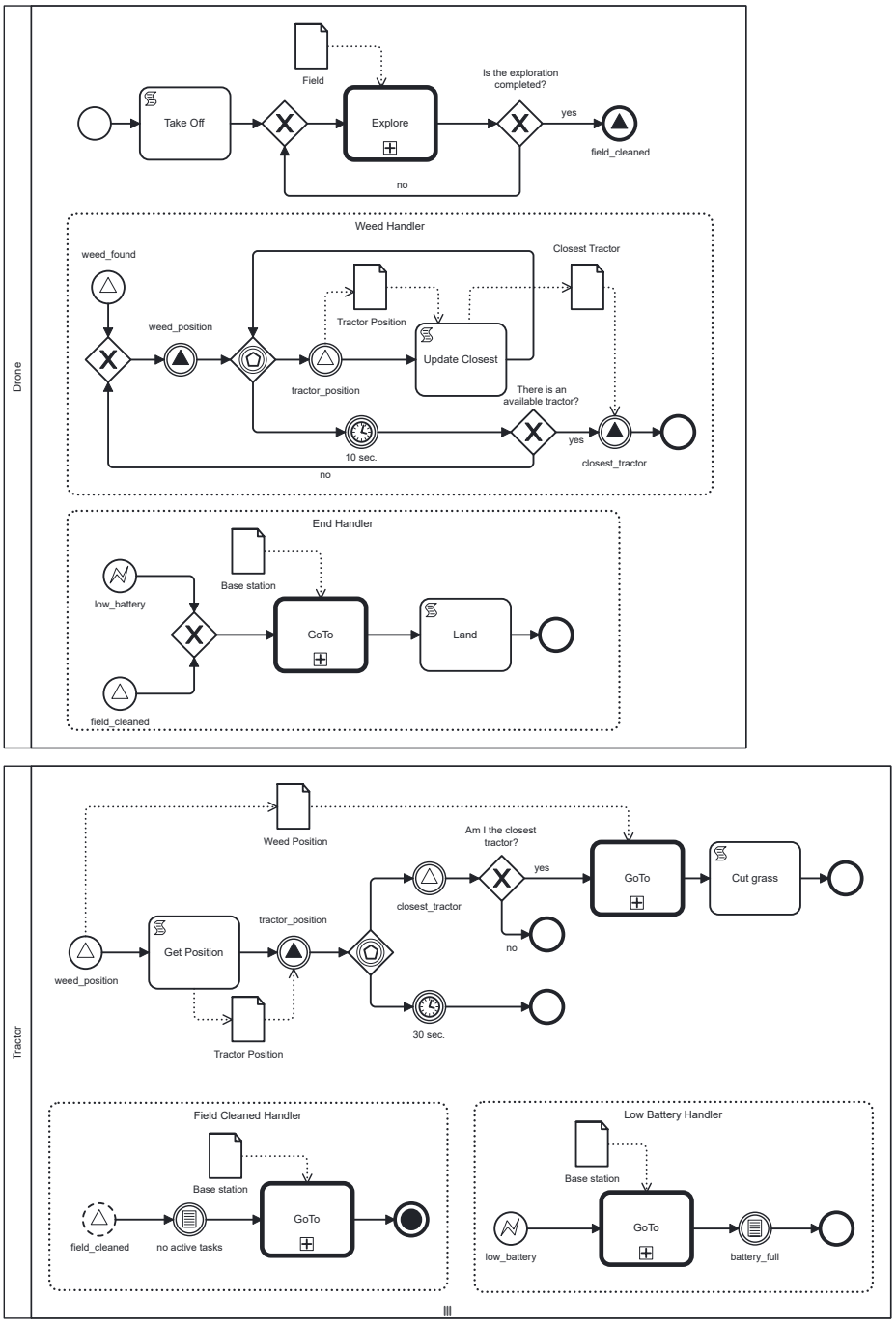


Figure 5.3: Collaboration diagram of the agricultural scenario

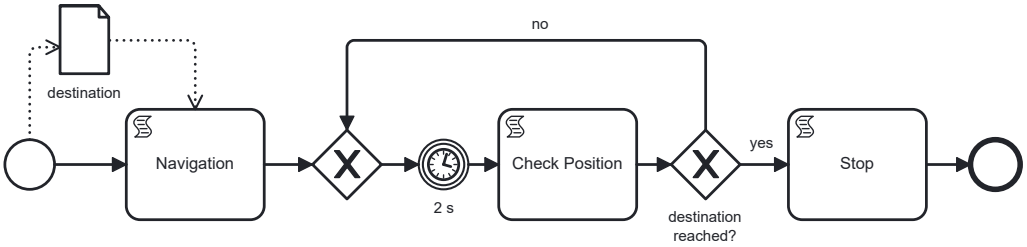


Figure 5.4: *GoTo* call activity

5.2.2 Configuration

The MRS execution described in the previous section lacks a concrete link with the robotic ecosystem. To obtain a BPMN collaboration executable directly by robots, the MRS designer has to provide in the BPMN file additional details. These aspects can be specified by configuring the BPMN directly in the FAME-modeler. As prescribed by the configuration phase (see Section 4.1.2), the elements requiring a configuration are data objects, events, scripts, and conditional choices.

- **Data Objects**. Data objects store information used in the process and can contain predefined values or dynamic ones. Considering the running scenario, the data object *Field* stores the predefined coordinates representing the boundaries of the agricultural field (see Figure 5.5). Differently, the data object *Closest Tractor* is marked as output data, therefore it is automatically associated with the value *${closest_tractor}*. Indeed, this value will be dynamically updated by the script task *Update Closest* (see Figure 5.6).

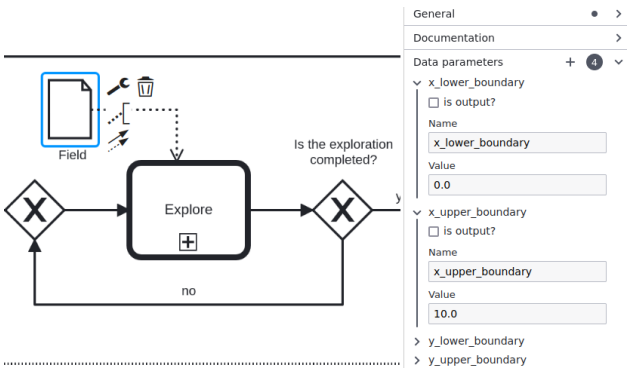


Figure 5.5: *Field* data object configuration

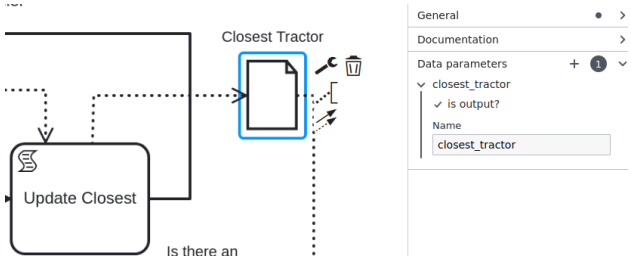


Figure 5.6: *Closest Tractor* data object configuration

- **Signal Events**. Depending on their nature, i.e., throwing or catching, signal events are configured to act respectively as ROS publishers or subscribers. Considering the inter-robot communication via the *closest_tractor* signal, the message throwing is configured by selecting the ROS message type, i.e., string, and the content of the message, i.e., the value stored by the *Closest Tractor* data object (see Figure 5.7). Whereas, the corresponding catching signal is only configured to be able to receive a string message type (see Figure 5.8).

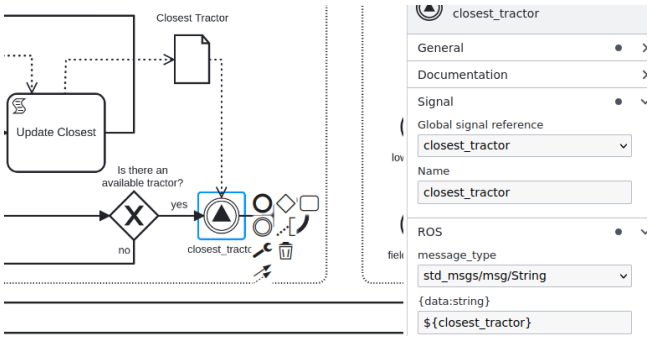


Figure 5.7: *closest_tractor* throwing configuration

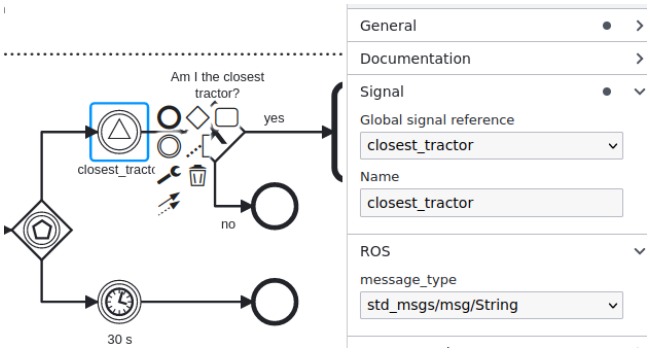


Figure 5.8: *closest_tractor* catching configuration

- **Error Events**. Error events act as ROS subscribers over the topic with the same name as the event. Considering the *low_ battery* error, the configuration of this event only requires the setup of the error event name (see Figure 5.9).

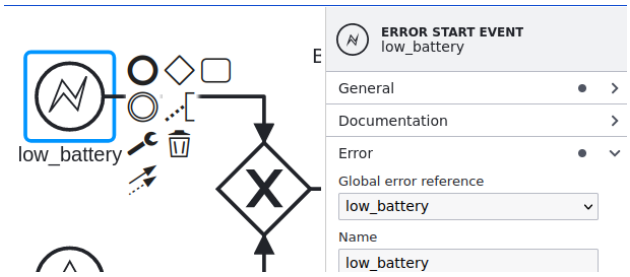


Figure 5.9: *low_ battery* error configuration

- **Timer Events**. Following the BPMN standard, timer events should be configured with the desired amount of waiting time. For instance, the *10 s* timer is configured with a time duration of 10 seconds (see Figure 5.10).

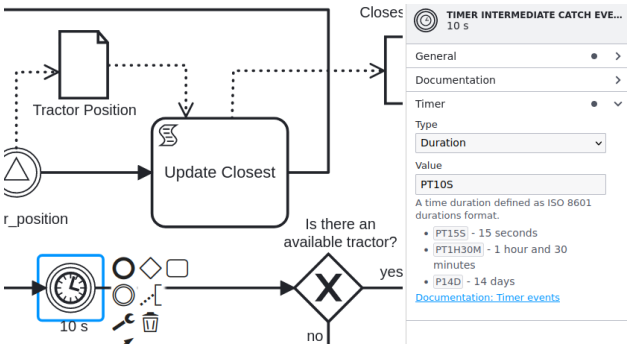


Figure 5.10: *10 s* timer configuration

- **Conditional Events**. The usage of conditional events requires the setup of the expression managing the triggering. Taking as an example the *battery_ full* conditional event, it is configured to evaluate when the battery is fully charged (see Figure 5.11).

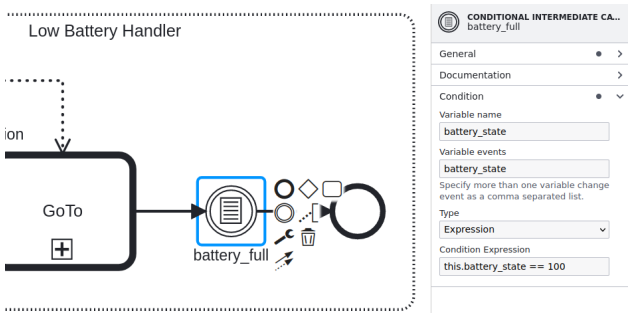


Figure 5.11: *battery_full* conditional event configuration

- **Script Tasks**. A script task contains the code that makes the robot perform the desired computation. As an illustrative example, the *Update Closest* script task is configured to compute if the position of the current tractor is the one closest to the weed. The output of this computation updates the *closest_tractor* variable value (see Figure 5.12).

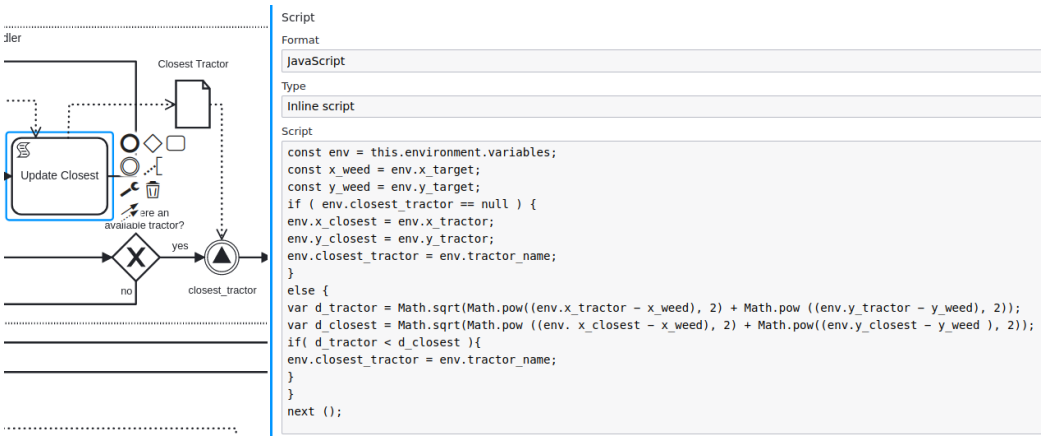


Figure 5.12: *Update Closest* script task configuration

- **XOR Conditions**. The configuration of the sequence flows leaving a XOR gateway guides the control flow along a choice. For instance, when the drone checks the presence of an available tractor for performing the weed removal, the *yes* flow is configured to check if the *closest_tractor* variable has been associated with a non-null value (see Figure 5.13).

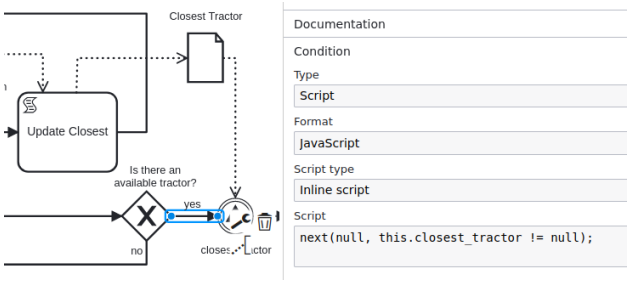


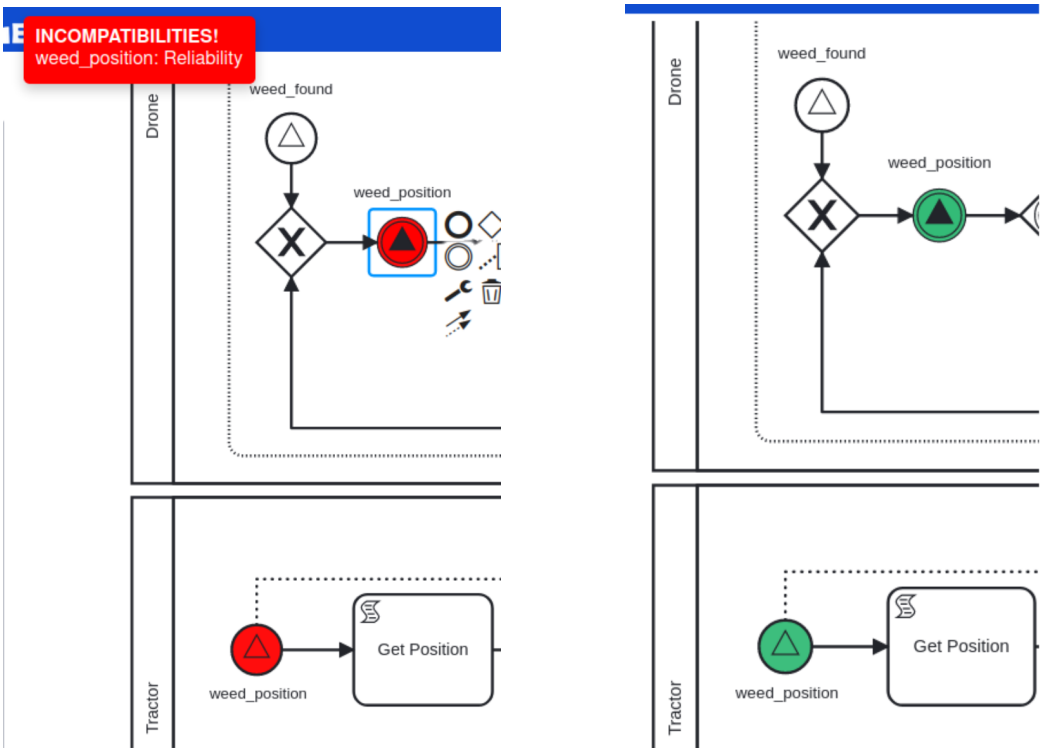
Figure 5.13: *yes* flow configuration for tractor availability condition

C-QoS module. After the configuration of the signals, the MRS designer can assign them a set of QoS policies required by the scenario. Exploiting the C-QoS module, FAME automatically performs the policies compatibility check to ensure communication between publishers and subscribers. For instance, the *weed_position* throwing signal is initially configured with a reliability equal to *Best Effort*, whereas the corresponding catching signal is configured with a reliability equal to *Reliable*. As stated in the communication standard, these policies are not compatible, thus the module triggers an alert helping the MRS designer in the identification of the incompatibility (see Figure 5.14a). Differently, when the designer successfully configures the policies, such as configuring both the *weed_position* signals with a reliability equal to *Reliable*, the module highlights the signals in green (see Figure 5.14b) stating that policies are compatible.

5.2.3 Enactment

After completing the modeling and configuration phases, the MRS designer can enact the simulation of the running scenario. To this aim, the designer has to first run the ros2-web-bridge package, to enable the communication between the modeler and ROS. After that, the designer has to launch all the nodes composing the system: the splitter, the engine nodes, and the nodes related to the Gazebo simulation. As a result, the MRS designer can take advantage of the FAME-modeler interface to directly publish the executable BPMN collaboration over the */collaboration_diagram* topic, and of the Gazebo graphical interface to visualize the execution of the running scenario in a realistic environment. Indeed, once the collaboration diagram is published, the splitter node can receive it, split it, and share it with the related robots. After that, the robots start performing their activities according to the behaviors modeled in the BPMN collaboration.

The enactment of the BPMN collaboration is executed as follows. At the system start-up, the only entry point that can be triggered is the none start event in the drone's pool, since the other start events are bounded to



(a) Incompatibility detected (b) Compatible QoS policies

Figure 5.14: *weed_position* signals compatibility check

the triggering of events. Therefore, the drone starts its main process by performing immediately the *Take Off* script task activity which refers to the code implementing the lift-off of the robot. Once it has been performed, the next activity, *Explore*, refers again to the drone. This call activity enacts a process that takes as input a data object containing the map of the area to be explored.

In case the drone spots some weed grass, the *Explore* process triggers a signal event named *weed_found*. Consequently, the interrupting signal start event, contained in the event sub-process of the drone's pool, is triggered. Now, the sub-process pauses the *Explore* activity and performs its behavior. In detail, the sub-process triggers the publishing of a message named *weed_position* that contains the weed coordinates and starts the tractors' processes. Indeed, the main process in the tractors' pool starts when a *weed_position* message is received. Then, each tractor performs the *Get Position* script task, publishes back to the drone its position, and waits either for receiving a *closest_tractor* message or the passing of 30 seconds. In the meantime, the drone is waiting for the positions of the tractors, corresponding to the subscription over the *tractor_position* topic. In case no position arrives within 10 seconds, the process loops back to the publishing

of another *weed_position* message, otherwise the drone updates accordingly the *Closest Tractor* data object. This is also the input for the next step in the drone's process execution, which assigns the duty of removing the weed to the closest tractor by triggering the *closest_tractor* signal. Thus, every tractor is informed of who is responsible for removing the grass, so that only this instance of tractor goes ahead to the *GoTo* call activity, while the others terminate their process. The *GoTo* call activity takes as input the weed position and performs the movements required to reach it, finally, the *Cut Grass* script task orders the robot to enact the blade to remove the grass in the reached position and the process terminates. Notably, after the sending of the *closest_tractor* message, the drone terminates the activities involved in the event sub-process, thus the control flow is given back to the main process (stuck in the exploration). This allows the drone to continue the field exploration and, in case other weeds are found, the collaboration described so far is repeated.

Otherwise, if the drone exploration does not reveal any weed to remove in the field, the exploration activity ends and the drone sends a message over the *field_cleaned* topic. This information is caught both by the drone itself that triggers the *End Handler* event sub-process and terminates its duty going back to the base station by performing the *Go To* call activity and the *Land* script task. Additionally, also the tractors are subscribed to this topic. Therefore, the *field_cleaned* signal triggers tractors' *Field Cleaned Handler* event sub-process. Since the catching signal is non-interrupting, the tractors caught the signal without blocking the current process execution. Furthermore, they utilize a conditional signal *no_active_tasks* to evaluate the absence of any pending tasks, allowing them to return to the base station through a *Go To* call activity only when other executions are terminated.

Considering that both the drone and the tractors are equipped with a battery, the low battery error is handled via an interrupting event sub-process. Indeed, whenever the drone spots an internal low battery error, the *End Handler* sub-process is triggered to terminate the drone's execution. Differently, when a tractor spots an internal low battery error, it stops the current execution flow, performs a *GoTo* call activity, goes in an idle state until the battery is completely recharged, and finally terminates the sub-process. Notably, the idle state is expressed via the *battery_full* conditional event that is consumed only when the robot battery level is full.

Figure 5.15 shows two stages of the running scenario executed using the FAME framework. The objects in red are the tractors, while the white one is the drone. Notably, the blue area projected from each robot corresponds to the view range of the ultrasonic sensor. In more detail, Figure 5.15a depicts the drone performing the exploration of the field and detecting a weed grass, while Figure 5.15b shows the moment in which the drone triggers the tractors

(a) Weed identification (b) Weed removal

Figure 5.15: Running scenario simulation

and assigns the duty of removing the grass to the tractor closest to the weed.

5.3 Evaluation in the Physical Scenario

This section presents the application of the framework phases to an MRS deployed in a physical scenario and evaluates robots' performances with model interpretation via FAME enactment compared to the direct implementation of the ROS code.

Notably, this experiment does not replicate the same conditions as the running scenario of Chapter 3 developed in the Gazebo simulator. Instead, this evaluation is conducted in a simple and small-scale MRS due to various considerations that prevent the replication of the running scenario in a physical scenario. One of the primary challenges is the complexity and accessibility of the hardware. Replicating the running scenario would have necessitated the use of a professional drone equipped with autonomous navigation capabilities. Simultaneously, two agricultural robots with weed eradication capabilities would have been required. Moreover, an agricultural area designed for the integration of robots, e.g., equipped with charging stations and monitoring systems, would have been needed. An essential factor for the real-world reproduction of the running scenario, albeit beyond the FAME's objectives, involves the integration of artificial intelligence techniques. While the use of call activities facilitates the integration and reuse of ROS libraries, such as for enabling autonomous navigation, image recognition algorithms to distinguish weeds from crops would need to be implemented specifically for the system. This would involve the combination of experts from both the agricultural domain and the artificial intelligence one. All these constraints are outside the scope of FAME, which aims to demonstrate the usability of BPMN in defining and guiding the high-level behavior of an MRS.

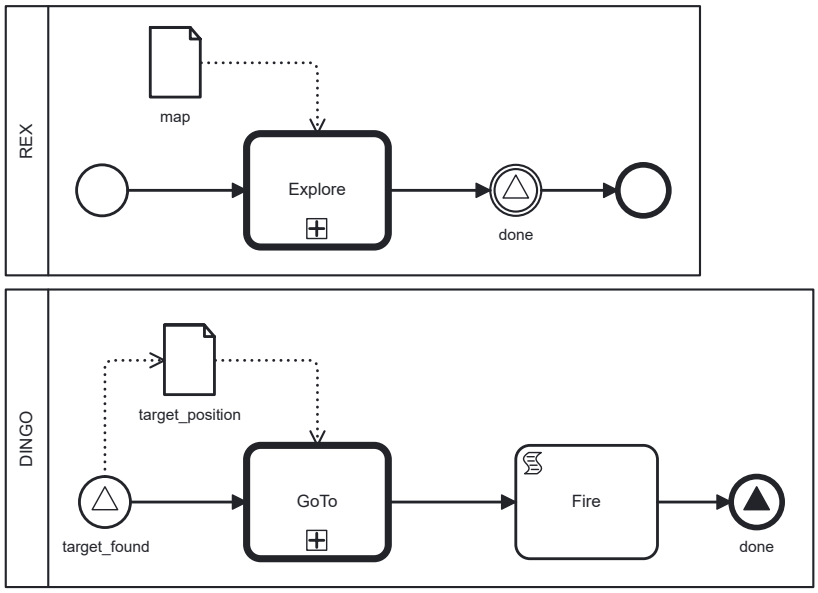


Figure 5.16: Collaboration diagram of the physical environment MRS

5.3.1 System description

The physical scenario comprises two cooperative ground vehicles with different capabilities, but with a common mission, i.e. identify a specific target and destroy it. REX (Robot EXplorer) is in charge of performing the exploration of the area and identifying the target that should be destroyed. Whereas, DINGO (DestroyING rObot) starts its execution when it receives the coordinates of the target so that it can reach and destroy it. The destruction of the target determines the ending of the MRS execution.

Essentially, the MRS cooperation resembles the one observed in the running scenario. Indeed, as in the running scenario, one robot has the objective of exploring an area to identify a target. The target identification triggers the other robot capable of directly acting on the environment and the target.

5.3.2 Modeling

The application of the FAME guidelines to this scenario resulted in the BPMN collaboration model depicted in Figure 5.16. At the system start-up, REX is the only one that starts its execution by performing the *Explore* call activity, to detect the desired target. When the exploration process ends, the robot moves to an idle state waiting for a *done* signal, before ending its main process. In case the exploration spots the target, this activates the DINGO's process, via the *target_found* start signal. Therefore, DINGO performs the *GoTo* call activity to reach the target and destroys it by enacting

the *Fire* script task. Its process ends with the throwing of a *done* signal that determines the end of the system's mission.

5.3.3 Configuration

The produced model is enriched with all the information required by the enactment phase. Therefore the *map* data object is configured to store four values for the x and y axes representing the boundaries of the area of interest, whereas, the *target_position* data object contains values that depend on the payload of the caught signal, i.e., the position of the target on the x-axis and the y-axis. The *done* signal is extended to be compatible with a boolean message type. Therefore, the catching signal only stores that the signal can handle boolean data, whereas the throwing also sets up the carried payload, equal to `true`.

Finally, the *Fire* script task is configured to send an internal command to the robot, to manage a specific actuator. Notably, this action has been implemented with the activation of a red LED. The obtained code, reported in Listing 5.8, initializes a publisher in the BPMN engine node able to send a `ColorRGBA` message type over the `led` topic. This publisher is then in charge of publishing a message equal to the red color.

```
const node = this.environment.variables.ros_node;
const publisher =
    node.createPublisher("std_msgs/msg/ColorRGBA", "led");
publisher.publish({r: 1.0, g: 0.0, b: 0.0, a: 0.0});
next();
```

Listing 5.8: *Fire* script task

5.3.4 Enactment

The enactment of the BPMN collaboration is executed firstly in the simulated environment to visualize the system behavior before moving on to the physical environment. To execute the experiment in the physical environment, the used robots are two smart cars equipped with four wheels, an ultrasonic sensor, an RGB LED module, and a Raspberry Pi 3B+ as an onboard computational unit. Each robot contains the ROS packages needed to control their devices and the FAME-ROS package that runs the BPMN engine node. Therefore, once the BPMN collaboration model is processed and executed, the cars can perform their collaborative behavior. Figure 5.17 shows part of the experiment. The simulated and the physical environments are put in comparison to emphasize their one-to-one correspondence. According to

the collaboration diagram, *REX* starts its execution with an exploration of the area (Figure 5.17a) and is able to identify a target (Figure 5.17b), which activates the *DINGO*'s process, that reaches the target (Figure 5.17c) and destroys it (Figure 5.17d).

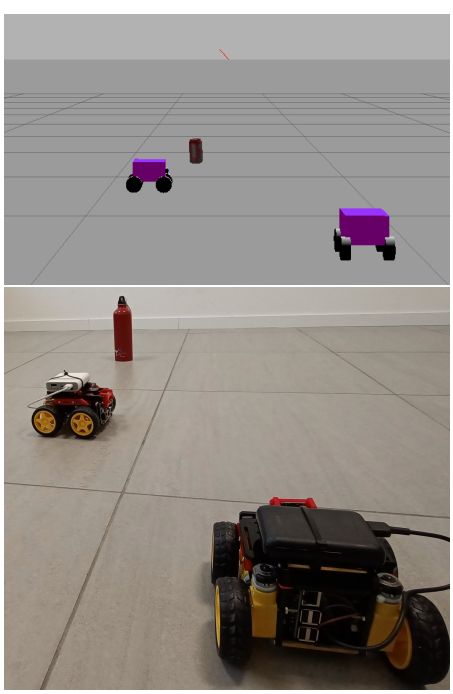
5.3.5 Performance evaluation

To assess the performances of the model enactment of FAME, the use of system resources and the impact of possible delays have been calculated. The experiments have been conducted on a virtual machine based on the Ubuntu 20.04 operating system. The system has 8GB of RAM, a quad-core CPU, and all the tools and dependencies required by FAME installed.

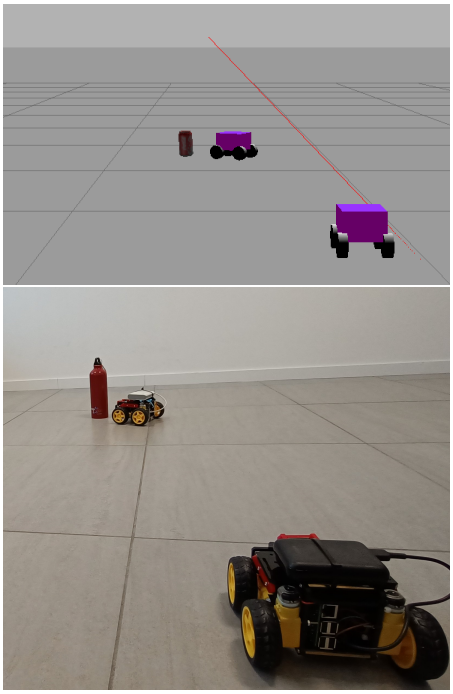
Referring to the use of system resources, the CPU and memory usage on ten executions of the physical experiment both by interpreting the model with FAME and by directly compiling the ROS code. The consumption of the system resources has been recorded using the `top` command.

Figure 5.18a shows the total CPU time used by the controller task, expressed in hundredths of a second. In the direct code compilation, this amount of time is lower than in the model interpretation approach. Figure 5.18b expresses the amount of memory that may be simultaneously accessed by multiple programs, expressed in megabytes. Here, the model interpretation performs more fluctuating and slightly worse behavior with respect to the code compilation. Figure 5.18c plots the total amount of virtual memory used by the ROS nodes, including all the code, the data, and the used libraries, expressed in megabytes. In this case, the approaches produce very similar results, minimizing the difference between the two. Finally, Figure 5.18d highlights how much memory the ROS nodes are consuming in the physical RAM, expressed in megabytes. The compiled code performs better than the interpretation of the model although, also in this case the difference is minimal. These results do not show a marked resource consumption difference between the two approaches, especially if considering that the ROS itself runs on powerful hardware. While introducing an engine in charge of interpreting the model and guiding the MRS mission accordingly, might seem less efficient, the resource analysis indicated that its impact is only slightly higher compared to code compilation.

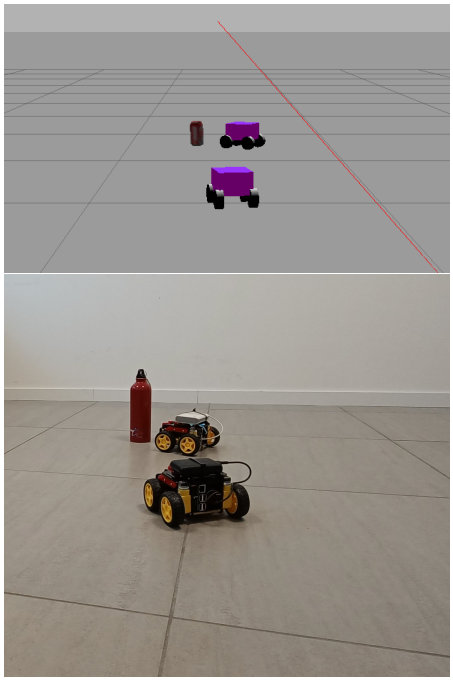
Additionally, an evaluation has been conducted on the physical robots to measure possible delays introduced by the usage of the FAME engine. Each physical robot is powered by a Raspberry Pi 3B+ with 1GB of RAM, a quad-core CPU, ROS, and the FAME dependencies installed. The experiment consists of making the robot move straight forward at a constant velocity in the direction of an obstacle, once it perceives a distance to the obstacle lower than 50 centimeters, it stops the wheels. On these bases, the distance between the obstacle and the point where the robot halted is measured. The



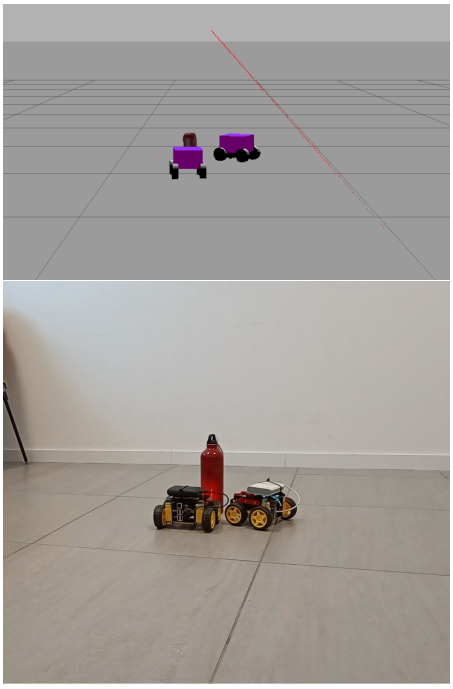
(a) REX exploration



(b) Target identified



(c) DINGO activation

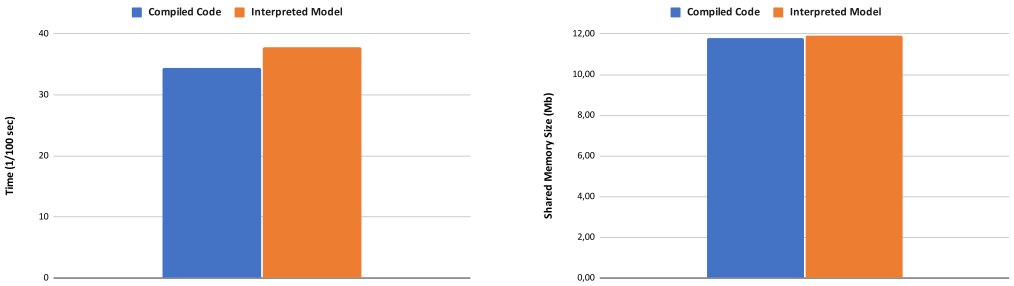


(d) Destruction

Figure 5.17: Real-life experiment

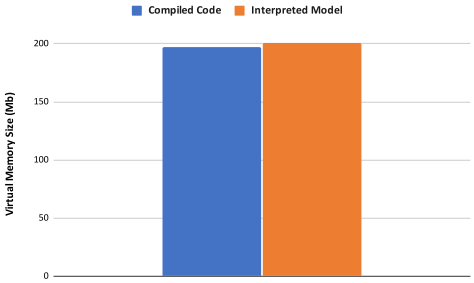
experiment has been conducted both with a direct code compilation and by exploiting the FAME framework, moreover, it considers both a simulated and a physical environment for the execution. The experiment has been run 10 times using three different speeds: 0.8 m/s, 0.9 m/s, and 1 m/s. Figure 5.19 summarizes and compares, on average, the distances perceived during the experiment. Specifically, a higher value corresponds to a better performance of the system, as it shows that the communication between the robot controller and the wheels has been performed quickly. Both in the simulated and in the physical environments the results are comparable, indeed there is not one approach performing better than the other. By aggregating the results obtained at different speeds, on average, in the physical environment FAME stops the robot at 39.07 cm from the obstacle, 1.11 cm after the approach with compiled code that stops the robot at 40.18 cm. While, in the simulated environments, FAME stops the robot at 14.09 cm while the approach with compiled code at 14.67 cm, with a difference of 0.58 cm. Notably, in the simulated environment, the final distance to the obstacle is slightly lower than in the real environment since the friction between the wheels and the ground is lower.

Summing up, the difference between the FAME model interpretation and the direct code compilation is not excessive. Indeed, considering that robots operate in dynamic environments, and thus should be able to efficiently react to them, the performance evaluation showed that the engine does not significantly impact robots' performances. However, in its current implementation, FAME is not intended for MRSs operating in critical situations which would demand higher robot efficiency to respond to unexpected situations.

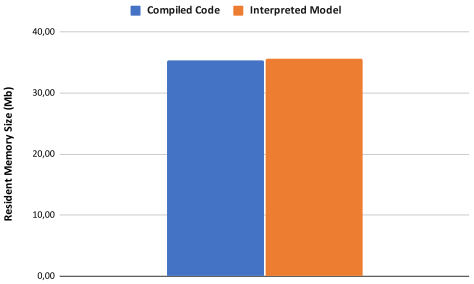


(a) Total CPU time used

(b) Shared memory used

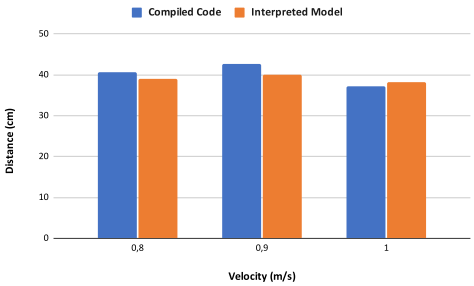


(c) Virtual memory used

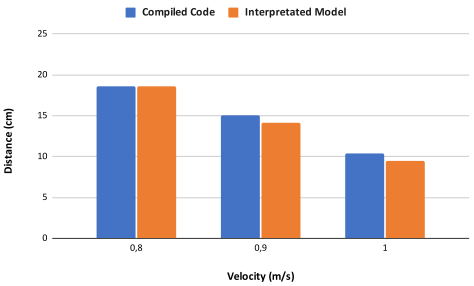


(d) Resident memory used

Figure 5.18: Resources consumption measurements



(a) Physical environment



(b) Simulated environment

Figure 5.19: Performances evaluation

Part III

Bottom-Up Approach

CHAPTER 6

The TALE Methodology

Following a bottom-up approach, this chapter presents the TALE (TAg-based muLti pErspective) methodology designed to support a robotic developer in the automatic extraction of multi-perspective event logs from the execution of a robotic system and their analysis through process mining[1]. In particular, the chapter investigates the complexity of obtaining robotic data capable of meaningfully representing robots' behavior. After that, the TALE methodology is provided to address the complexity of extracting robotic data and getting insight into the MRS mission as well as the related behavior of each robot, via process mining techniques.

6.1 Robotic Data

The application of process mining techniques is spreading to robotic systems [66, 75] for the analysis of the behavior of robots through process models extracted from event logs generated during the execution of robotic systems [22]. Nevertheless, the application of process mining techniques to robotic data is still in its early stages, resulting in the absence of well-established techniques specifically designed for their analysis. Unlike traditional processes with standardized event logs, robotic systems lack uniform

[1]It is worth noting that, despite the name similarity, the application of process mining to MRSs does not fall under the umbrella of Robotic Process Mining, i.e., a collection of tools capable of discovering automatable routines from logs recording the interactions between workers and web and desktop applications [55].

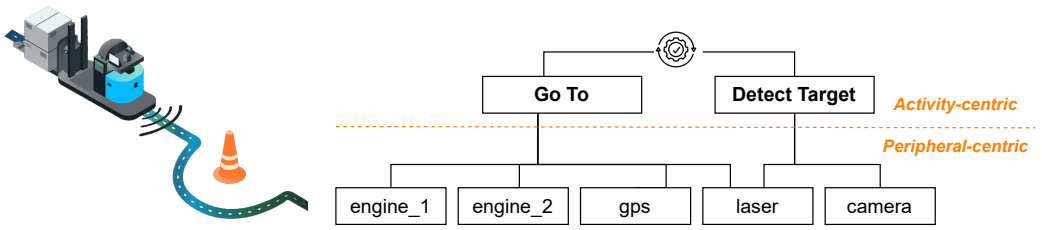


Figure 6.1: Robotic Example

data structures and comprehensive logging mechanisms, making it challenging to capture the diverse and complex activities involved in robotic operations. The main challenge lies in the difficulty of obtaining robotic datasets in a format suitable for applying process mining techniques. Existing datasets primarily provide information in the form of collected images and sensor readings, with the aim of training machine learning algorithms to optimize the system [71, 100]. Indeed, while images and sensor data are crucial for developing and enhancing low-level robotic capabilities, the integration of process mining techniques for the analysis of robot behavior would necessitate the availability of high-level event logs that capture the sequences of activities and decision points related to low-level data.

The usage of process mining techniques would require the generation of robotic event data in an activity-centric way, i.e., a recorded event should represent a meaningful high-level activity performed by a robot. However, robots collect a continuous flow of low-level data sent or received by their peripherals, i.e., sensors and actuators. Altogether these event data shape the robots' behavior from the point of view of the peripherals usage in the form of sequences of acting and sensing events with a very low-level of granularity and without correspondence with meaningful activities.

Specifically, during its execution, a robot produces continuous event data in a *peripheral-centric* way, keeping track of data produced by sensors and actuators. A robotic peripheral-centric event is typically characterized by the *time* in which the event is produced, by the name of the *peripheral* that triggered it, and by other attributes storing additional data. For instance, a robotic event triggered by a laser sensor may store the distance data read as an attribute. These event data represent that the robot has accessed a peripheral functionality, whereas to foster the research and the application of process mining techniques, the event logs should follow an *activity-centric* structure, recording high-level activities that should be executed to fulfill an objective in the process model, like a movement action to reach a destination.

To better clarify the difference between peripheral- and activity-centric event logs, consider an example involving a wheeled robot equipped with engines, GPS, camera, and laser sensors. In this scenario, the robot has to move

Time	Peripheral	Other Data
09:33:20.147	engine_1	...
09:33:20.147	camera	...
09:33:20.149	engine_2	...
09:33:21.239	gps	...
09:33:30.941	camera	...
09:33:32.410	gps	...
09:33:32.495	laser	...
09:33:32.499	laser	...
09:33:32:655	camera	...
09:33:33.003	engine_2	...
09:33:33.110	laser	...
09:33:33.118	laser	...
09:33:33.120	laser	...
09:33:33:240	camera	...
09:33:33.955	laser	...
...	...	...

(a) Peripheral-centric

Time	Activity	Lifecycle
09:33:20.147	Go To	start
09:33:20.147	Detect Target	start
09:33:33.240	Detect Target	complete
09:50:34.921	Go To	complete

(b) Activity-centric

Table 6.1: Event logs of the same robotic execution with different granularity

around to reach a destination (activity *Go To*) while looking for a target (activity *Detect Target*), see Figure 6.1. The performed activities exploit several sensors and actuators, therefore the event logs from both a peripheral- and activity-centric perspective are presented and discussed below. Table 6.1a shows an excerpt of a peripheral-centric event log of one robot execution, whereas Table 6.1b reports the same event log from an activity-centric point of view. Notably, the activity-centric event log is not derived by applying an abstraction technique to the peripheral-centric event log. Instead, Table 6.1a is a possible result of an event log produced by the robot's execution compared with the ideal activity-centric event log, of Table 6.1b, recording the performed activities. The peripheral-centric log describes the working flow of the robot peripherals, e.g., the first two rows of Table 6.1a indicate that the robot has used one engine and the camera. Instead, the activity-centric log provides a high-level view of the robotic execution, e.g., the first two rows of Table 6.1b show that the robot has concurrently started the Go To and Detect Target activities. To graphically visualize a correlation between the peripheral- and the activity-centric events and properly present the robotic granularity problem, the events belonging to the Go To activity are colored in purple, while the events correlated to the Detect Target activity are in yellow. In doing so, the obtained mixed occurrence of the colors in Table 6.1a shows that the peripheral events are produced in a continuous and unordered way, thus preventing the identification of a pattern for recognizing the related high-level activities. Moreover, the peripheral event log shows that both activities depend on the laser sensor device, creating a challenge in identifying the completion of an activity.

Therefore, to properly apply process mining and investigate its poten-

tialities in this domain, a robotic system should produce an activity-centric event log. This event log should contain a collection of robotic executions, possibly performed in different settings.

6.2 The Methodology

This section presents the TALE methodology steps defined to ease the extraction of robotic event logs and their analysis via process mining techniques. Figure 6.2 depicts the steps of the TALE methodology involving the event log **preparation** and the event log **analysis**.

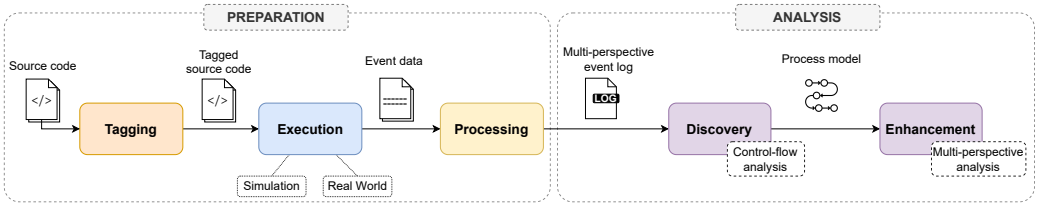


Figure 6.2: The TALE Methodology

6.2.1 Preparation

The event log preparation step aims to automatically extract an activity-centric event log suitable for applying process mining techniques. During the preparation, the source code of the MRS is *tagged* by the developer in correspondence with the activities of interest. Then, the system is *executed* and, finally, the produced data are *processed* to extract event logs. The following description explores these three sub-steps.

Tagging

The first sub-step guides the developer in integrating tags in the source code, making it possible to determine where a relevant activity begins and finishes. Notably, the user performing this sub-step is typically the developer of the robotic system, who has complete knowledge of the code controlling the robots' behavior; therefore, the tagging integration is a task that would not require much effort from the developer.

The developer indicates points, i.e., tags, in the code that trigger events during the robot execution. A tag has to report at least the name of the *activity*, and its execution status, i.e., the *lifecycle*, which either indicates the start or the completion transition of the activity, i.e., *start* or *complete*. Moreover, tags can be enriched with other perspectives of interest relevant to analyzing the robotic mission. For instance, considering a robotic system

capable of communicating with other systems, events related to message exchange can additionally store information related to the message identifier or message payload. Notably, an essential characteristic is that the identified tags must share the same level of granularity [99].

Execution

In the second sub-step, the code is executed multiple times in a simulation environment or a real-world scenario. Notably, a simulation environment allows for checking the system's correctness while avoiding deploying an incorrect behavioral model into physical devices. Additionally, the simulation eases and speeds up the massive execution of the system while allowing the testing with different environment settings.

During the system execution, a listener catches the tagged events and other data. Specifically, since the peripheral data produced by a robot, e.g., spatial position, can change during the execution of an activity, the listener also records a series of values referring to the perspectives of interest for the considered MRS.

Processing

Finally, in the processing sub-step, the recorded event data are processed to become an event log suitable for process mining analysis. Notably, the processing generates event logs suitable both for the application of traditional process mining techniques and for the usage of EKGs to apply object-centric analysis.

The processing first converts all the data recorded during the MRS execution into the CSV format. After that, the obtained event log is enriched with the MRS perspectives. Indeed, each perspective is correlated with tagged activities by comparing the timestamps. In doing so, the tags are enriched with the perspectives by matching the upper closest timestamp. Additionally, between the start and the completion of an activity, other events are inserted. These events have the same activity name, a lifecycle state equal to *inprogress* [70], and a series of continuous data referring to the perspectives of interest for the considered system. Moreover, TALE associates all the events of a robotic execution with a unique case identifier. Finally, the event log in CSV format is transformed into XES format. An excerpt of XES-based events schema with multiple perspectives is depicted in Listing 6.1.

```
...
<event>
    <date key="time:timestamp" value="2023-08-01T12:11:18" />
    <string key="concept:name" value="robot_activity" />
    <string key="lifecycle:transition" value="start" />
```

```
    <string key="lifecycle:state" value="none" />
    <float key="x" value="0.0" />
    <float key="y" value="5.0" />
    <float key="z" value="1.47" />
    <string key="resource" value="robot_name" />
    <string key="message" value="message_id" />
</event>
<event>
    <date key="time:timestamp" value="2023-08-01T12:11:20" />
    <string key="concept:name" value="robot_activity" />
    <string key="lifecycle:transition" value="none" />
    <string key="lifecycle:state" value="inprogress" />
    <float key="x" value="0.5" />
    <float key="y" value="4.8" />
    <float key="z" value="1.46" />
    <string key="resource" value="robot_name" />
    <string key="message" value="message_id" />
</event>
...
```

Listing 6.1: XES-based schema of multi-perspective event log

Therefore, TALE prepares activity-centric event logs, where each recorded event represents a meaningful robotic activity and is enriched with all the perspectives the robotic developer chooses. The final result of this step is a multi-perspective event log extracted both in the CSV and in the XES formats. Notably, in the preparation step, tag integration relies entirely on manual input from the robotic developer. Therefore, the level of granularity and the properties stored in the tags depend on the targeted system and the type of analysis the developer intends to perform. Although manual tagging is prone to errors, which can affect the quality of the resulting event log, the flexibility of the preparation step allows the tags to be adjusted to improve the produced event logs.

6.2.2 Analysis

The event log generated in the previous step is suitable for the application of process mining techniques. In its current state, TALE supports the MRS control-flow analysis via process discovery techniques to produce the process model depicting the MRS mission. Additionally, TALE enables continuous data analysis using process enhancement to extend the discovered process model with the information of perspectives data stored in the event log. Notably, since the methodology can be applied to produce multi-perspective robotic event logs, the choice of different perspectives affects the type of analysis that will be applied to them. Moreover, since the event logs serve as standard input in process mining, several techniques can be applied, or

developed, to improve the overall analysis [4].

Specifically, the selection of the analysis supported by TALE has been driven by both current literature on MRSs and the data the reference system can produce. The control-flow analysis is the core step for examining the system's behavior, combined with the need to find a trade-off between the robot's spatial occupancy, execution time, and energy consumption [6, 101]. In the literature, this trade-off has mainly been addressed by transitioning from single-robot systems to MRSs [47], thus integrating the need to also analyze robots' cooperation. The TALE analysis step handles behavior and execution time through process discovery techniques, while spatial and cooperation aspects are examined using process enhancement techniques. Due to the lack of battery data, TALE does not implement this aspect, although it could be equally analyzed using enhancement techniques.

Control-flow: *Process Discovery*

The control-flow analysis is the first analysis sub-step prescribed by TALE. This step exploits the event log produced during the preparation step and aims to discover the process model depicting the behavior of the robots.

Starting from traditional process mining approaches, TALE adopts process discovery to provide a process model that describes the robot's behavior, by observing the event log. The discovery algorithm, presented in [94], extracts from the log the events belonging to the robot and collects the directly-follows relations, i.e., binary relations indicating whether an activity occurs immediately after another. The relations between the activities of the robot are graphically represented as a DFG, i.e., a directed graph whose nodes represent the process activities and the edges indicate a directly-follows relation. In this form, the DFG summarizes the different executions of the robots seen in the event log. Each path from the start to the end node represents a sequence of activities the robot performs during the system execution. Moreover, the labels of the edges report the number of times the connected activities have been executed one following the other. Notably, DFGs correlate events temporally by flattening other information such as the robot performing the activity or the messages enabling MRS cooperation. This permits discovering the DFG depicting the overall mission, where individual robot behaviors are not identifiable. However, to obtain different views of the DFG, the event log can be filtered before performing process discovery. For instance, filtering by robot identifier enables the discovery of the process model depicting the behavior of the selected robot.

Differently, the usage of EKG enables the generation of a multi-entity DFG [7]. In this case, the nodes of the graph can both represent the events recorded in the event log and the entities impacting the execution, such as the robot identifier. Specifically, the EKG temporally relates the event

nodes only if they share a specific entity type [34]. This avoids the event log flattening since additional information is used to create the relations from one node to another. Furthermore, as the EKG generates a distinct event node for each event recorded in the event log, manipulation operations are fundamental to facilitate the extraction of desired insights. Notably, among these operations, the node aggregation operation is required for querying the data model to discover the process model and support process mining across different behavioral perspectives [33]. Therefore, TALE adopts a query-based process discovery mechanism to produce an aggregated EKG that represents robots' behavior and inter-robot communication.

Multi-perspective: *Process Enhancement*

The multi-perspective analysis is the second sub-step prescribed by the TALE-analysis step. Indeed, it aims at extending the process model extracted using process discovery techniques, with additional information found in the event log [28]. Notably, the enhancement is independent of the approach used for discovering the process model.

Specifically, in the analysis of MRSs, TALE extends the process model view with the following information.

- **Spatial Occupancy**. The spatial perspective strongly impacts the control-flow, since robots may act in different parts of an unknown environment [1]. TALE extracts from the event log the x, y, and z coordinates of each event. These values represent the position in the environment occupied by the robot when performing its mission through points in a 3D space. In doing so, the activities in the discovered process model are associated with information related to the spatial perspective. Specifically, each activity is correlated with a set of points delineating the robot's position during its execution. This enables the analysis of robots' behavior from a broader view to understand if the spatial occupancy has led to certain behaviors.

- **Inter-robot Communication**. The distributed and cooperative nature of MRSs requires a timely and accurate exchange of information among devices [62]. Indeed, by exploiting communication mechanisms the robots can coordinate with each other and manage task assignments. TALE extracts from the event log the information related to the messages exchanged between robots to provide insights on the communication performances. Therefore, the process model view is extended with the inter-robot communication perspective showing the performances of the exchanged messages in terms of duration and message loss.

6.3 Related Works

Considering the phases of the TALE methodology the analysis of the related works has been conducted by exploring both the works addressing the data preparation of low-level events, and the works that apply process mining techniques to robotic systems.

Data preparation. Given that the application of process mining techniques in the robotics domain is still emerging, there is a lack of methodologies for extracting high-level activities from low-level robotic data suitable for the generation of process mining-compliant datasets. Therefore, to have a broader view of the current state of the art of process mining-based data preparation methodologies, the investigation has been conducted on works aiming at extracting high-level activities from IoT events. The objective is to assess whether these approaches can be adapted for the abstraction of robotic data. Furthermore, approaches that explore ways to manage and analyze all the continuous data captured by IoT systems are analyzed. This analysis aims at understanding emerging approaches that could deal with the continuous data produced by robotic systems.

Concerning high-level activity abstraction, in [60] and in [85] the authors label low-level event logs to overcome the uncertainty brought by the application of unsupervised learning techniques, such as identifying the name of clusters and choosing the desired level of abstraction. However, labeled-based approaches aim to associate each IoT low-level event with a static label, to enable the extraction of high-level activities. Nevertheless, in robotic systems, a low-level event may have been used to perform different activities, thus it is unfeasible to label a low-level event statically. In this regard, [81] and [97] propose approaches for enabling an expert to manually detect in the event log high-level activities starting from groups of, possibly shared, sensor data. These approaches require the domain expert to make considerable effort to associate all the groups identified in the event log with the corresponding high-level activity. Differently from these approaches, TALE enables a domain expert to define the structure and the level of granularity of the event log before executing the system. Indeed, the idea of tagging the robotic code before system execution, also exploited in the approach presented in [48], enables the extraction of event logs that properly reflect high-level activities and are suitable for the discovery of meaningful process models.

Considering the integration of continuous data within event logs, in [58] authors propose an extension of the XES standard. Specifically, the values generated by IoT devices are recorded between the start and completion of an event with a new lifecycle transition: *stream/data*. In this way, the event log combines a joint representation of process event logs and IoT data related to

the environment where these events occur. In the object-centric perspective, in [41] the authors propose an extension of the OCEL standard, i.e., DOCEL, capable of supporting data awareness. The proposal aims at integrating the representation of object attributes that can change over time. Differently, in [96] the author separates the concept of events from the data produced by the continuous measurements of an industrial system. Indeed, the events recorded by the system and data captured by the continuous measurement are maintained as distinct logs, and their correlation is established using the timestamp value and the shared object. TALE adopts the integration of continuous data directly in the event log. This is achieved by exploiting the concept of the *inprogress* lifecycle state, to associate each event with the continuous data generated by the robots.

Process mining analysis. Recently, a few works have experimented with the application of process mining techniques for analyzing and diagnosing a robotic system. In [66] the author proposes a process conformance verification using a process mining tool on the event log of a single-robot system. The approach aims to check the correctness of the robot's execution, by supporting the correctness and quality assurance of a robot from design to execution. Considering the application of a robotic system in the manufacturing domain, in [46] the authors exploit process mining to get insight from the robot's executions. Indeed, the aim is to assess the use of process mining to discover robots' emergent behavior, and the decision-making process, find repetitive patterns, and identify bottlenecks. Focusing on multi-robot scenarios, in [75] the authors assess the application of process mining to extract robots' mission and perform analysis related to the execution time, resource occupancy, and area usage. In [76] the authors apply process mining techniques to event logs produced in a robotic soccer scenario. Process discovery has been applied to construct a process model of both the team formations and the behavior of each robot. Additionally, decision-mining techniques have been exploited to discover the conditions leading to team formation. However, the presented works identified a common drawback in the application of process mining in robotic systems: the huge amount of data generated at a low level of granularity. This may lead to obtaining processes describing the system's behavior that are not meaningful, i.e., do not reflect the actual behavior of the robots. Moreover, the difficulty in extracting and manipulating robotic data leads to a shortage of process mining approaches that could be exploited or designed to address the dynamics of robotic scenarios. In this context, the generation and preparation of data following the TALE methodology can foster the generation of process mining compliant robotic datasets, thus the application of process mining to the robotics domain.

CHAPTER 7

TALE at Work

The TALE methodology aims at fostering the application of process mining techniques to analyze the behavior of MRSs and the related multiple perspectives. This chapter presents the tools implemented to support the methodology. Following this, the TALE steps are applied to the scenario outlined in Chapter 3. Notably, the source code for replicating these steps and all the analysis results can be found online[1].

7.1 TALE Implementation

The TALE methodology can be automatized both in the preparation and in the analysis steps. Specifically, data preparation is supported by a script collection enabling the automatic translation of the ROS-based data, generated during the MRS execution, into multi-perspective event logs suitable for the application of process mining techniques. Differently, the analysis is supported by a tool enabling both the discovery of the MRS process model and the application of control flow and multi-perspective analysis. Notably, the implementations are flexible enough to be used independently of each other.

[1] https://pros.unicam.it/tale

7.1.1 TALE-preparation

Data generated during the execution of an MRS can be acquired using the *ros2bag*[2] library. This library serves as a listener for a custom set of ROS topics transmitted within the robot's network. However, the recorded data are stored in a format specific to ROS, which needs to be processed to create event logs. To facilitate this processing, a set of scripts has been developed, supporting the data preparation workflow illustrated in Figure 7.1. These scripts enable the transformation of the raw ROS data into multi-perspective event logs.

After the execution of the MRS, the unrefined ROS data is given as input into the *Data Conversion* script. This script decodes the ROS-based data into several CSV files, representing both the events generated by the system's tags and other perspectives identified by the MRS developer. However, to create process mining-compliant event logs, the retrieved event data requires further refinement. This refinement is carried out by the *Data Enrichment* script, which enriches the tag-based data by incorporating the perspectives of interest and adding events keeping track of the evolution of the perspectives. The result is a multi-perspective event log available in both CSV and XES formats, providing a comprehensive overview of the MRS execution.

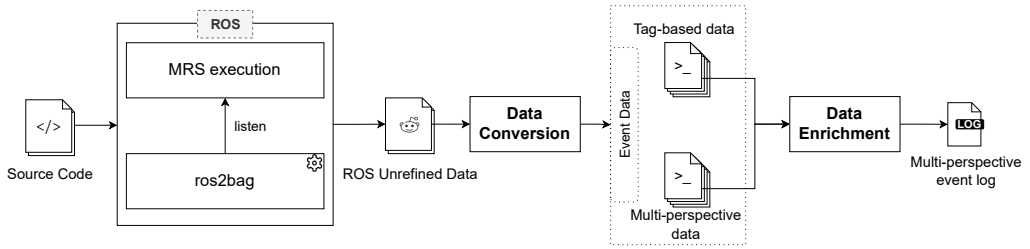
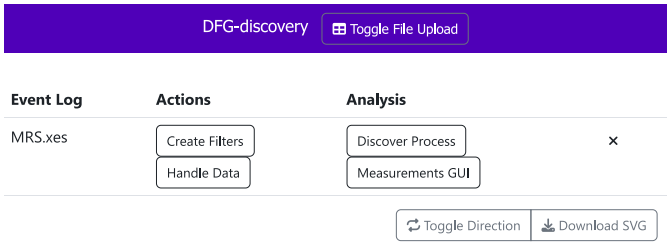


Figure 7.1: TALE-preparation workflow

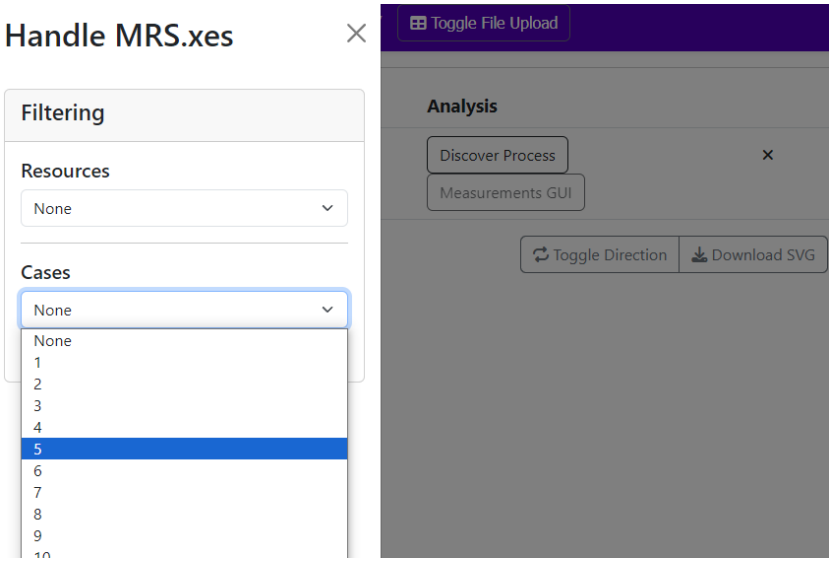
7.1.2 TALE-analysis

The tool enabling the analysis is a web application built upon the Flask python framework[3] that provides easy access to the analysis step. The tool is composed of three modules. Two modules are responsible for generating the process model from the event log, achieving this through discovering the DFG and the EKG respectively. Differently, the third module utilizes the information about the process model generated by either of the modules mentioned above to graphically render the received process model and enhance it with multiple perspectives.

[2]https://github.com/ros2/rosbag2
[3]https://flask.palletsprojects.com



(a) Data upload



(b) Data manipulation

Figure 7.2: DFG-discovery data uploading

The **DFG-discovery** module enables the discovery of a DFG based on the event log given as input. It exploits the *pm4py*[4] library both for importing and manipulating the event log and for discovering the DFG.

The interface of this module offers functionalities for both uploading event logs and applying filters to them, as illustrated in Figure 7.2a. Indeed, since the DFG discovery correlates the events based on temporal occurrence, analysts can exploit the interface to apply filters to the event log before initiating the process discovery. This allows them to refine the event log and focus the analysis on specific aspects of interest. At present, the module incorporates two primary filters: one for the case identifier and another for the robot identifier, see Figure 7.2b.

The discovery of the DFG results in a dictionary mapping pairs of activities that follow each other directly, to the number of corresponding observations, i.e., the number of occurrences one node directly follows another node. Additionally, to make the graph more user-friendly, and thus ease the

[4]https://pm4py.fit.fraunhofer.de

analysis, the module appends a *start node* to the events that serve as a start event, and an *end node* to the events that serve as end nodes. The output of the discovery is a collection of nodes, edges, and labels representing the DFG structure.

The **EKG-discovery** module enables the discovery of an EKG based on the custom choices of the analyst. It exploits the *neo4j*[43] graph database to store nodes and relationships built upon the observation of the event log, and the *Cypher Query Language*[37] to create, manipulate, and retrieve data from the graph. This module provides a user interface to automatize the construction of EKGs. To enable the connection and querying with the database, the module exploits the *neo4j python*[5] library.

The homepage depicted in Figure 7.3a eases data uploading by automatizing the query based on the selected file and the columns the analyst wants to take into account. Therefore the query follows the Cypher syntax and expresses variables by marking them with the $. The query[6] for loading the event log and creating an event node for each recorded event, is in the form of:

```
LOAD CSV WITH HEADERS FROM "file:///$path" as line
CREATE (e:$node_type {Log:"$log_name" {$data_list}})
```

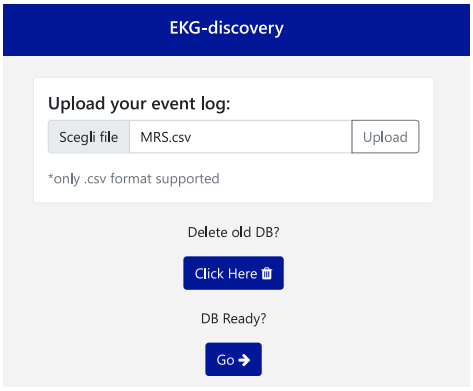
where *path* is the path to the CSV file, the *node_type* is associated with the type of the nodes the query aims at creating, in this context it refers to the Event type, the *log_name* stores the name of the input log and finally *data_list* is a dictionary associating the node with several properties. For instance, following the data uploading depicted in Figure 7.3b, an Event node, generated after the loading query, has the structure of Listing 7.1.

```
<id>: node_id
Log: MRS
activity: activity_name
msg_id: message_name
robot: robot_name
time: event_timestamp
x: x_position
y: y_position
z: z_position
```

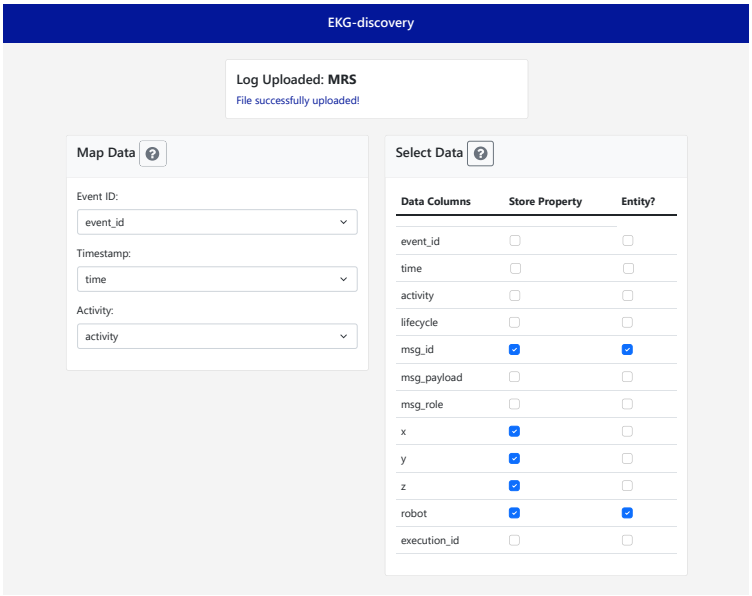
Listing 7.1: Example of event node stucture

[5]https://pypi.org/project/neo4j

[6]Notably, a detailed description of the query semantics can be found in the Cypher documentation (https://neo4j.com/developer/cypher/).



(a) Data uploading



(b) Data mapping

Figure 7.3: EKG-discovery data uploading

Moreover, selecting the entities in the user interface automatically executes the query for generating the corresponding Entity nodes and creating a correlation relationship between event nodes and related entity nodes. After event-entity correlation creation, a directly-follows relationship is created between event nodes that are correlated to the same entity and are temporally sequential. The executed queries are in the form of:

```
MATCH (e:Event)
UNWIND e[$entitytype] AS ent_name
WITH DISTINCT ent_name, e
MERGE (n:Entity {id: ent_name, type: $entitytype})
MERGE (e)-[c:CORR]->(n)
```

```
MATCH (e:Event)-[:CORR]->(n:Entity)
WHERE n.type = $entitytype
WITH n, e AS event_nodes ORDER BY e.time
WITH n, collect(event_nodes) AS event_list
UNWIND range(0, size(event_list)-2) AS i
WITH n, event_list[i] AS e1, event_list[i+1] AS e2
MERGE (e1)-[df:DF {type:$entitytype,
                   id:n.entity_id,
                   weight: 1}
            ]->(e2)
```

where the variable *entitytype* is associated with the entity types checked in the dedicated panel.

Finally, the analyst can select the desired level of aggregation. Indeed, the aggregation can be performed only based on the event activity name or can be performed by considering also entities. Notably, the aggregation based on the activity name aggregates event nodes that share the same activity name and aggregates the corresponding directly-follows edges by storing the number of observed occurrences. Differently, the aggregation that also considers entities, aggregates event nodes only if they share the same activity name and entity identifiers. The aggregation query executed is in the form of:

```
MATCH (e:Event)-[:CORR]->(n:Entity)
WITH DISTINCT e.activity as act, {$entType}
MERGE (c:Class {id: act+"_"+{$entType}, name: act)
```

where *entType*, which can also be null, represents a set of entity types over which the analyst wants to perform the aggregation. Notably, following the formalization in [34], the aggregation of Event nodes is represented in the form of nodes labeled as *Class*.

The output of the aggregation is a collection of nodes, edges, and labels representing the structure of the EKG.

Finally, the **Enhancement** module receives the collection of nodes, edges, and labels produced by one of the aforementioned modules. It exploits the *dagre-d3*[7] graph library, to render the received information in an easy-to-read and investigable process model. Additionally, the module enables the generation of an enhanced process model by integrating the perspectives view with the activities. The perspectives are rendered in the form of plots built with the *plotly* library[8] and generated from the information retrieved in the event log. Specifically, the spatial perspective is rendered in the form of a 3D scatter plot representing the spatial occupancy of the robots. Whereas, the communication perspective is rendered in the form of a bar chart showing

[7]https://github.com/dagrejs/dagre-d3

[8]https://plotly.com/python

the impact of the communication messages in terms of duration and message loss.

This module provides an overview of the system execution by showing the process model of the MRS, the behavior of the robots in terms of spatial occupancy, and the inter-robot communication performance. Additionally, the analyst can perform more accurate analysis over single activities, by selecting a node in the graph and the perspectives of interest.

7.2 Evaluation in the Running Scenario

This section presents the application of the TALE methodology to the running scenario presented in Chapter 3 and executed in the Gazebo simulator. Notably, a detailed system setup and the extracted data are illustrated in the online documentation.

7.2.1 Preparation

To be compliant with the ROS architecture of the running scenario, a custom ROS message capable of containing the tag data is defined. Therefore, the message is defined to contain the *activity* field that identifies the action that the robot is performing, and the *lifecycle* that determines the status of the related activity. Once the message has been defined, the activities performed by the robots are identified, selecting a level of abstraction appropriate for behavioral analysis. Specifically, the drone activities are: *take off*, *explore*, *weed found*, *weed position*, *tractor position*, *closest tractor*, *low battery*, *field cleaned*, *return to base*, and *land*. For the tractors *weed position*, *tractor position*, *closest tractor*, *move*, *cut grass*, *low battery*, and *return to base*. Consequently, tags are placed in the points of the code where activities begin or finish. Notably, during tag integration, the activities related to messages end with a *!*, if they are sender type and with a *?* if they are receiver type [98]. To better clarify the tag concept, Listing 7.2 shows an excerpt of the tag integration in the ROS code. Here the *CutGrass* function call is preceded and succeeded by publishing a topic containing the aforementioned fields.

After the integration of the tags, the running scenario has been executed in the Gazebo simulator with different initial settings. Specifically, for each execution, the scenario has random locations of weeds and variable battery discharge levels for each robot. To obtain a significant amount of data, the system has been executed 70 times. The execution of the simulations generates robotic event data that are processed to extract event logs enriched with spatial and communication perspectives.

```
tag = TagTopic()
tag.activity = "cut_grass"
tag.lifecycle = "start"
ros_publisher.publish(tag)
CutGrass(ros_node)
tag.lifecycle = "complete"
ros_publisher.publish(tag)
```

Listing 7.2: Excerpt of tag integration

Table 7.1 shows an excerpt of the multi-perspective event log related to a simulation run. The *Case* value uniquely identifies one run of the system execution, and the *Time* value stores a timestamp, i.e., the date and time in which the event occurred. *Activity* is related to the high-level activity produced via tags, whereas *Transition* and *State* represent the lifecycle, respectively an event execution transition or the event state. Considering the spatial perspective, the x, y, and z values store the position of the robot when an activity starts, is in progress, or completes. Moreover, to distinguish the robot that generates a given event, its identifier is saved in the *Robot* field. Finally, events defining message passing also store the *MsgID*, i.e., the identifier of the message shared between robots.

Case	Time	Activity	Transition	State	Attributes				
					x	y	z	Robot	MsgID
...	...	...	...	...	...	...	...	...	...
38	10:08:53.12	explore	null	inprogress	0.59	3.33	1.48	drone	null
38	10:08:55.01	explore	null	inprogress	0.99	4.01	1.48	drone	null
38	10:08:56.55	explore	null	inprogress	1.12	4.52	1.48	drone	null
38	10:08:57.61	explore	complete	null	1.66	5.03	1.48	drone	null
38	10:08:57.88	weed_found	start	null	1.66	5.03	1.48	drone	null
...	...	...	...	...	...	...	...	...	...
38	10:08:58.00	weed_postition!	complete	null	1.66	5.03	1.48	drone	m52
38	10:08:58.30	weed_postition?	start	null	0.25	1.12	0.0	tractor_1	m52
38	10:08:58.30	weed_postition?	complete	null	0.25	1.12	0.25	tractor_1	m52
38	10:08:58.30	tractor_position!	start	null	0.25	1.12	0.0	tractor_1	m53
38	10:08:58.32	tractor_position!	complete	null	0.25	1.12	0.0	tractor_1	m53
38	10:08:58.34	weed_postition?	start	null	0.17	1.12	0.0	tractor_2	m52
38	10:08:58.35	tractor_position?	start	null	1.66	5.03	1.48	drone	m53
38	10:08:58.35	tractor_position?	complete	null	1.66	5.03	1.48	drone	m53
38	10:08:58.36	weed_postition?	complete	null	0.17	1.12	0.0	tractor_2	m52
38	10:08:58.37	tractor_position!	start	null	0.17	2.04	0.0	tractor_2	m54
38	10:08:58.39	tractor_position!	complete	null	0.17	2.04	0.0	tractor_2	m54
38	10:09:00.91	tractor_position?	start	null	1.66	5.03	1.48	drone	m54
...	...	...	...	...	...	...	...	...	

Table 7.1: Excerpt of multi-perspective event log

7.2.2 Analysis

The produced multi-perspective event log has been analyzed both from the control-flow perspective and the multiple perspectives. Indeed, by exploiting the tool it is possible to discover the process models of the robots and enhance it with the related perspectives. Notably, for the sake of presentation, the features and results of the tool for the multi-perspective analysis are displayed on the EKG, although they are equally applicable to the DFG.

DFG-discovery. The event log produced by the TALE-preparation can be easily uploaded into its interface. After the event log uploading, the analyst can discover the DFG depicting the control-flow of the MRS. Figure 7.5 represents the DFG discovered with the log prepared with TALE and depicts the process of the overall MRS. Each node of the DFG represents the performed activity, and the arcs are labeled with the number of times the relation between the connected activities has been executed. In this representation, event connections are established only based on temporal occurrences. Consequently, events executed by different robots cannot be differentiated. Nevertheless, this DFG offers a comprehensive overview of the MRS mission. It reveals a consistent pattern where the drone always concludes (via the *land* activity) its execution before the tractors.

However, to analyze each robot's behavior, it is possible to filter the event log by the robot identifier. Figure 7.4 depicts the process of the drone. The process shows that the drone properly performed the desired behavior, by starting with a *take off* activity and then moving to the *explore* activity. During this phase, it can navigate and look for weed grasses. When a weed is identified, it triggers a *weed found* and notifies the tractors through the *weed position* message. Afterward, when it has received tractor responses, it moves to the *tractor position* status. Then, it can elect the tractor closest to the weed, i.e., *closest tractor*, and restart its exploration. Finally, the drone's execution ends when it spots a *low battery* state. Therefore, it can perform the *return to base* and *land* activities. Notably, the extracted process shows that the drone has never performed the *field cleaned* activity, meaning that it has not been able to explore the whole field.

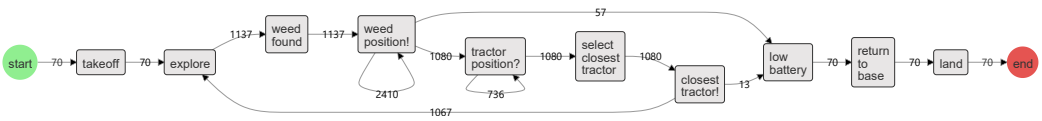


Figure 7.4: DFG filtered on the *drone* resource

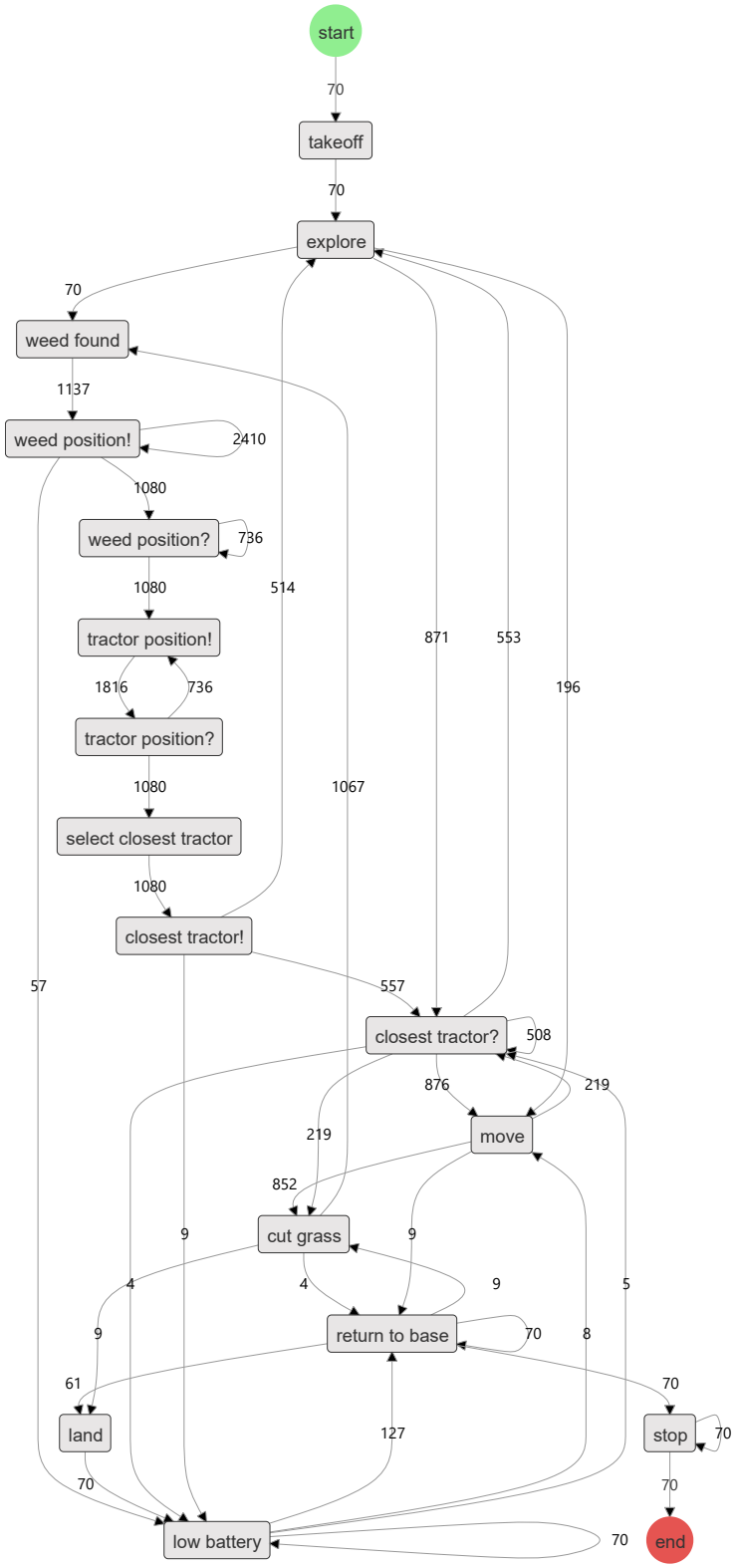


Figure 7.5: DFG of the running scenario

EKG-discovery. As described in Section 7.1.2, the event log produced by the TALE-preparation can be easily uploaded into the EKG-discovery interface, by both selecting the information and the entities of interest.

Data manipulation enables a customized aggregation of event nodes. The analyst can select how to aggregate the events exploiting the interface. Indeed, for each selected entity, an aggregation selector is shown, see Figure 7.6.

By setting the aggregation requirements, the analyst can easily discover the EKG depicting the control-flow of the MRS. Following the aggregations selected in Figure 7.6 each event node with the same activity name and robot identifier is aggregated, i.e., all the same-named activities are aggregated only if they are correlated to the same entity of robot type, whereas the aggregation over message identifiers has not been selected. Figure 7.7 represents the EKG aggregated based on these parameters and uses the log prepared with TALE to depict the process of the overall MRS. Each node of the EKG represents the activity performed by each robot since the node aggregation has been set to aggregate nodes sharing the same activity name and robot identifier. Notably, the tool employs distinct colors for nodes based on the type of aggregation performed. Specifically, in this EKG, drone activities are represented in blue, while tractor activities are depicted in green and orange. Moreover, an additional node referring to the unique entity values is connected to the first activity correlated with the corresponding entity.

The arcs are labeled with the number of times the relationship between the connected activities has been executed, and the type of relation between nodes. Specifically, a *robot* relationship type connects event nodes from the same robot, whereas, a *msg_ id* relationship associates and depicts the inter-robot communication, i.e., an inter-robot communication event follows another one if there is a message entity correlation in common by the two. Therefore, the discovered EKG can represent the collaboration process model of the MRS. As in the previous case, the EKG shows that the *field cleaned* activity has never been performed by the drone.

Indeed, the control-flow analysis shows that the running robots worked as expected, without performing any incorrect behavior. However, to better understand the reason for not having an expected activity in the process, it is necessary to inspect other system perspectives that may have affected its behavior.

Enhancement. After the discovery of the process, the enhancement module takes the generated process to visualize both the process model and the spatial and communication perspectives. The result, depicted in Figure 7.8, shows a global view of the system and its perspective.

For the spatial perspective, TALE extracts from the event log the robots' information related to their position in the space, to reflect the performed activities in a 3D plot. The spatial perspective in Figure 7.8 shows the space

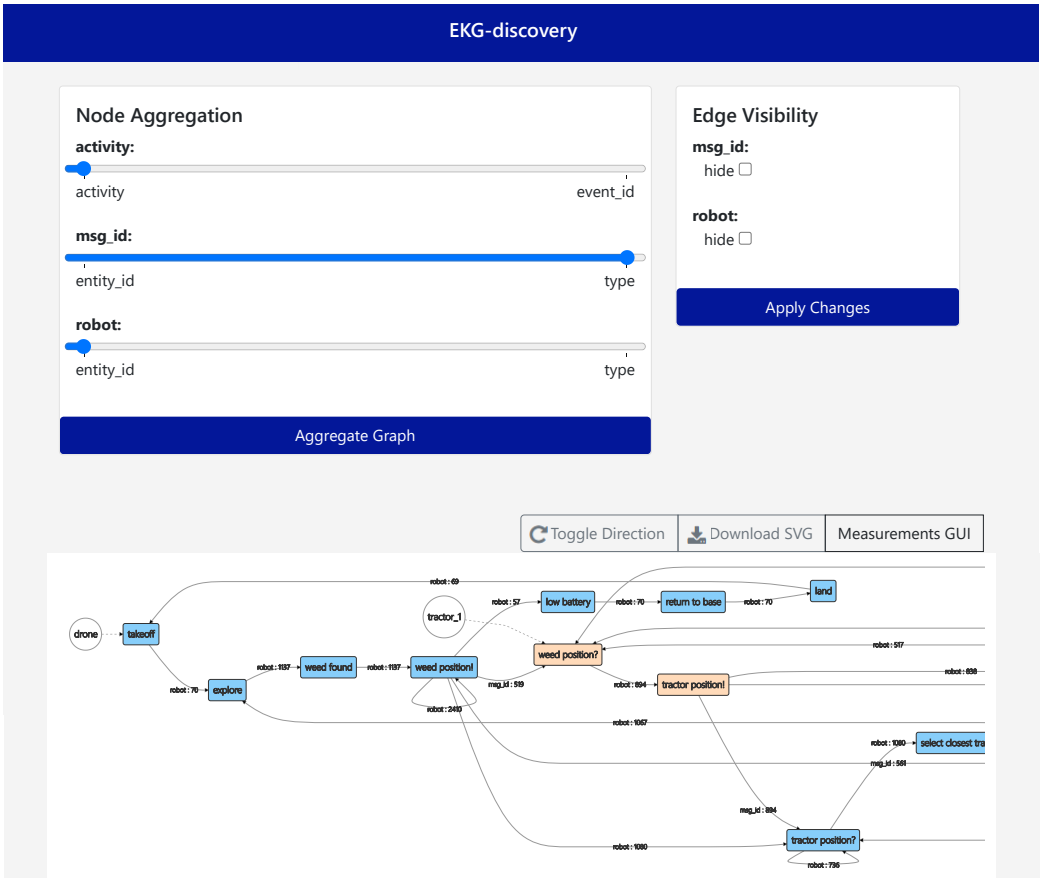


Figure 7.6: EKG-discovery aggregation interface

occupancy of the MRS, in a 10x10-meter field, during all the executions of the system, representing with different symbols and colors different activities performed by the robots. For what concerns the drone, the plot shows that in none of the cases the drone's Explore activity, represented by red circles, has been able to navigate within the intended field. Specifically, the drone always takes off from the base station and starts the navigation of the field. During the navigation, it spots some weeds and reports them to the tractors. If, on the one hand, this sequence of actions has already been verified during the control-flow analysis, on the other hand, the 3D analysis of the space helps to identify further potential errors in the system. Specifically, although the drone has properly performed the desired activities, it could not explore most of the field. This might be due to a programming error, which prevents the robot from navigating correctly. Additionally, a system developer may have overestimated the capabilities of the robots. For instance, a single drone may not be sufficient for exploring the intended field, or the two tractors are too slow to satisfy the demands of the drone. Similarly, the tractors properly start and end their execution from the base station and react to the drone's

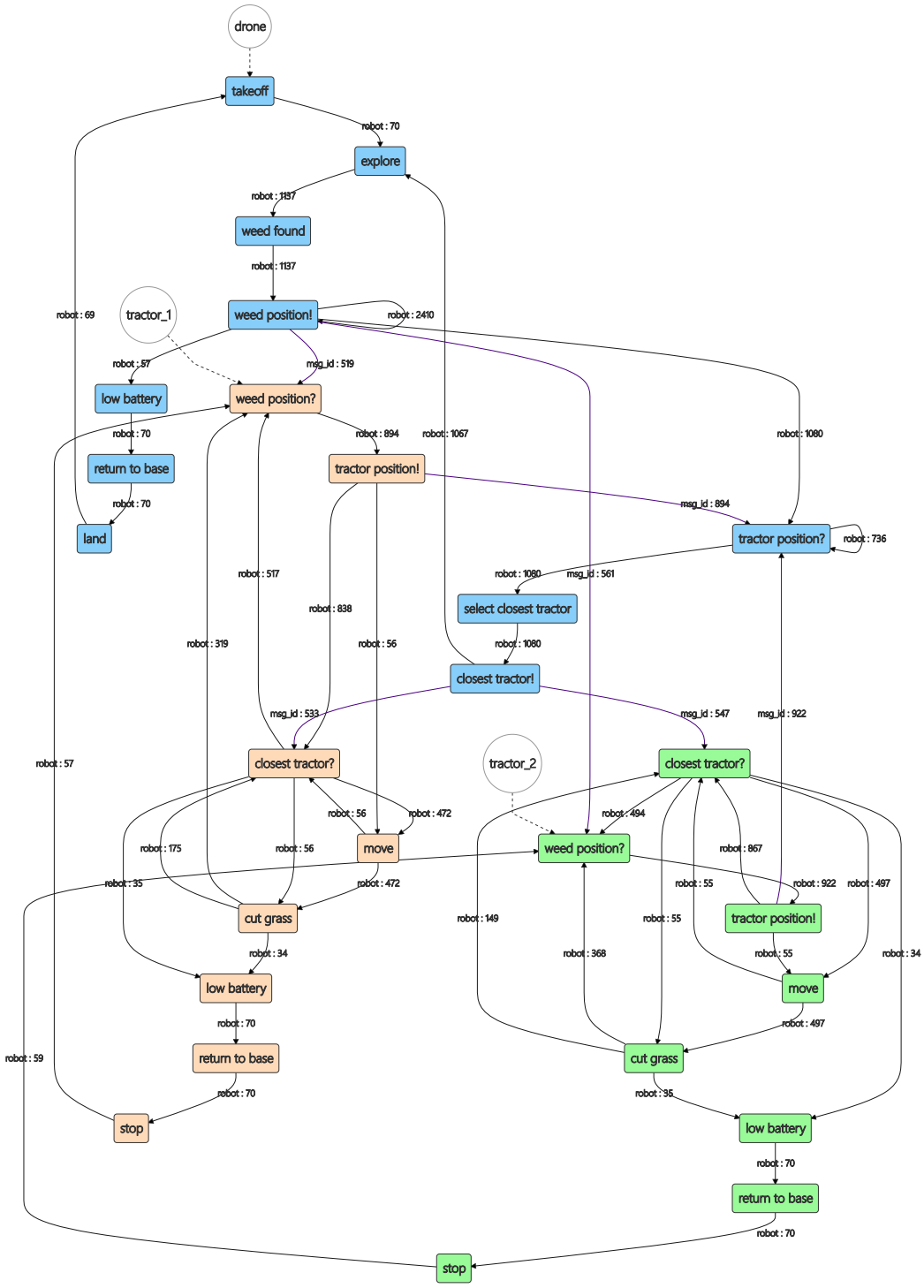


Figure 7.7: EKG of the running scenario

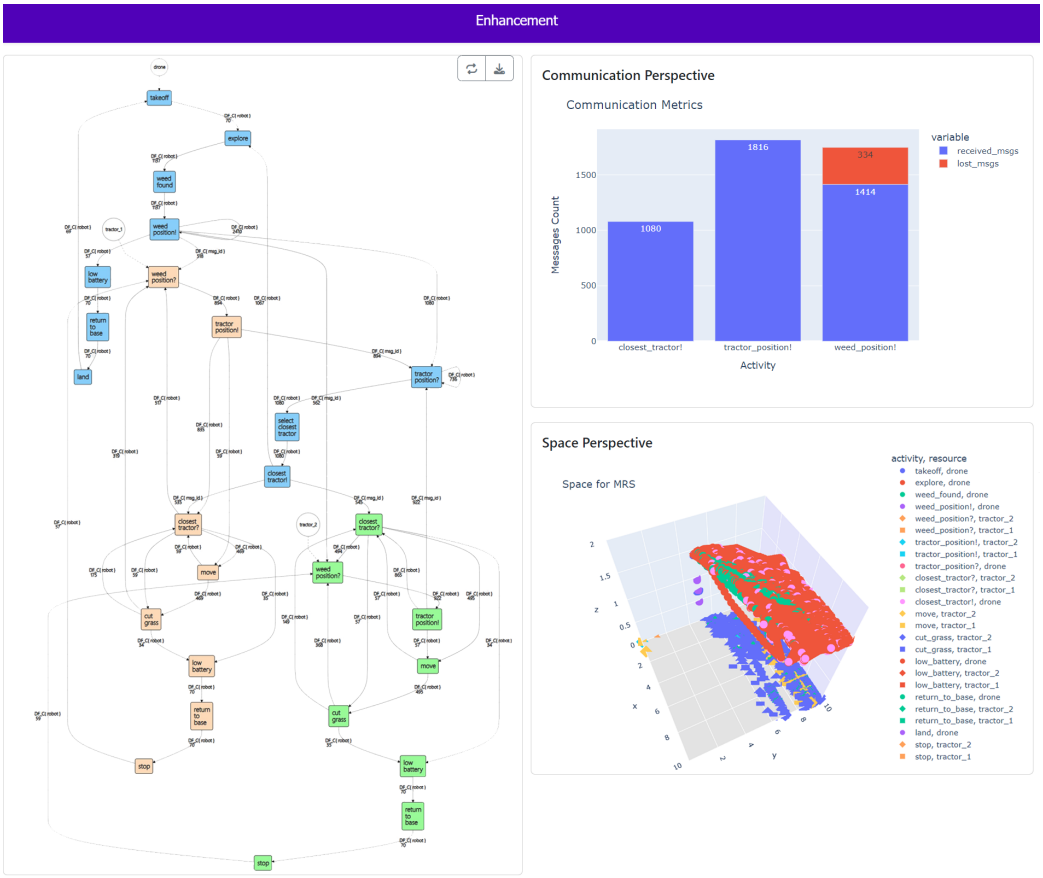


Figure 7.8: Enhancement interface

input. As a result of the drone partial exploration, the tractors' cutting grass operations, depicted in violet, are just focused on part of the field.

For the communication perspective, TALE retrieves data from activities associated with inter-robot communication, exploiting both the message identifier stored in the *msg_id* field and the syntax chosen during the preparation step, i.e., the activities related to the communication end with a *!* if they are sender type and with a *?* if they are receiver type. Therefore, for each unique message identifier value, the set of corresponding events is retrieved to calculate how many messages have been sent in total, and of these how many were lost. The communication plot shows that the 1080 *closest_tractor!* and 1816 *tractor_position!* messages have been shared without any loss. For the *weed_position!*, 1414 messages have been sent by the drone and properly received by at least one of the tractors, whereas 334 messages have been lost. Message losses could have been due to tractors being busy at the moment the drone transmitted the message.

Part IV

Concluding Remarks

CHAPTER 8

Concluding Remarks

The work presented in this thesis tackles the usage of process models within MRSs from two directions. Initially, a top-down approach has been proposed for modeling and enacting an MRS mission via BPMN collaboration diagrams. Then, a bottom-up approach has been proposed to enable the analysis of an MRS mission exploiting process mining techniques. Both approaches are fully automatized exploiting the tools provided by this thesis. Moreover, the approaches have been evaluated in a running scenario of an MRS deployed in a smart agricultural field and executed in a simulation environment.

The rest of the chapter concludes the thesis by presenting the results of the work, the discussion of the given approaches, and also the directions for future work.

8.1 Thesis Results

The top-down approach seeks to (*i*) support the representation of the MRS cooperative behavior using BPMN as a modeling language, (*ii*) enrich the BPMN process model with the information required for the MRS implementation, and (*iii*) assess the usage of BPMN for guiding the MRS execution. This thesis fulfills these objectives by proposing the FAME framework. As a starting point, the framework exploits a selection of BPMN elements and a set of guidelines to guide the modeling of an MRS mission. This allows modeling the behavior of each robot within the MRS and the inter-robot

communication using the BPMN standard as-it-is, i.e., without providing any extension. Once the mission is designed, the collaboration diagram requires a configuration for enriching it with information related to the MRS enactment. This enables the generation of a collaboration diagram that the robots can directly interpret. Indeed, the last phase of FAME defines how to enact the BPMN models to guide the behavior of the robots. Specifically, each robot can directly execute the proposed models without needing a central orchestrator.

The bottom-up approach aims to (*i*) support the extraction of robotic data in a process mining-compatible manner, (*ii*) assess the usage of process mining techniques to discover and analyze the behavior of the MRS, and (*iii*) enhance process mining techniques to deal with multiple perspectives impacting MRS execution. To achieve these objectives, this thesis proposes the TALE methodology enabling both MRS data preparation and mission analysis. At first, TALE facilitates an MRS developer to extract activity-centric robotic event logs at the desired level of granularity and related to multiple perspectives of interest. The produced data are suitable for the application of process mining techniques to analyze the execution of the MRS mission by reconstructing the robots' behavior into a process model. Specifically, TALE supports both the control-flow analysis, exploiting process discovery techniques, and multi-perspective analysis, by enhancing the discovered process with information related to the spatial occupancy and the inter-robot communication perspectives.

The evaluation of the approaches shows the applicability of the proposals in the running scenario. Indeed, all the phases composing the FAME framework have been applied to model and enact the running scenario. Additionally, the framework has been evaluated in a small-scale scenario involving physical robots, comparing the performance of robots utilizing a BPMN engine versus directly compiling the code. The results illustrate that employing a BPMN engine to interpret the mission model had a low impact on robots' performance. Regarding the TALE methodology, activity-centric event logs, enriched with spatial occupancy and inter-robot communication data, have been extracted from the running scenario. These event logs have been used to provide insights into the MRS mission via control-flow and multi-perspective analysis.

8.2 Discussion

As MRSs gain more popularity and BPM techniques are applied in diverse fields, new concepts emerge regarding the approaches examined in this thesis, requiring additional exploration. It is worth noting that, although this thesis focuses on MRSs, the proposed approaches are fully compatible with

single-robot systems. This section discusses the extension of the suggested techniques to a heterogeneous scenario involving both robotic and IoT systems. Additionally, it examines the motivation behind using two different modeling notations for the top-down and bottom-up approaches. Finally, it suggests the potential for integrating the top-down and bottom-up approaches into a unified framework.

8.2.1 On the Integration of MRS with IoT

The application of BPM techniques in the robotic domain enables the modeling and enactment of robots' high-level missions and supports the analysis of the performed mission and different system perspectives. The work presented in this thesis only focuses on the robotic domain albeit it could be declined also in the IoT one. Indeed, with the spread of interconnected devices, novel solutions apply BPM techniques to IoT systems to support the adoption of model-driven development approaches, and the analysis of the system execution by mining the produced logs [45].

Looking at IoT and robotic systems as complementary technologies, their integration aims to combine the facilities provided by the two [82]. The resulting system leverages the capabilities of IoT devices for pervasive sensing and monitoring and the capabilities of robots to produce action, interaction, and autonomous behavior [42]. Indeed, IoT systems offer a comprehensive view of the system state, while robots autonomously interact with the environment, combining their functionalities with data from IoT devices. For instance, in an agricultural field, an IoT system with the capacity to monitor environmental factors, such as weather conditions and temperature, can enhance the efficiency of robotic farming operations by enabling robots to adapt their behaviors in response to real-time environmental data.

In this heterogeneous system, a top-down approach would overcome the necessity for high-level programming skills to manage the complexity of both IoT and robotic systems. This approach would also facilitate the management of message exchanges necessary for coordinating the devices. Considering the integration of BPMN with IoT, the literature proposes different methodologies. IoT-aware business processes [49, 80] aim to design the workflow of a system in which IoT devices are deployed. Specifically, these methodologies do not design the behavior of IoT devices. Instead, they focus on designing the behavior of the overall system, where the flow is guided by data obtained from IoT devices. Differently, other methodologies [31, 88] model the workflow of the devices composing an IoT system, and the interactions with other devices, and with the environment. These models provide a complete view of the system. Notably, these methodologies can be integrated into the FaMe framework to create a cooperative robotic and IoT system, as shown in [23]. Indeed applying the FaMe framework would ease both system

behavior modeling and setting up QoS policies supporting communications requirements. However, although FAME guidelines could offer enough expressiveness to model such a system, the enactment phase must consider the heterogeneous nature of the devices. Extensions in supported technologies and model enactment may be necessary. For instance, while robots can run a BPMN engine, IoT devices with constrained capabilities may necessitate alternative solutions. One potential solution could involve integrating the engine into an additional IoT architectural layer, like a fog node [15], to manage data collection and communication with robots. Additionally, a model-to-code transformation could facilitate deploying the desired behavior on IoT edge devices.

Considering the bottom-up approach, the integration of MRSs and IoT devices necessitates the application of process mining techniques capable of dealing with the huge amount of data generated during their execution [45] and discovering the cooperation between the devices composing the system. Indeed, as in the case of MRSs, IoT systems record data in a peripheral-centric way, keeping track of the continuous data collected or received. As prescribed by the TALE methodology, activity-centric data can be easily extracted from this system by inserting tags. For instance, considering an IoT system for autonomous irrigation a corresponding activity can be tagged as *watering*. During the TALE analysis step, data provided by the IoT devices enriches the data produced by the robots with additional perspectives of the system. This would require an extension of the analysis proposed in the TALE methodology since more information can be connected to the process execution and used for process discovery and enhancement [59].

8.2.2 On the Application of BPMN for MRS Analysis

The proposed top-down and bottom-up approaches exploit process models to meet their objectives. However, two different process representations have been adopted: BPMN for the development and DFG for the analysis.

For designing the FAME framework, BPMN has been chosen as the target notation for the rich set of elements. This notation allows robotic missions to be expressed not only in terms of sequential activities but also to include behavioral choices, parallelism, deviations from the main process, and message exchange. Additionally, the availability of BPMN-engines based on the standard, which can be customized with additional features, makes this notation highly suitable for model-driven solutions. Notably, in this approach, using DFGs for representing system behavior would have been limiting due to their low expressiveness [93]. Indeed, a DFG can represent only sequential activities, therefore concurrency, behavioral choices, and inter-robot interaction could not be represented.

Differently, the TALE methodology proposes to analyze the behavior of

robots using DFGs. Indeed, the TALE methodology aims to provide robotic experts with analysis techniques from the BPM discipline. Therefore, using a DFG, i.e., the simplest representation of process models, allows robotic behavior to be represented in an immediate and intuitive way, even for those unfamiliar with the BPM discipline. Notably, BPMN collaborations can also be discovered using the event logs generated through the TALE-preparation step. Performing process discovery on MRS data would involve discovering a BPMN process model that depicts the overall mission with various elements, ideally, those considered during the FAME modeling phase. However, existing techniques do not cover discovering all the BPMN elements that define an MRS mission, particularly signals and event sub-processes. In this context, a preliminary effort has been made in [24], by proposing a technique for discovering BPMN collaboration diagrams where participants interact via signals. However, this technique only discovers the process depicting the behavior of each robot, without distinguishing process deviations as event sub-processes. To also express event sub-processes, a potential solution could be combining this technique with approaches capable of discovering hierarchical process models like the one presented in [18].

8.2.3 On Merging Top-Down and Bottom-Up

Although the top-down and bottom-up approaches are presented separately, they could be integrated to create a comprehensive solution capable of facilitating the design and management of MRS missions, optimizing operations, and enhancing the analysis of activities [104].

As proposed in the FAME framework, BPMN collaboration diagrams are suitable to model an MRS mission and can be used with a BPMN engine for direct system enactment. Following the TALE methodology, each activity in the model can be treated as a tag. As the MRS execution is guided by BPMN, tokens entering and exiting activities mark the start and completion of those activities, respectively. Indeed, integrating the tag concept within the FAME enactment phase enables the engines to generate activity-centric data, while the system captures multi-perspective data. By leveraging the TALE methodology, this data can be further processed to obtain activity-centric event logs enhanced with relevant perspectives. The produced event log can serve for performing real-time and post-mortem system analysis as well as simulations and predictions of future process behavior under changed conditions [5]. Analysis techniques proposed by TALE enable the MRS analysis after system execution, however, the integration of further techniques would provide additional insights into system execution. MRSs can leverage streaming process mining techniques for analyzing continuous data streams [13]. Additionally, predictive analysis can be exploited to anticipate future process evolution in response to environmental changes [30].

The insights obtained via process mining-based analysis can drive the (re-)modeling of the collaboration diagram, to better reflect MRS requirements, solve potential issues, or enhance the system performances. This creates a cyclic integration between the top-down and bottom-up approaches.

The merge of the top-down and the bottom-up approaches can enable the creation of a Digital Process Twin (DPT) for MRSs. Specifically, a DPT is an emerging concept that declines digital twins in process-driven systems to provide valuable insights into system operations. This enables decision-making, performance optimization, and prediction of potential outcomes based on data from both the process model and the physical system [20]. A DPT is the digital replica of the business processes of a system and it should comprise the representation of multiple and interdependent processes, models showing different aspects of the system, analysis capabilities, and system adaptation features based on experience [5]. Consequently, DPTs aim to assist developers, analysts, and end-users to understand system behavior and facilitate data interpretation and analysis.

In addition to the facilities provided by the FAME framework and the TALE methodology, to show different aspects of the system execution, the DPT can both leverage the BPMN semantics to show the process execution, and 3D simulators to visually represent the MRS overall behavior [21]. Starting from the collaboration diagram used to represent and drive the system execution, it is possible to highlight the activities performed by robots within the process. Indeed, considering the concept of tokens, the distribution of tokens across process elements indicates an execution state. Several tools [79, 19, 103] implement token simulator to show the evolution of the system, and effectively track the interplay between control flow, data handling, and messages exchange, to identify undesired executions. Therefore, the integration of a token simulator in the DPT enables debugging facilities on the top of a BPMN diagram. This allows checking if the designer's intentions are reflected in how the process and the robots' behavior are performed [3]. Additionally, since an MRSs is capable of moving and interacting with the environment, the DPT should also integrate a realistic view of the system. To this aim, a 3D environment, such as the one provided by Gazebo, can be used to visualize how the devices act and interact. Combining this visualization with token simulation enables a concurrent view of the execution of the system behavior and the visualization of the corresponding states of the involved devices.

Finally, the general concept of digital twin states that any changes to it must be reflected in the actual behavior of the physical counterpart [84]. This requirement can be achieved by developing and exploiting a process adaptation mechanism capable of interpreting analysis results and refining the process model. Once the model has been changed, its updated version

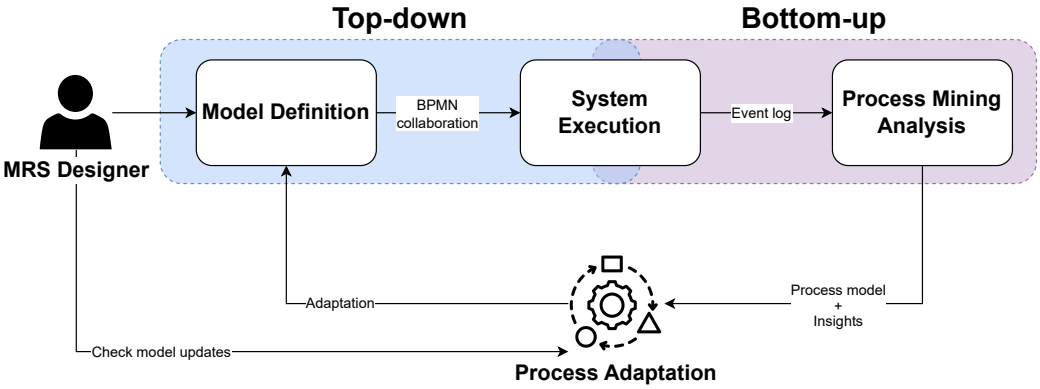


Figure 8.1: MRS digital process twin proposal

can be deployed into the BPMN-driven system. This approach allows for enhancing the devices' behavior, according to the evolution of the system. Notably, to achieve a full integration between the top-down and bottom-up approaches, a process adaptation module should be developed to enable robots with adaptation capabilities. At the same time, also the FAME framework should be enriched with adaptation capabilities to efficiently and safely deal with runtime model updates. Therefore, the proposed DPT can fall under the umbrella of *models@run.time* [9]. This concept leverages models to provide insight into a running software system, allowing for tasks such as monitoring, analysis, rectification of design errors, exploration of new design alternatives, and adaptation of the system through a direct connection between a model and the system. Considering the DPT as a model@run.time, the FAME framework would facilitate guiding system execution via a process model, while the TALE methodology would enable system analysis. Indeed, various techniques from the process mining domain could be employed. Event logs produced by the TALE methodology and process discovery techniques could aid in model transformations to transition from one model to another. Additionally, other process mining techniques could support the model@run.time, such as enabling runtime system monitoring, extracting insights from the model, and predicting future system states.

Summing up, as proposed in Figure 8.1, a DPT can be seen as the result of merging the top-down and bottom-up approaches. Specifically, the BPMN collaboration diagram is used to guide the behavior of the MRS. During the system execution, exploiting the TALE methodology, event logs are generated to perform process mining-based system analysis. Finally, based on the insights provided during the analysis, a process adaptation step, supported by the MRS designer supervision, can be performed to re-design the model defining the mission and enact the improved MRS.

8.3 Future Works

In future works, the first objective is to improve the proposed approaches by implementing them in real-world MRS use cases and gathering feedback from robotic experts. By applying the FAME framework to enact an MRS deployed in a real-world scenario and employing the TALE methodology to analyze system performance, it becomes possible to incorporate new features based on observed outcomes and expert opinions. This objective aims to bring the approaches and, consequently, BPM techniques closer to the robotic field.

Regarding the top-down approach, one of the future aims is to enhance the FAME framework by incorporating a model analysis phase to check the correctness and to identify anomalies and security concerns in the collaboration diagrams. For the other phases, the intention is to enrich the configuration phase by adding the possibility of selecting the device target frameworks, such as the OROCOS robotic framework [12] or the ThingsBoard IoT framework [86], and to enhance the enactment phase by enabling direct model enactment or model-to-code translation. This extension aims to make the framework applicable to systems composed of heterogeneous devices, as discussed in Section 8.2.1.

Concerning the bottom-up approach, the novelty of applying process mining in the robotic world opens the way for many future developments. In the traditional process mining context, event logs extracted with the TALE methodology can be used with existing process discovery algorithms. The first objective is to evaluate the suitability of approaches aimed at discovering collaborative process models of distributed systems, such as the ones proposed in [25, 87, 106]. Indeed, these approaches are capable of discovering high-level process models like BPMN or Petri nets, which can express choices and concurrency. Differently, in object-centric process mining, the operations that can be done on an EKG to discover the process model will be extended. Specifically, the development of aggregation functions capable of merging multiple system perspectives can enhance the exploration of the graph across different aggregation levels. This enables analysts to perform personalized and interactive analyses. In terms of process enhancement, leveraging or deploying approaches for discovering multi-perspective processes [59] can be useful in improving the MRS process model with information that has guided robots' decisions.

Finally, considering both the top-down and bottom-up approaches, a fundamental aspect lies in facilitating and implementing runtime adaptation of the behavior. Such adaptations demand a comprehensive understanding of the mechanisms enabling real-time changes and their deployment, raising questions about the entities responsible for orchestrating these modifications.

Integration of such capabilities into the top-down approach necessitates a complex solution, comprising runtime change management while ensuring operational continuity. On the bottom-up side, adaptability needs the development of new process discovery techniques to understand whether a robot has executed an adaptation. This includes not only detecting such adaptations but also evaluating their consequences impacting MRS functionalities and performances.

Bibliography

[1] Imad Afyouni, Cyril Ray, and Claramunt Christophe. Spatial models for context-aware indoor navigation systems: A survey. *Journal of Spatial Information Science*, 1(4):85–123, 2012.

[2] Afsoon Afzal, Claire Le Goues, Michael Hilton, and Christopher Steven Timperley. A study on challenges of testing robotic systems. In *Software Testing, Validation and Verification*, pages 96–107. IEEE, 2020.

[3] Thomas Allweyer and Stefan Schweitzer. A tool for animating bpmn token flow. In *Business Process Model and Notation*, pages 98–106. Springer, 2012.

[4] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo. Automated discovery of process models from event logs: review and benchmark. *Transactions on knowledge and data engineering*, 31(4):686–705, 2018.

[5] Markus C Becker and Brian T Pentland. Digital twin of an organization: Are you serious? In *Business Process Management Workshops*, pages 243–254. Springer, 2021.

[6] Abdenour Benkrid, Abdelaziz Benallegue, and Noura Achour. Multi-robot coordination for energy-efficient exploration. *Journal of Control, Automation and Electrical Systems*, 30(6):911–920, 2019.

[7] Alessandro Berti and Wil MP van Der Aalst. Extracting multiple viewpoint models from relational databases. In *International Symposium*

on Data-Driven Process Discovery and Analysis, pages 24–51. Springer, 2018.

[8] Alessandro Berti and Wil MP van der Aalst. OC-PM: analyzing object-centric event logs and process models. *International Journal on Software Tools for Technology Transfer*, 25(1):1–17, 2023.

[9] Gordon Blair, Nelly Bencomo, and Robert B France. Models@run.time. *Computer*, 42(10):22–27, 2009.

[10] Khalid Bourr, Flavio Corradini, Sara Pettinari, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. Disciplined use of BPMN for mission modeling of Multi-Robot Systems. In *Proceedings of the Forum at Practice of Enterprise Modeling*, volume 3045 of *CEUR Workshop Proceedings*, pages 1–10. CEUR-WS.org, 2021.

[11] Darko Bozhinoski, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Massimo Tivoli. FLYAQ: Enabling non-expert users to specify and generate missions of autonomous multicopters. In *ASE*, pages 801–806, 2015.

[12] Herman Bruyninckx. Open robot control software: the OROCOS project. In *International conference on robotics and automation*, volume 3, pages 2523–2528. IEEE, 2001.

[13] Andrea Burattin. Streaming process mining. In *Process Mining Handbook*, volume 448 of *LNBIP*, pages 349–372. Springer, 2022.

[14] Josep Carmona, Boudewijn van Dongen, and Matthias Weidlich. Conformance checking: foundations, milestones and challenges. In *Process Mining Handbook*, pages 155–190. Springer, 2022.

[15] Mung Chiang and Tao Zhang. Fog and iot: An overview of research opportunities. *Internet of things journal*, 3(6):854–864, 2016.

[16] Stefan-Gabriel Chitic. *Middleware and programming models for multi-robot systems*. PhD thesis, INSA Lyon, 2018.

[17] Federico Ciccozzi, Davide Di Ruscio, Ivano Malavolta, and Patrizio Pelliccione. Adopting MDE for Specifying and Executing Civilian Missions of Mobile Multi-Robot Systems. *IEEE Access*, 4:6451–6466, 2016.

[18] Raffaele Conforti, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Bpmn miner: Automated discovery of bpmn process models with hierarchical structure. *Information Systems*, 56:284–303, 2016.

[19] Flavio Corradini, Chiara Muzi, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. Mida: Multiple instances and data animator. In *BPM (Dissertation/Demos/Industry)*, pages 86–90. CEUR-WS.org, 2018.

[20] Flavio Corradini, Sara Pettinari, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. An approach to support digital process twin. In *Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress*, pages 1–4. IEEE, 2022.

[21] Flavio Corradini, Sara Pettinari, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. Executable digital process twins: Towards the enhancement of process-driven systems. *Big Data Cognitive Computing*, 7(3):139, 2023.

[22] Flavio Corradini, Sara Pettinari, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. A methodology for the analysis of robotic systems via process mining. In *Enterprise Design, Operations, and Computing*, volume 14367 of *LNCS*, pages 117–133. Springer, 2023.

[23] Flavio Corradini, Sara Pettinari, Barbara Re, Luca Ruschioni, and Francesco Tiezzi. Enhancing compatibility in qos communication for the internet of robotic things. In *International Conference on Conceptual Modeling: ER Forum*, volume 3618 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[24] Flavio Corradini, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. A technique for collaboration discovery. In *Business Process Modeling, Development and Support*, volume 450 of *LNBIP*, pages 63–78. Springer, 2022.

[25] Flavio Corradini, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. A technique for collaboration discovery. In *International Conference on Business Process Modeling, Development and Support*, pages 63–78. Springer, 2022.

[26] Edson de Araújo Silva, Eduardo Valentin, Jose Reginaldo Hughes Carvalho, and Raimundo da Silva Barreto. A survey of model driven engineering in robotics. *Journal of Computer Languages*, 62:101021, 2021.

[27] Jean-Pierre de la Croix and Grace Lim. Event-driven modeling and execution of robotic activities and contingencies in the Europa lander mission concept using BPMN. In *i-SAIRAS*. ESA, 2020.

[28] Massimiliano de Leoni. Foundations of process enhancement. In *Process Mining Handbook*, pages 243–273. Springer, 2022.

[29] Clarence W de Silva. Some issues and applications of multi-robot cooperation. In *Computer Supported Cooperative Work in Design*, pages 2–2. IEEE, 2016.

[30] Chiara Di Francescomarino and Chiara Ghidini. Predictive process monitoring. In *Process Mining Handbook*, volume 448 of *LNBIP*, pages 320–346. Springer, 2022.

[31] Dulce Domingos, Ana Respício, Francisco Martins, and Breno Melo. Automatic decomposition of iot aware business processes–a pattern approach. *Procedia Computer Science*, 164:313–320, 2019.

[32] Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo A Reijers, et al. *Fundamentals of business process management*, volume 2. Springer, 2018.

[33] Stefan Esser and Dirk Fahland. Multi-dimensional event data in graph databases. *Journal on Data Semantics*, 10(1-2):109–141, 2021.

[34] Dirk Fahland. Process mining over multiple behavioral dimensions with event knowledge graphs. In *Process Mining Handbook*, pages 274–319. Springer, 2022.

[35] Maksym Figat and Cezary Zielinski. Robotic system specification methodology based on hierarchical petri nets. *IEEE Access*, 8:71617–71627, 2020.

[36] Maksym Figat and Cezary Zieliński. Parameterised robotic system meta-model expressed by Hierarchical Petri nets. *Robotics and Autonomous Systems*, 150:103987, 2022.

[37] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *International conference on management of data*, pages 1433–1445, 2018.

[38] Ramkumar Gandhinathan and Lentin Joseph. *ROS Robotics projects: build and control robots powered by the Robot Operating System, machine learning, and virtual reality.* Packt Publishing Ltd, 2019.

[39] Avinash Gautam and Sudeept Mohan. A review of research in multi-robot systems. In *International Conference on Industrial and Information Systems*, pages 1–5. IEEE, 2012.

[40] Matthias Geiger, Simon Harrer, Jörg Lenhard, Mathias Casar, Andreas Vorndran, and Guido Wirtz. Bpmn conformance in open source engines. In *Symposium on Service-Oriented System Engineering*, pages 21–30. IEEE, 2015.

[41] Alexandre Goossens, Johannes De Smedt, Jan Vanthienen, and Wil MP van der Aalst. Enhancing data-awareness of object-centric event logs. In *International Conference on Process Mining*, pages 18–30. Springer, 2022.

[42] Luigi Alfredo Grieco, Alessandro Rizzo, Simona Colucci, Sabrina Sicari, Giuseppe Piro, Donato Di Paola, and Gennaro Boggia. IoT-aided robotics applications: Technological implications, target domains and open issues. *Computer Communications*, 54:32–47, 2014.

[43] José Guia, Valéria Gonçalves Soares, and Jorge Bernardino. Graph databases: Neo4j analysis. In *ICEIS*, pages 351–356, 2017.

[44] James Harbin, Simos Gerasimou, Nicholas Matragkas, Athanasios Zolotas, and Radu Calinescu. Model-driven simulation-based analysis for multi-robot systems. In *Model Driven Engineering Languages and Systems*, pages 331–341. IEEE, 2021.

[45] Christian Janiesch, Agnes Koschmider, Massimo Mecella, Barbara Weber, Andrea Burattin, Claudio Di Ciccio, Giancarlo Fortino, Avigdor Gal, Udo Kannengiesser, Francesco Leotta, et al. The internet of things meets business process management: a manifesto. *IEEE Systems, Man, and Cybernetics Magazine*, 6(4):34–44, 2020.

[46] Jose-Fernando Jimenez, Gabriel Zambrano-Rey, Santiago Aguirre, and Damien Trentesaux. Using process-mining for understating the emergence of self-organizing manufacturing systems. *IFAC-PapersOnLine*, 51(11):1618–1623, 2018.

[47] Homayun Kabir, Mau-Luen Tham, and Yoong Choon Chang. Internet of robotic things for mobile robots: Concepts, technologies, challenges, applications, and future directions. *Digital Communications and Networks*, 9(6):1265–1290, 2023.

[48] Haresh Karnan, Anirudh Nair, Xuesu Xiao, Garrett Warnell, Sören Pirk, Alexander Toshev, Justin Hart, Joydeep Biswas, and Peter Stone. Socially compliant navigation dataset (scand): A large-scale dataset of demonstrations for social navigation. *Robotics and Automation Letters*, 2022.

[49] Yusuf Kirikkayis, Florian Gallik, and Manfred Reichert. Towards a comprehensive bpmn extension for modeling iot-aware processes in business process models. In *International Conference on Research Challenges in Information Science*, pages 711–718. Springer, 2022.

[50] Mateja Kocbek, Gregor Jost, Marjan Hericko, and Gregor Polancic. Business process model and notation: The current state of affairs. *Computer Science and Information Systems*, 12(2):509–539, 2015.

[51] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *International Conference on Intelligent Robots and Systems*, volume 3, pages 2149–2154. IEEE, 2004.

[52] Anis Koubâa et al. *Robot Operating System (ROS).*, volume 1. Springer, 2017.

[53] Dániel Kozma, Pál Varga, and Felix Larrinaga. Data-driven Workflow Management by utilising BPMN and CPN in IIoT Systems with the Arrowhead Framework. In *International Conference on Emerging Technologies and Factory Automation*, pages 385–392. IEEE, 2019.

[54] Otsu Kyohei, Tepsuporn Scott, Thakker Rohan, Tiago Vaquero, Edlund Jeffrey, Walsh William, Miles Gregory, Heywood Tristan, Wolf Michael, and Agha-Mohammadi Ali-Akbar. Supervised Autonomy for Communication-degraded Subterranean Exploration by a Robot Team. In *Aerospace Conference*, pages 1–9. IEEE, 2020.

[55] Volodymyr Leno, Artem Polyvyanyy, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. Robotic process mining: vision and challenges. *Business & Information Systems Engineering*, 63:301–314, 2021.

[56] Joaquín López, Pablo Sánchez-Vilariño, Rafael Sanz, and Enrique Paz. Implementing autonomous driving behaviors using a message driven petri-net framework. *Sensors*, 20:449, 2020.

[57] Matt Luckcuck, Marie Farrell, Louise A Dennis, Clare Dixon, and Michael Fisher. Formal specification and verification of autonomous robotic systems: A survey. *Computing Surveys*, 52(5):1–41, 2019.

[58] Juergen Mangler, Joscha Grüger, Lukas Malburg, Matthias Ehrendorfer, Yannis Bertrand, Janik-Vasily Benzin, Stefanie Rinderle-Ma, Estefania Serral Asensio, and Ralph Bergmann. Datastream xes extension: embedding iot sensor data into extensible event stream logs. *Future Internet*, 15(3):109, 2023.

[59] Felix Mannhardt. *Multi-perspective process mining*. PhD thesis, Eindhoven University of Technology, 2018.

[60] Felix Mannhardt, Massimiliano De Leoni, Hajo A Reijers, Wil MP Van Der Aalst, and Pieter J Toussaint. From low-level events to activities-a pattern-based approach. In *Business Process Management*, pages 125–141. Springer, 2016.

[61] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ROS2. In *International Conference on Embedded Software*, pages 1–10, 2016.

[62] Petra Mazdin and Bernhard Rinner. Distributed and communication-aware coalition formation and task assignment in multi-robot systems. *IEEE Access*, 9:35088–35100, 2021.

[63] Claudio Menghi, Christos Tsigkanos, Patrizio Pelliccione, Carlo Ghezzi, and Thorsten Berger. Specification patterns for robotic missions. *Transactions on Software Engineering*, 47(10):2208–2224, 2019.

[64] Sonja Meyer, Andreas Ruppen, and Carsten Magerkurth. Internet of things-aware process modeling: integrating iot devices as business process resources. In *Advanced Information Systems Engineering*, pages 84–98. Springer, 2013.

[65] Takeshi Morita and Takahira Yamaguchi. Generating ROS Codes from User-Level Workflow in PRINTEPS. In *Domain-Specific Conceptual Modeling: Concepts, Methods and ADOxx Tools*, pages 435–455. Springer, 2022.

[66] Turcanu Cristina Nicoleta. Process mining on a robotic mechanism. In *International Conference on Software Testing, Verification and Validation Workshops*, pages 205–212. IEEE, 2021.

[67] Arne Nordmann, Nico Hochgeschwender, and Sebastian Wrede. A survey on domain-specific languages in robotics. In *Simulation, modeling, and programming for autonomous robots*, pages 195–206. Springer, 2014.

[68] OMG. Business Process Model and Notation (BPMN) v. 2.0, 2011.

[69] OMG. Data Distribution Service (DDS) v. 1.4, 2015.

[70] IEEE Task Force on Process Mining. Standard lifecycle transition model. https://www.tf-pm.org/resources/xes-standard/about-xes/standard-extensions/lifecycle/standard.

[71] François Pomerleau, Ming Liu, Francis Colas, and Roland Siegwart. Challenging data sets for point cloud registration algorithms. *The International Journal of Robotics Research*, 31(14):1705–1711, 2012.

[72] Partha Pratim Ray. Internet of robotic things: Concept, technologies, and challenges. *IEEE access*, 4:9489–9500, 2016.

[73] Rafael Rey, Marco Corzetto, Jose Antonio Cobano, Luis Merino, and Fernando Caballero. Human-robot co-working system for warehouse automation. In *International Conference on Emerging Technologies and Factory Automation*, pages 578–585. IEEE, 2019.

[74] Juan Jesús Roldán, Jaime del Cerro, David Garzón-Ramos, Pablo Garcia-Aunon, Mario Garzón, Jorge De León, and Antonio Barrientos. Robots in agriculture: State of art and practical experiences. *Service robots*, pages 67–90, 2018.

[75] Juan Jesús Roldán, Miguel A Olivares-Méndez, Jaime del Cerro, and Antonio Barrientos. Analyzing and improving multi-robot missions by using process mining. *Autonomous Robots*, 42(6):1187–1205, 2018.

[76] Anne Rozinat, Stefan Zickler, Manuela Veloso, Wil MP van der Aalst, and Colin McMillen. Analyzing multi-agent activity logs using process mining techniques. *Distributed Autonomous Robotic Systems*, pages 251–260, 2009.

[77] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2):1729881419839596, 2019.

[78] Yuvraj Sahni, Jiannong Cao, and Shan Jiang. Middleware for multi-robot systems. In Habib M. Ammari, editor, *Mission-Oriented Sensor Networks and Systems: Art and Science - Volume 2: Advances*, volume 164, pages 633–673. Springer, 2019.

[79] SAP. Signavio. https://www.signavio.com.

[80] Stefan Schönig, Lars Ackermann, Stefan Jablonski, and Andreas Ermer. Iot meets bpm: a bidirectional communication architecture for iot-aware process execution. *Software and systems modeling*, 19:1443–1459, 2020.

[81] Ronny Seiger, Marco Franceschetti, and Barbara Weber. An interactive method for detection of process activity executions from IoT data. *Future Internet*, 15(2):77, 2023.

[82] Pieter Simoens, Mauro Dragone, and Alessandro Saffiotti. The internet of robotic things: A review of concept, added value and applications. *International Journal of Advanced Robotic Systems*, 15(1):1729881418759424, 2018.

[83] Manisha Singh and Gaurav Baranwal. Quality of Service (QoS) in internet of things. In *International Conference On Internet of Things: Smart Innovation and Usages*, pages 1–6. IEEE, 2018.

[84] Maulshree Singh, Evert Fuenmayor, Eoin P Hinchy, Yuansong Qiao, Niall Murray, and Declan Devine. Digital twin: Origin to future. *Applied System Innovation*, 4(2):36, 2021.

[85] Niek Tax, Natalia Sidorova, Reinder Haakma, and Wil MP van der Aalst. Event abstraction for process mining using supervised learning techniques. In *SAI Intelligent Systems Conference*, pages 251–269. Springer, 2018.

[86] ThingsBoard. ThingsBoard - Open-source IoT Platform. https://thingsboard.io/.

[87] Andrei Tour, Artem Polyvyanyy, Anna Kalenkova, and Arik Senderovich. Agent miner: an algorithm for discovering agent systems from event data. In *International Conference on Business Process Management*, pages 284–302. Springer, 2023.

[88] Pedro Valderas, Victoria Torres, and Estefanía Serral. Modelling and executing iot-enhanced business processes through bpmn and microservices. *Journal of Systems and Software*, 184:111139, 2022.

[89] Antti Valmari. The state explosion problem. In *Advanced Course on Petri Nets*, pages 429–528. Springer, 1996.

[90] Wil MP van der Aalst. Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems*, 3(2):1–17, 2012.

[91] Wil MP Van der Aalst. Business process management: a comprehensive survey. *International Scholarly Research Notices*, 2013, 2013.

[92] Wil MP van der Aalst. *Process Mining Data Science in Action*. Springer, 2016.

[93] Wil MP Van Der Aalst. A practitioner's guide to process mining: Limitations of the directly-follows graph. *Procedia Computer Science*, 164:321–328, 2019.

[94] Wil MP van der Aalst. Foundations of process discovery. In *Process Mining Handbook*, pages 37–75. Springer, 2022.

[95] Wil MP van der Aalst. Process mining: a 360 degree overview. In *Process Mining Handbook*, pages 3–34. Springer, 2022.

[96] Wil MP van der Aalst. Experiences from the internet-of-production: Using "data-models-in-the-middle" to fight complexity and facilitate reuse. In *International Conference on Business Process Management*, pages 87–91. Springer, 2023.

[97] Maikel L Van Eck, Natalia Sidorova, and Wil MP Van der Aalst. Enabling process mining on sensor data from smart products. In *International Conference on Research Challenges in Information Science*, pages 1–12. IEEE, 2016.

[98] Rogier M Van Eijk, Frank S De Boer, Wiebe Van Der Hoek, and John-Jules Ch Meyer. Process algebra for agent communication: A general semantic approach. *Communication in Multiagent Systems: Agent Communication Languages and Conversation Policies*, pages 113–128, 2003.

[99] Sebastiaan J van Zelst, Felix Mannhardt, Massimiliano de Leoni, and Agnes Koschmider. Event abstraction in process mining: literature review and taxonomy. *Granular Computing*, 6:719–736, 2021.

[100] Mallikarjuna Vayugundla, Florian Steidle, Michal Smisek, Martin J Schuster, Kristin Bussmann, and Armin Wedler. Datasets of long range navigation experiments in a moon analogue environment on mount etna. In *ISR 2018; 50th International Symposium on Robotics*, pages 1–7. VDE, 2018.

[101] Janardan Kumar Verma and Virender Ranga. Multi-robot coordination analysis, taxonomy, challenges and future scope. *Journal of intelligent & robotic systems*, 102(1):10, 2021.

[102] Ovidiu Vermesan, Arne Bröring, Elias Tragos, Martin Serrano, Davide Bacciu, Stefano Chessa, Claudio Gallicchio, Alessio Micheli, Mauro Dragone, Alessandro Saffiotti, et al. Internet of robotic things–converging sensing/actuating, hyperconnectivity, artificial intelligence and iot platforms. In *Cognitive Hyperconnected Digital Transformation*, pages 97–155. River Publishers, 2022.

[103] Visual Paradigm. Visual Paradigm, Business process design with powerful BPMN software. https://www.visual-paradigm.com/features/bpmn-diagram-and-tools.

[104] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures*. Springer, 2019.

[105] Robert Woitsch, Wilfrid Utz, Anna Sumereder, Bernhard Dieber, Benjamin Breiling, Laura Crompton, Michael Funk, Karin Bruckmüller, and Stefan Schumann. Collaborative model-based process assessment for trustworthy ai in robotic platforms. In *Society 5.0*, pages 163–174. Springer International Publishing, 2021.

[106] Qingtian Zeng, Sherry X Sun, Hua Duan, Cong Liu, and Huaiqing Wang. Cross-organizational collaborative workflow mining from a multi-source log. *Decision support systems*, 54(3):1280–1301, 2013.