Received May 23, 2022, accepted May 31, 2022, date of publication June 6, 2022, date of current version June 10, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3180547

Middleware Control Systems Design and Analysis Using Message Interpreted Petri Nets (MIPN)

JOAQUÍN LÓPEZ[1], ALEJANDRO SANTANA-ALONSO[1], AND DIEGO PÉREZ LOSADA[2]
[1]System Engineering and Automation Department, EEI, University of Vigo, 36310 Vigo, Spain
[2]Robotics and Control Unit, AIMEN Technology Centre, 36418 O Porriño, Spain

Corresponding author: Joaquín López (joaquin@uvigo.es)

This work was supported in part by the Spanish ''Ministerio de Ciencia e Innovación'' through the ''Centro para el Desarrollo Tecnológico Industrial'' (CDTI) by the Project ''5R—Cervera network of robotic technologies in smart manufacturing'' under Grant EXP00139978CER-20211007.

ABSTRACT Many distributed frameworks use a message-oriented middleware to interchange information among several independent distributed modules. Those modules make up complex systems implementing basic actions and reporting events about their state. This paper introduces the Message Interpreted Petri Net (MIPN) model to design, analyze, and execute the central control of these middleware systems.
The MIPN is a new Petri net extension that adds message-based high-level information communications and hierarchic capabilities. It also contributes to the definition and study of new properties such as terminability for the hierarchy-wide analysis of a system. Special attention is given to the analyzability of the model. Useful relations between the individual properties of each MIPN and the global properties of a hierarchic MIPNs system are extracted through a mathematical analysis of the model. The goal is to analyze each net separately and then build up the properties of the whole system. This results in a great aid for the programmer and optimizes the development process.
This paper also shows the actual integration of this new MIPN model in different robot control frameworks to design, analyze, execute, monitor, log, and debug tasks in such heterogeneous systems. Finally, some applications created with this framework in the fields of robotics, autonomous vehicles, and logistics are also presented.

INDEX TERMS Middleware control systems, petri nets, robotics, distributed control systems, inter-process communication systems.

I. INTRODUCTION

A. RELATED WORK AND MOTIVATION

Model-driven engineering (MDE) is a well-established approach in software control development [1] where abstraction is used to model systems. These models can be used to implement the software that controls the system and also for documenting purposes [2]. They can be created using domain-specific languages (DSLs) specialized for their particular application domain [3]. Many DSL are based on Petri nets [4], which embody a well known and widely used tool for modeling control algorithms in many fields of application. For example, industrial processes with high demanding environments, like Flexible Manufacturing Systems (FMS) [5] or automated warehouses [6]. It is also commonly used in traffic networks [7], resource optimization in business processes [8], predicting availability levels of photovoltaic power generation systems [9], and supply chain risk management [10]. Despite the versatility of the model, in some cases, the standard definition of PN has been extended to add functionality to fulfill further requirements or to optimize the model towards certain applications. Among a large number of extensions, we can find some well-known models such as Hierarchical PNs [11], [12], Workflow-nets (WF-nets) [13], Coloured PNs [14], or timed PNs [15], just to name a few.

Basic PNs do not explicitly model the relation between the control system itself and its environment. To solve that, Moalla introduced the Synchronized PNs (SPNs) [16] and afterward König and Quäck developed it into the Interpreted PNs (IPNs[1]) [18], providing the means to model systems

The associate editor coordinating the review of this manuscript and approving it for publication was Zhiwu Li.

[1]English IPN notation was taken from [17].

where the relationship between the control logic and its environment is based on events. IPNs has been widely used to model discrete event systems and many works have been published worldwide since its definition [19]. Automation Petri Nets (APN) [20] are also extensions to include actions and sensor readings as formal structures within the net. They were specifically designed for manufacturing systems to accommodate impulse and level actions at places and leading-edge, falling-edge, and level signals of sensors at transitions. Later, the Signal Interpreted PN (SIPN) [17] extension was proposed as an evolution of the previous IPN and differs in that the influence of the environment is based on signals instead of events. The SIPN model allows a more precise way to model inputs and outputs, specially thought for PLC programming, a field where it has been extensively applied [21]. SIPN also differs from IPN in that several signals can change simultaneously, while events occur one at a time. In this case, MIPN is closer to IPN, because messages are processed one at a time.

Events, sensor readings and signals of IPNs, APNs and SIPNs are very useful in many situations, as in PLC programming. However, there is also a wide variety of systems that surpass the definition of both and cannot be easily modeled just with events or signals. That is the case of distributed systems, where the coupling between the control and environment is carried out via some communication mechanism. Most of the distributed control frameworks are based on a middleware layer with several independent modules. These modules are usually connected with different inter-process communication mechanisms using the publish/subscribe paradigm. This use of messages instead of events (IPNs), sensor readings and actions (APNs) or signals (SIPNs) establishes a significant difference with the previous models. Also, the correct management of messages is a critical feature that defines and conditions the functioning of these systems. Such differences were made clear while developing a message-based multi-robot framework called **RIDE** (**R**obotic **I**ntegrated **D**evelopment **E**nvironment) [22], [23]. In RIDE we first tried using the IPN model [24], but finally, we stated that the complexities of message-based communications and practical needs exceed the modeling capabilities of the current event-based and signal-based models. For example, each message type is defined by the programmer with its own complex data structures. The middleware may use queues, so no messages are lost and can be processed in order according to some communication data. Messages can also add some time-related restrictions not available with the previous models. Finally, due to the complexity of most of these distributed systems, a hierarchic modular model is required.

Beyond the modeling capabilities of PNs, there is a more practical interest to automated code generation [25]. In that aspect a specific tool was created for such a matter called RoboGraph [26], [27], where MIPNs are also used as the code for complex tasks to be carried out by robots or other agents within the system. With this approach, instead of generating the code, RoboGraph is used to execute the control sequences modeled on the nets of the MIPN model. In a similar way a virtual machine runs the same code on different machines, RoboGraph (coded in java) runs the nets of the MIPN model.

Most applications based on middleware systems like the ones we used in this research implement a finite-state machine (FSM). For example, in the case of the autonomous vehicles, the Tartan team that won the DARPA Urban Challenge [28] used a hierarchical finite-state machine to design the behavior generation component [29]. A similar approach was used by the Standford Uiversity team [30] that finished in second place and by the Ohio State University [31]. Since FSMs need to define explicitly all states, the number of nodes tends to grow much faster than the complexity of the reactive system it describes. This can happen in some situations where it is necessary to decompose an action into smaller actions that can run in parallel or when a system can be divided in subsystems. Behaviors Trees (BT) [32] provide a lot of flexibility and can be used to implement the executive layer architecture [33]. BTs are also easier to maintain and to expand.

B. CONTRIBUTION

The main contribution of this paper is the definition of a new model specially designed for distributed control systems that use a message-oriented middleware. This new model named **Message Interpreted Petri Net (MIPN)** is based on the PN model and includes the following features:

- Modeling. Provides a simple hierarchical way to model the process. The definition of the model is presented in Section II.
- Analysis. Provides a method to verify the system through the properties of the model. The process to analysis and verify the model is presented in Section III.
- Control program generation. Unlike other approaches for automatic generation of control code after a formal model [34], the model itself is the code that is executed in our case for a virtual machine named *RoboGraph dispatch* (Section IV-B).
- Control program debugging. As a result of the last feature, the model serves to debug the process presenting the evolution of the system graphically through the sequence of tokens on the net using *RoboGraph GUI* (Section IV-B).

Another advantage of the MIPN derives from its task-oriented design and its aim for applicability. Some applications in Flexible Manufacturing Systems (FMSs) or mobile robot control [35] demand more than cyclic tasks, which are usual in any PN-based model and in PLC programming. They require tasks that start, perform, and finish, communicating their termination to the system. A contribution of the MIPN model is the introduction of a new controlled start-termination capability. This new capability of MIPNs enhances the controllability and analyzability of the system.

The MIPN model is hierarchic, allowing to nest different sub-nets within higher-order ones. Another contribution of this paper is the development of a set of new properties used for the hierarchic-wide analysis of MIPN systems, allowing atomic analysis, i.e., global properties of a system can be obtained from the individual properties of each of its nets. These results are very helpful in designing, validating and verifying all the modeled tasks, allowing both top-down and bottom-up developments. This leads to a faster design and reconfiguration cycle of Petri net models [36]. It allows to model a structure of complex, interrelated, and heterogeneous systems, that can be considered as a system-of-systems (SoS) [37].

This paper is structured in four parts: in the first one (Section II) the MIPN is defined, its benefits are presented, and its similarities and differences with previous models are explained. A second part (Section III) shows an analysis of MIPN model properties, requirements and restrictions for validation and verification of the modeled tasks. A third part (Section IV) shows the results that include our implementation of the model and its applications. Finally, a fourth part (section V) includes the final conclusions about this research.

II. MESSAGE INTERPRETED PETRI NETS

This section provides a progressive insight into the model, from the basic definitions to a more detailed description. The notation used here follows [17].

A. FORMAL DEFINITION

A Message Interpreted Petri Net (MIPN) is described by an n-tuple $MIPN=(\vec{P}, \vec{T}, \vec{F}, \vec{m}_0, \vec{m}_f, \vec{I}, \vec{O}, \vec{W}, \vec{S}, \vec{\varphi}, \vec{\omega})$ with:

- $(\vec{P}, \vec{T}, \vec{F}, \vec{m}_0)$: an ordinary pure binary Petri net with places $\vec{P}$, transitions $\vec{T}$, arcs $\vec{F}$ and initial binary marking $\vec{m}_0$, with $|\vec{P}|, |\vec{T}|, |\vec{F}|, |\vec{m}_0| > 0$, as also defined in [17].
- $\vec{m}_f$: final binary marking, with $|\vec{m}_f| \geq 0$ (i.e. $\{\varnothing \subseteq \vec{m}_f\}$). Whenever all tokens of current marking $\vec{m}_i$ are within the final marking $\vec{m}_f$ or there are no tokens left at all ($|\vec{m}_i \cap \vec{m}_f| = 0$) the execution of the MIPN terminates. Places that include a token in the final marking are named terminal places. This final marking and termination rule are new features introduced in the model that allows the system to control the actual termination of any task modeled by an MIPN.
- $\vec{I}$: a set of incoming message types that the net can receive through subscription, where $|\vec{I}| > 0$. Each message type may have a unique information structure.
- $\vec{O}$: a set of outgoing message types the net can publish, where $|\vec{O}| > 0$. Note that $\vec{O}$ should be disjoint with $\vec{I}$ ($\vec{I} \cap \vec{O} = \varnothing$) to avoid formally incorrect synchronization within the same net, as it will be explained in Section II-D.
- $\vec{W}$: a set of timers, that can be defined to provide time-related control within the MIPN, such as timeouts or delays, and may also act as firing conditions to transitions.
- $\vec{S}$: a set of schedulers, that can be defined to handle repetitive time-related matters within the MIPN, such as hourly, daily, or weekly events, etc. and may also act as firing conditions to transitions.
- $\vec{\varphi}$: a set of firing conditions functions, associating each transition $t_i \in \vec{T}$ with a function expressing its firing condition state $\vec{\varphi}(t_i)$.
- $\vec{\omega}$: a set of output functions, associating each place $p_i \in \vec{P}$ with an output function. Each of these output functions may involve actions related to messages (publish), timers (start), and schedulers (start).

This definition aims to avoid *hidden assumptions* built-in the MIPN definition. Any *hidden assumption* may lead to design decisions that are difficult to correct in a later stage [38].

Regarding the graphic aspects, MIPN uses the traditional representation of ordinary PNs: places are depicted by circles, transitions by bars, arcs by arrows joining places and transitions, and tokens by dots inside the circles of the corresponding places. Terminal places (related to $\vec{m}_f$) are distinguished by a thicker border circle. Also, as MIPN is a binary PN extension, no more than one token is allowed within a place. An example of its graphical representation is shown in FIGURE 1.

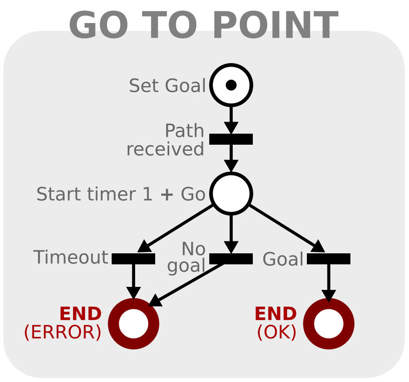


FIGURE 1. Graphical representation of an MIPN. This one contains two terminal places, i.e. two places belonging to the final marking $\vec{m}_f$.

B. PHYSICAL INTERPRETATION

The main purpose of the MIPN is to model the desired dynamic behavior of a discrete event control system while executing a task. So an MIPN describing a task can be seen as a "control system algorithm" and its environment as the "action and event modules" or processes to be controlled (or anything outside the MIPN itself). A graphical representation of those relations between the different elements is shown in FIGURE 2.

In MIPNs, the interaction between the control system and the environment is managed by information exchanged through messages. From the MIPN point of view, the output messages (sent to the environment modules) correspond to the commands to be executed by the distributed modules and the input messages (received from the environment modules) correspond to the events produced by these distributed modules. The input messages and their information structure,

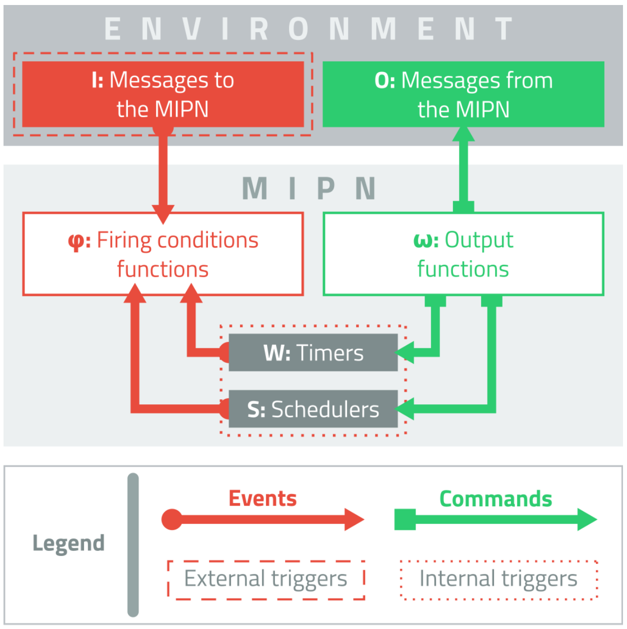


FIGURE 2. Events and commands from the point of view of the net, as well as the interaction between an MIPN and its environment (or process to control).

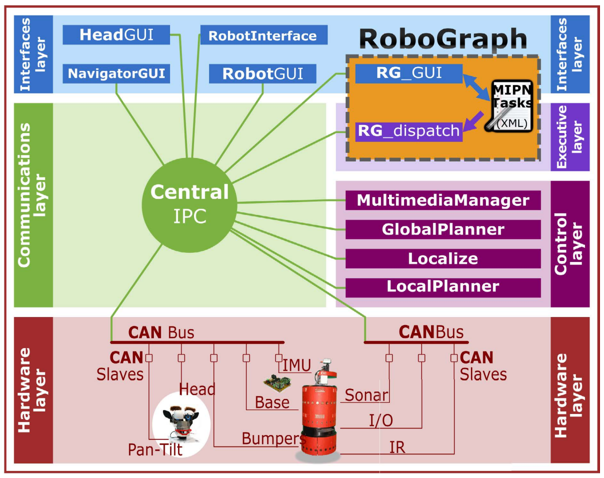


FIGURE 3. Nodes included in a tour-guide mobile robot. Each rectangle in the control, executive and interface layer are independent programs.

together with the internal events produced by timers and schedulers, are the basic components of the firing conditions associated with transitions.

For example, in a tour-guide mobile robot application (FIGURE 3), the main control of the system is carried out by the MIPN node (RG_dispatch in our case) implementing the executive functions that coordinates the other modules. This node is a program that loads and executes the MIPNs. The environment includes the rest of the nodes (programs) that implement the different navigation and human-interaction functions on the robots such as "LocalPlanner", "Localize", "GlobalPlanner", "MultimediaManater", "HeadGui", "RobotInterface", "NavigatorGUI" or "RobotGUI"'. Interaction with the environment are messages published to those nodes (commands) or received from them (events). For example, some commands could be "plan_path", "follow_path", "change_map", "say_text", and "show_video" while possible events could include "path_planned", "goal_reached", "goal_unreachable", and "video_finished".

Whenever an event occurs, the evaluation of enabled transitions is carried out according to the firing conditions functions ($\vec{\varphi}$). If conditions are fulfilled, the corresponding transition will be fired and the marking will evolve.

Places may have associated commands according to the output functions ($\vec{\omega}$). Whenever a token enters a place p_i, its output function $\vec{\omega}(p_i)$ is executed, which leads to the execution of the associated commands. There are three types of commands: output messages to other environment modules (i.e. belonging to $\vec{O}$), timers, and schedulers. Commands to timers and schedulers include start, restart, or modify their parameters.

C. COMMUNICATIONS IN MIPNs

The *interpreted* nature of the MIPN model demands a way to connect the MIPN control algorithm with its dynamic environment. That connection is managed distinctively to other IPN models, both in concept and in the range of target applications. That is the case of systems using message-oriented middleware communications, where the environment can be a heterogeneous combination of entities with very different characteristics, such as different physical locations, software, protocols, or abstraction levels. It is also usual in this case that data is produced and consumed by many different software modules across the system. Such a complex scenario can be found, for example, on distributed and asynchronous systems like multi-robot applications. The coupling provided by messages, usually with the publish/subscribe paradigm, has proven to be a good solution [27].

Middleware can be described as software that connects several components and their applications. It allows multiple different processes to interact, regardless of their particular physical location (one or more computers over one or more networks), platform (usual support for Linux, Windows, macOS, and others), or programming language (C, C++, Java, the most usual) of each one. In these distributed systems the information is transmitted through messages, where data producers are the senders and data consumers are the receivers, taking into account that a process can be sender and receiver at the same time. There are several advantages of the decoupling between publishers and subscribers such as more flexible and dynamic network topology and greater scalability.

Apart from the benefits of transparent connection among very diverse systems, there is also a major advantage of this kind of communication inherited by MIPNs: the quality of the information supported by messages. Traditional data types (booleans, strings, integers, floats, and so on) are supported. But also complex types, such as arrays and objects are permitted. Thus the MIPN model can exchange much richer information with the rest of the system, compared to other previous PN-based models, which are usually constrained to boolean data. As a result of this type of coupling with the environment using message-based communications (see FIGURE 2), a wider range of systems and applications are possible, compared with previous PN models, thus evolving from the usual limited PLC applications to a new vast array of possibilities, especially in distributed applications like process control, robot applications or production plant control among others. This communication paradigm includes the possibility of sending control messages over any network, including the internet. So far, IPC (Inter Process Communication) [39], JIPC [40], and the ROS [41] message-related communication mechanism have been considered.

D. HIERARCHICAL CHARACTERISTICS

PN models for real-world applications end up being huge, complex, and visually challenging net designs. As with most other visual modeling systems, when a model size and complexity increase, it requires a bigger effort to handle it. In that situation, the concept of hierarchy allows to group parts of the net in smaller subnets with specific behaviors [42]. The hierarchic composition features three main assets. Firstly, it increases the readability of the net, making it easier to understand, analyze, and develop. Secondly, it allows layered top-down design, letting the programmer concentrate on global ideas at the beginning and then go deeper into the implementation details, refining each part of the system in separate, nested subnets. From a practical point of view it allows to design small nets for common tasks which are used as building blocks in bigger tasks (e.g., go to a point, text-to-speech, etc.). Thirdly, it allows to analyze and verify the whole system from the analysis of each net (Section III). This, along with the communication capabilities described in II-C, can be effectively considered modeling a system-of-systems [37].

Intuitively each net can be associated with a task. They may involve subtasks, therefore containing subnets representing each of them. Recursively, subtask nesting could go as deep as needed. Also, any termination of a net is reported to its parent net, so it can act subsequently.

The MIPN hierarchical design approach aims for further functionality through modularity but also for transparency [43], seeking that nets can be reused as subnets in other nets (an example can be seen in FIGURE 4), thus gaining all the aforementioned advantages. But subnet nesting has implications that affect the structure and analysis of the resulting composite net (further explained in Section III).

A message is used to start the execution of a subnet in MIPNs as well as to report the end of the subnet. This behavior differs significantly from other interpreted extension models such as SIPN, where subnets have exactly one input and one output place [44]. MIPNs and subnets do not have this restriction and may have more than one initially marked place and more than one eligible termination place, defined by initial marking $\vec{m}_0$ and final marking $\vec{m}_f$, respectively. This allows for more flexible and modular programming.

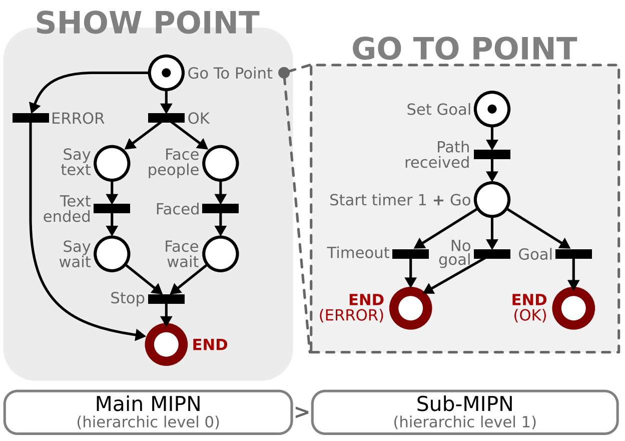


FIGURE 4. Graphical representation of an MIPN (left) that hierarchically contains a subnet (right). In the task "SHOW POINT", the initial place includes the execution of the sub-task "GO TO POINT" defined in the right subnet. When this subnet finishes, a termination report message is issued and a string return value is used to know the outcome of the execution (in this case "OK" or "ERROR").

E. TIMING CAPABILITIES

To describe time-dependent dynamic systems, the time notion has been incorporated into PNs in different ways [45], normally related to timed places, arcs, or transitions.

MIPNs implement the timing capabilities with two specific timing tools and with time-related information associated with messages.

The two timing inner tools are timers and schedulers. Both are capable of triggering the evaluation of firing conditions of those enabled transitions. A set of timers $\vec{W}$ allows countdowns or time counts for custom periods of time. This alone provides similar functionality to timed transitions. Beyond that, a set of schedulers $\vec{S}$ allows planned evaluation of transitions, i.e. hourly, daily or weekly events. Schedulers can generate "precise time point" triggers or "time span" triggers. Timers and schedulers are not associated directly with places, arcs, or transitions. Instead, they are considered net-wide, but can be modified by places (start, re-start, stop) and can affect several transitions at once (by triggering the evaluation of firing conditions). Also, it is possible to define and use simultaneously as many timers and schedulers as needed within an MIPN.

Time-related information associated with messages has an important role within MIPN model. It affects the evaluation of firing conditions depending on the moment a message is received and its type. The message type is relevant, as the sole reception of a message of a certain type at any time can be meaningful within a task. Transitions in MIPNs feature two different evaluation modes for message reception:

1) "Recently" mode: Evaluates if a message of a given type arrived after the transition became enabled. This is the usual behavior in event-based Petri nets, where

transitions are only aware of events that happened after they have been enabled.

2) "Anytime" mode: Evaluates if at least one message of a given type arrived since the start of the net, regardless of whether the transition was enabled or not at the moment of arrival. This unique feature allows using of valuable information generated at any moment in the past, even before a transition became enabled. This allows being aware of things that are no longer active and do not produce more events. This is the case of the completion of other tasks, i.e., the reception of termination messages. This may also be useful to be aware of alarms, initializations, etc.

The designer may choose which mode to use in each case. These evaluation modes are used within the firing conditions functions ($\vec{\varphi}$) as part of the conditions related to message reception. Other conditions may involve the message type and its specific data. If new messages arrive after the first, only the parameters of the last one will be evaluated if a parameters check is required.

All these capabilities give the MIPN model the power and flexibility to handle any kind of time-related tasks needed in the applications they were implemented so far (Section IV-C).

F. DYNAMIC BEHAVIOR

MIPN marking evolution follows the same general rules applied to ordinary PNs. More specifically, conditions are evaluated only when one or more of the three types of events occur —message reception, timer event, and scheduler event—. These events cause the evaluation of transitions according to their associated $\vec{\varphi}$ functions and may cause their firing, which consequently causes the "flow" of tokens and the execution of $\vec{\omega}$ functions defined in those new occupied places.

MIPN dynamic behavior is characterized by the transition firing rules and the evaluation/evolution algorithm.

The transition firing rules are:

1) A transition is enabled if all its input places are marked. To avoid possible conflicts, the output transitions of a place cannot fulfill the firing conditions simultaneously.
2) A transition is fired if it is enabled and fulfills all its firing conditions.
3) The firing of a transition is considered instantaneous, i.e. consumes no time.
4) Firing is a step-by-step process, meaning that if a firing of a transition t_i grants new conditions for the firing of another transition t_j, it will not occur until the next evaluation of transitions. This prevents iterated firing within the same validation cycle and the "jump" of tokens beyond the output places, as seen in SIPN.
5) Events cannot happen simultaneously. Instead, they are treated only one at a time. Even though messages can reach their destination simultaneously, they are queued and handled sequentially. Therefore not more than one new message event takes effect in transition firings within the same validation cycle.

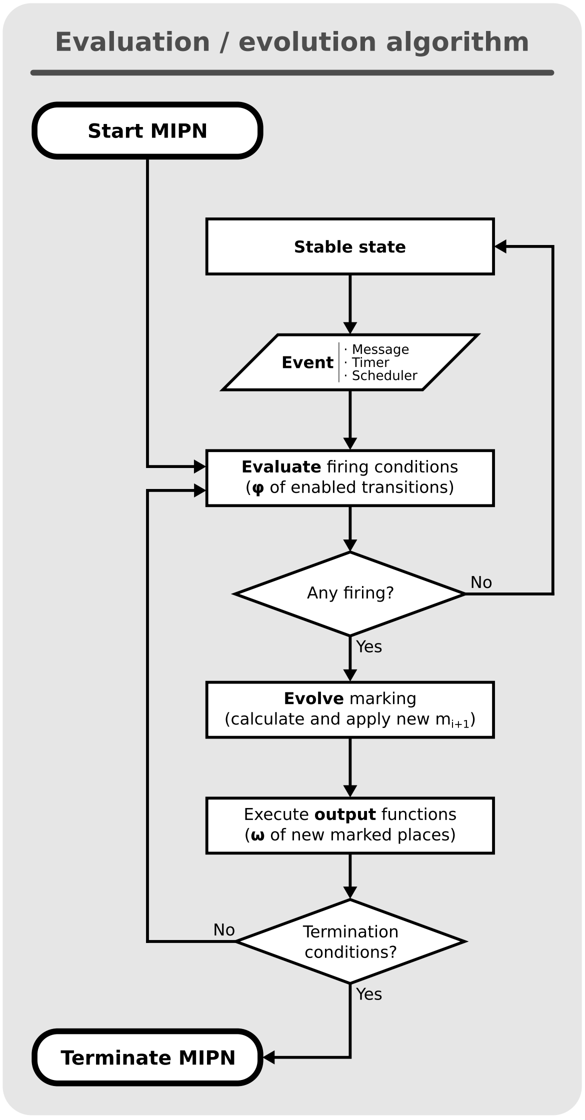


FIGURE 5. Flowchart of the evaluation/evolution algorithm.

SIPN has a particular characteristic, dynamic synchronization (DS). It may lead a token to "jump" beyond a transition t_i output places if the firing of t_i grants the conditions to fire another one. IPN does not even consider this possibility and MIPN explicitly prevents the possibility of DS. It establishes its dynamic behavior as a step-by-step process. The evaluation/evolution algorithm evaluates the necessary conditions for the evolution of the marking. Furthermore, it carries out all needed actions to effectively make the net evolve. The flowchart of the evaluation/evolution algorithm is depicted in FIGURE 5, and its steps are the following:

1) One or more trigger events occur.
2) All enabled transitions are evaluated according to their related firing conditions functions ($\vec{\varphi}$): those fulfilling all their conditions, if any, are fired.

3) If any firing occurred:
    a) The new marking $\vec{m}_{i+1}$ is calculated and applied, and thus the net evolves, i.e. the tokens "flow".
    b) All output functions ($\vec{\omega}$) that might have been defined in the new marked places are executed.
    c) All enabled transitions are evaluated again, starting a new evaluation/evolution cycle.
4) If no firing occurred: the system stays stable, waiting for the next message or time event to trigger a new evaluation/evolution cycle.

As part of the firing conditions evaluation, the message arrival deserves a special study. Message-based communications with the environment is one of the main assets of this model and has several possibilities. Message conditions can be classified into:

1) "Just arrival" mode: only the message type is evaluated to see if it matches the given type of condition, no other message parameters are evaluated. This is useful when the mere reception of a message has an implicit meaning, e.g., subnet termination messages.
2) "Field conditions" mode: the type of arrival message is evaluated, as well as some of its parameters. The condition, in this case, can be any expression on the message parameters which results in a boolean value.

III. MIPN ANALYSIS

Petri nets are well known for their strength in modeling discrete event dynamic systems, but they have another key advantage: They can also be used for model-checking, validation, and verification. Therefore, PN analysis allows early detection and immediate correction of errors in algorithm design. One example is the deadlock (or the livelock), where the algorithm gets stuck with a permanent marking or in a local loop during the execution. The next section reviews the basic MIPNs properties and the following sections are focused on the verification and analysis of the specific systems modeled by MIPNs. We will focus mainly on reachability, liveness and terminability because they are the more relevant properties for the kind of applications that use these middleware control systems. Reachability testing is one of the most popular and basic means for checking the accuracy of the model compared to its expected execution. For example, reachability testing of the Petri net modeling through the use of Gröbner bases was performed in [46] for the control of a mobile robot. On the other side, a bounded and live net means that the modeled system cannot have capacity overflows and deadlocks, two types of unwanted behaviors in applications that include manufacturing [47]. Finally, terminability is added to make sure that some tasks come to an end for the analysis of hierarchical and nested networks.

A. BASIC PROPERTIES

The analysis process allows the extraction of characteristics of a given PN-based model. It allows verification and avoids design errors or unwanted behaviors. Some basic properties define the main characteristics of a PN-based model. They provide information about performance, restrictions, conflicts, design errors, etc. These basic properties can be either dependent on an initial marking $\vec{m}_0$, called behavioral properties or independent of $\vec{m}_0$, called structural properties. A summary of the properties used in this research follows; further explanation can be found on [48]:

- Reachability: Indicates whether the "flow" of tokens can reach all, some, or none of all possible markings of the net. A marking $\vec{m}_i$ is reachable if there exists a sequence of transition firings that leads to that particular marking $\vec{m}_i$ starting from the initial marking $\vec{m}_0$. The set of all reachable markings from $\vec{m}_0$ is denoted by $R(\vec{m}_0)$.
- Coverability: Shows whether a marking is contained by another reachable one or not. A marking $\vec{m}_i$ is coverable if: $\exists \vec{m}_j \in R(\vec{m}_0) \;:\; \vec{m}_j(p_n) \geq \vec{m}_i(p_n),\;\; \forall p_n \in \vec{P}$
- Boundedness: Sets a limit on the number of tokens in a place. A place is k-bounded if it does not contain more than k tokens for all markings. The MIPN is k-bounded if all the places are k-bounded: $\vec{m}_j(p_i) \leq k;\;\; \forall p_i \in \vec{P}, \forall \vec{m}_j$
- Safeness: It is a specific case of boundedness, with $k = 1$ (1-bounded), meaning that the number of tokens contained in any place will never exceed 1. A place is safe when it is 1-bounded. Therefore, a PN is safe when each one of its places is safe. An MIPN is, by definition, safe.
- Reversibility: This means that an initial marking $\vec{m}_0$ can be reached again. A PN is reversible if for each marking $\vec{m}_i \in R(\vec{m}_0)$, initial marking $\vec{m}_0$ is reachable from $\vec{m}_i$, which means the PN can always revert to its initial state.
- Liveness: Indicates whether transitions can be fired again. A transition t_i is said to be live for an initial marking $\vec{m}_0$ if, from any reachable marking $\vec{m}_j$, there exists a sequence of transition firings that grants t_i can be enabled and fired again. Otherwise, it is said to be dead. A PN (B) is said to be live ($L[B]$) if and only if all its transitions are live.
- Quasi-liveness: Indicates the capability of transitions to be fired at least once. A transition t_i is said to be quasi-live for an initial marking $\vec{m}_0$ if, there exists a sequence of transition firings that grants t_i can be enabled and fired. This is, ($\exists \vec{m}_j \in R(\vec{m}_0)$) such that t_i is fireable from $\vec{m}_j$. A PN (B) is said to be quasi-live ($QL[B]$) if and only if all its transitions are quasi-live.

Besides the properties inherited from PNs, the MIPN model comprises specific properties that can simplify and extend the analysis, especially when dealing with nested, multilevel hierarchical MIPNs.

Definition 1 (Terminability): An MIPN (B) is said to be terminable ($\tau[B]$) for an initial marking $\vec{m}_0$ if, from any reachable marking $\vec{m}_j$, there exists a sequence of transition firings that ends in a terminal marking $\vec{m}_t$ and therefore its execution terminates. $\vec{m}_t$ is a terminal marking if it is covered by the final marking $\vec{m}_f$ (i.e. all tokens of current marking $\vec{m}_i$ are within the final marking $\vec{m}_f$ or there are no tokens left at all, given that $\{\varnothing \subseteq \vec{m}_f\}$). When reaching $\vec{m}_t$, the execution of the MIPN terminates, noticing it to the system, through a termination message.

$$\tau[B] \iff \forall \vec{m}_i \in R(\vec{m}_0),\ \exists\, \vec{m}_t \in R(\vec{m}_i) \Big/ \vec{m}_t \subseteq \vec{m}_f \tag{1}$$

where:

$$\vec{m}_t \subseteq \vec{m}_f \iff \vec{m}_t \cap \overline{\vec{m}_f} = \varnothing \tag{2}$$

Due to their definitions, liveness and terminability are mutually exclusive within the same MIPN.

B. BEHAVIORAL ANALYSIS

Model checking is one of the formal verification methods most used in software and hardware production [12]. As in model checking, our interest is to provide a method for checking whether the MIPN model of a system meets a given specification and identify the possible problems. However, since we work with interpretable PNs, it is necessary to narrow down the nature of problems that these methods are capable of detecting. A possible division of these issues is the following:

- Structural issues: These are problems associated with the structure of the net and they are independent of the conditions associated with transitions. For example, there is no reachable terminal marking $\vec{m}_t$.
- Functional issues: These are problems related to the conditions associated with transitions. For example, the condition would never hold.

Only the structural issues are considered in this study. This is, all transition conditions are considered well designed and therefore, all transitions can eventually be fired.

There are mainly two approaches to analyze PNs: algebraic analysis and reachability graph-based analysis [49]. A reachability graph (RG) is a graph representation of all possible firing sequences of a net, and it is commonly used for the analysis of net evolution and properties depending on initial marking $\vec{m}_0$. With the RG it is possible to establish if a specific marking is reachable. It can also identify some behaviors such as loops, deadlocks, dead-ends, etc. The analysis presented here is based on the RG analysis. The generation of the RG is conceptually simple and is integrated into most PN software analysis tools. One way to obtain the RG is by using the Karp and Miller algorithm [50] to obtain first the Reachability tree. Since MIPNs are 1-bounded, the reachability graph is finite and it can be obtained iteratively, starting with the initial marking and then considering all reachable markings. A simple example is shown in FIGURE 6 for the ''Show point'' task of a tour guide robot application [27]. The vertices of the graph correspond to possible markings. Every vertex is labeled as $M_{i.j...}$ where the indexes $(i, j, \ldots)$ correspond to the places with a mark. For example, $M_{2.4}$ means that place $p2$ and $p4$ are marked.

In the case of bounded nets, the Reachability Tree (RT) [48] is a tree representation of its possible firing sequences. The Reachability Graph (RG) is obtained from the RT by merging the nodes with the same marking. The RT can be obtained by the Karp and Miller algorithm [50] that for the case of binary Petri nets is reduced to algorithm 1.

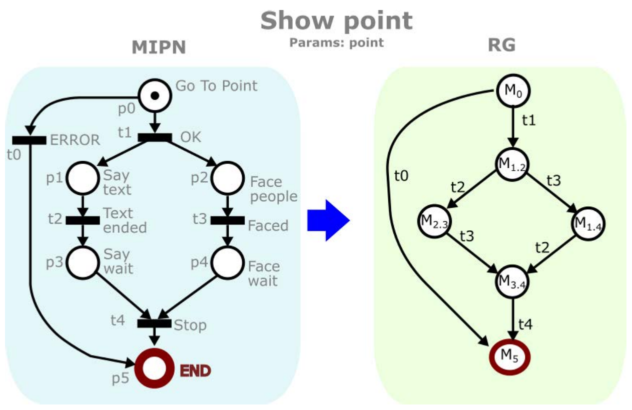


FIGURE 6. Representation of the MIPN model and Reachability Graph for the "Show point" task.

Algorithm 1 Algorithm to Obtain the Reachability Tree

```
1:  Label initial marking $\vec{m}_0$ as the root of the tree and tag it
    as *new*.
2:  **for** each *new* marking **do**
3:      select a new marking $\vec{m}_j$.
4:      **if** M is identical to a marking on the tree **then**
5:          tag M as *old*
6:          go to another *new* marking
7:      **end if**
8:      **if** no transitions are enabled at $\vec{m}_j$ **then**
9:          tag M as *dead-end*.
10:     **end if**
11:     **for** each enabled transitions at $\vec{m}_j$ **do**
12:         obtain the marking $\vec{m}_i$ that results from firing t at $\vec{m}_j$.
13:         introduce $\vec{m}_i$ as a node, draw an arc with label t from
            $\vec{m}_j$ to $\vec{m}_i$ and tag $\vec{m}_i$ as *new*.
14:     **end for**
15: **end for**
```

In model checking the specifications are usually defined using some temporal logic formula but structural properties can also be checked. The specifications we are interested in include:

- **Reachability**: some unsafe states should never happen.
- **Liveness**: there should be no unreachable actions in the model.
- **Terminability**: deadlocks should be avoided and most tasks should come to an end.

1) REACHABILITY

The state of a system modeled by an MIPN is defined by the current marking $\vec{m}_i$. In theory, if the MIPN has p places, the combination of marks on states is 2^p. However, in practice, only the states represented in the RG can take place. For example, in FIGURE 6 the state corresponding to making $M_{0.1}$ is not reachable because there is no sequence of events that will lead to the situation where places $p0$ and $p1$ have a mark at the same time.

In model checking, the model of the system is checked for some defined properties or specifications. In case some of those properties do not hold, appropriate counterexamples are generated to identify the error source. For most of the message-driven distributed systems modeled with MIPNs an easy way to define unsafe states is to use the markings that correspond to those states ($\vec{m}_u$). In that case, checking if those states are reachable can be done using the RG. If some of those unsafe states exist in the RG, there is a possibility to reach that state. Besides, in this case, the sequence of events (transitions) that lead to that unsafe state is the sequence of events associated with the transitions corresponding to the arcs in the RG between the initial marking $\vec{m}_o$ and the marking corresponding to the unsafe or failure state $\vec{m}_u$.

As a simple example, in the "show point" task of FIGURE 6, before start talking, the robot needs to reach the point. In the MIPN model, this means that places *p0* ("Go To Point") and *p1* ("Say text") can not have a mark at the same time. That corresponds with the RG since there is no vertex labeled $M_{0.1}$.

2) TERMINABILITY

Most of the tasks modeled in this kind of systems should finish in a limited period. Therefore, verifying the absence of deadlocks is another important property to check on the model. This can be tested on the RG using the following theorem:

Theorem 1: A single MIPN is terminable if its RG fulfills the following condition:

- All markings at the end of the RG branches are terminal markings.

Proof: The RG branches represent all possible firing sequences and the terminal nodes represent all the possible markings where it can end or stop. If all the markings associated with those terminal nodes are terminal markings, it fulfills the terminability condition (Definition1). ☐

In case the condition does not hold, the model checker will generate the list of branches that end in a non-terminal marking. For each branch, the list of transitions fired that lead to the deadlock is provided. This list of transitions identifies the sequence of events that end in a deadlock.

The example in FIGURE 6 is a well-modeled task and the MIPN is terminable because the only terminal vertex of the RG (M_5) is a terminal marking. However, wrong modeling of the net like in FIGURE 7 where the two branches start from a place (*or* node) instead of a transition (*and* node) leads to a net that does not hold the terminable condition. This situation can easily be detected in the corresponding RG of FIGURE 7 because two vertexes at the end of the RG branches correspond to non-terminal markings (M_3 and M_4). Only the branch in the middle (M_5) corresponds to a terminal branch.

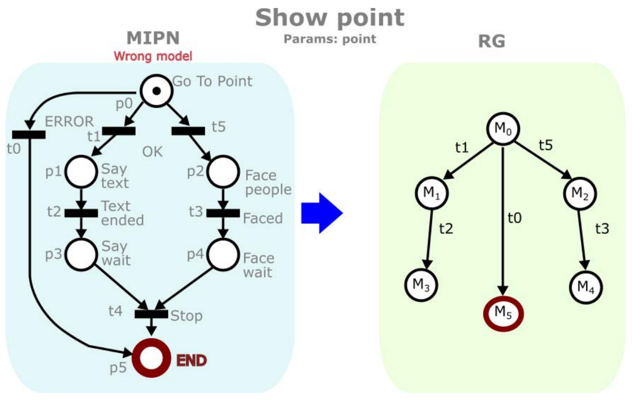


FIGURE 7. MIPN model and Reachability Graph for the case of wrong modeling of the "Show point" task. The correct model is shown in FIGURE 6.

3) LIVENESS

For the kind of applications that MIPNs are designed, liveness is not always a desired requirement because terminability and liveness are mutually exclusive. However, quasi-liveness should be checked to avoid design problems with transitions of a Petri net that are unreachable. Having unreachable transitions is a clue for an error in the design. It would be similar to have a part of a software program that is unreachable and therefore will never be executed. The quasi-liveness property of an MIPN can be checked using the following theorem:

Theorem 2: A single MIPN is quasi-live if its RG fulfills the following condition:

- All transitions of the MIPN are included in some arc of the RG.

Proof: The RG branches represent all possible firing sequences and the arcs represent the transitions fired in the sequence. If all the transitions t_i are included in the RG means that $\exists \vec{m}_j \in R(\vec{m}_0)$ such that t_i is fireable from $\vec{m}_j$ and therefore, fulfills the quasi-liveness condition. ☐

In case the condition does not hold, the model checker will generate the list of transitions not reachable. This is, the transitions that are not included in the RG.

The example in FIGURE 6 is a quasi-live MIPN because all transitions are included in some arc of the RG (M_5). However, the case shown in FIGURE 7 does not hold the quasi-live condition. This situation can easily be detected in the corresponding RG of FIGURE 7 because there is no label t_4 in any of the arcs. That means that transition t_4 is not quasi-live and consequently the MIPN is not quasi-live.

C. ATOMIZATION OF ANALYSIS

The RG can be obtained simply by computing all successor markings starting with the initial marking $\vec{m}_0$. However, in general, due to the state-space explosion, the generation of the reachability graph is inefficient even for bounded PNs. This analysis becomes more inefficient or intractable for nets with a lot of places and transitions like the ones needed to model complex tasks in these message-oriented middleware distributed systems. To deal with this problem, several

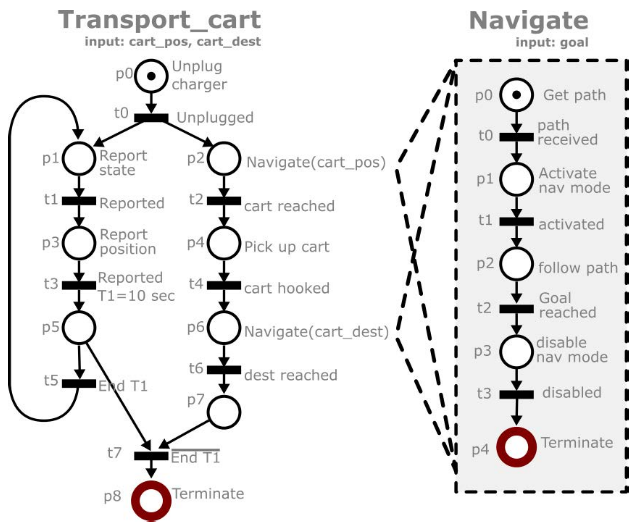


FIGURE 8. **MIPN model for the *transport_cart* task. This task includes the *navigate* subtask.**

researchers proposed different hierarchical solutions [42], [49] as a divide and conquer approach. Since the MIPNs are by definition hierarchical, a similar approach can be used to reduce the dimensionality of the problem. One example of the application of hierarchy is shown in FIGURE 8.

It is important to keep in mind that the execution of subtasks in MIPNs is carried out through messages as any other commands or actions in the system. This means that when one of the places p_2 or p_6 gets a token, a message to run the navigate subtask is issued. When the goal is reached and therefore the execution of the navigate subtask finishes, a special message regarding the end of the subtask is issued. That message can include data regarding the outcome of the execution. This outcome message is associated with the output transitions t_2 and t_6.

In the case of hierarchic nets (i.e. a net A with one or more nested subnets spanning one or more hierarchical levels), it is mandatory to evaluate the whole multi-level system to extract its global properties and behavior. The termination capabilities of the MIPN play a very important role in the hierarchical properties of an MIPN and its derived global behavior and characteristics (such as safeness and liveness), as will be explained later. The possibility of hierarchic nesting implies some challenges for the analysis and some design restrictions that should be followed.

Before dealing with further analysis some core definitions must be established.[2]

- Let A be an individual single MIPN, which may have subnets. H_A is the hierarchic system of MIPNs where A is the top-level (or parent) net.
- Hierarchic level $l \in [0, h]/l \in \mathbb{N}$ is the level of nesting of a net within the whole hierarchic system of nets. Thus, the 0-level is the top level of the hierarchy (i.e. a net not contained by any other) and the h-level is the bottom of the hierarchy (i.e., the most nested level, not containing any further subnets). Also, $level_A(B) = k$ is the hierarchical level of B within that net system H_A.

$$0 \leq level_A(B) = k \leq h_A = maxlevel\{H_A\} \tag{3}$$

- N_A is the set of all n nets ($| N_A | = n$) within the net system H_A.
- $N_A(l)$ is the set of all nets of hierarchic l-level out of N_A, and $| N_A(l) | = n_l$ is the number of nets within l-level.
- SN_A is the set of all subnets of A, if any, i.e. all nets within the system H_A except A itself.

$$N_A = A \cup SN_A \tag{4}$$

- $N_{A,m}$ is the m^{th} net out of the set N_A, where $0 \leq m \leq n$

1) REACHABILITY

In the case of a hierarchical MIPN, the state of the system is defined by the current marking $\vec{m}_i$ of all the running nets. If the state to analyze include some subnets, the two next conditions should be checked to see if the marking matches the state:

- There is at least a vertex with a matching marking for each RG involved in that state.
- All the subtasks included in that state should be running. Therefore, for each subnet involved in that situation, there must be, in another net, some marked place running the subnet.

In the task represented in FIGURE 8 a simple example is to verify that the robot is never unplugging from the charger (place p_0 in MIPN *Transport_cart*) and following a path (place p_2 in MIPN *Navigate*) at the same time. In this case, it is necessary to make sure that both places are not marked at the same time. To reach that situation the next conditions should hold:

- There should be a marking in the *Transport_cart* RG that includes the place p_0 and one of the places p_2 and/or p_6 that run the *Navigation* subtask. Therefore, the possible markings are $M_{0.2}$, $M_{0.6}$ and $M_{0.2.6}$.
- There should be a marking in the *Navigate* RG that includes the place p_2.

The RG for FIGURE 8 is represented in FIGURE 9 and it is easy to see that it does not meet the first condition because the RG for the main net does not include any of the three markings ($M_{0.2}$, $M_{0.6}$ nor $M_{0.2.6}$) on a vertex. Therefore, according to the MIPN model that fault situation should never take place.

2) TERMINABILITY

Let's first extend the definition of terminability to the case of a hierarchical system.

[2]For simplicity, direct or obvious demonstrations for definitions and corollaries are omitted.

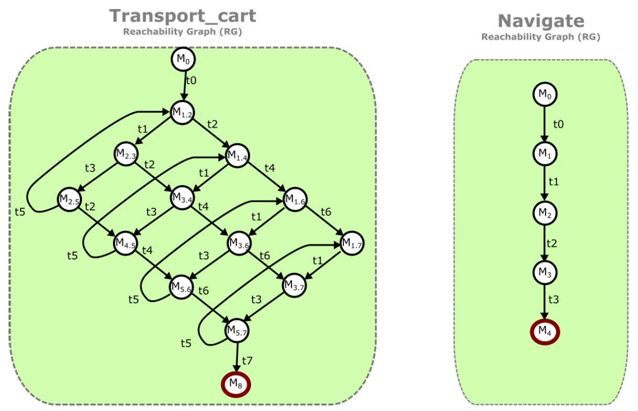


FIGURE 9. **Reachability Graph for the *transport_cart* and *navigate* MIPNs shown in FIGURE 8.**

Definition 2 (Terminability): Terminability is expressed by a boolean function represented by τ:

$$\tau_d[A] = \begin{cases} 0 & \text{, if A is not terminable} \\ 1 & \text{, if A is terminable} \end{cases} \tag{5}$$

The Subindex $_d$ indicates the "depth" of the analysis. Depths may be "individual[3]" (τ_i), "subhierarchy" (τ_{sh}) or "global" (τ_g). The default depth is "individual", if no subindex is specified.

- **Sub-hierarchy terminability (τ_{sh}):** The sub-hierarchy terminability of a net $A \in H_A$ refers to the combined terminability of the subnets of A, but not considering A itself. Considering that $N_A(1)$ are the subnets of A, this function is defined depending on $|N_A(1)| = n_1$:
  1) the combined terminability (productory) of individual and subhierarchy terminabilities of each subnet of A ($N_A(1)$), if A has subnets (i.e. $n_1 \neq 0$).
  2) 1, if A has no subnets (i.e. $n_1 = 0$)

$$\tau_{sh}[A] = \begin{cases} \prod_{m=1}^{n_1} \left(\tau_i[N_A(1)_m] \cdot \tau_{sh}[N_A(1)_m]\right), & \\ \qquad \text{if } SN_A \neq \varnothing \\ 1, \quad \text{if } SN_A = \varnothing \end{cases} \tag{6}$$

- **Global terminability (τ_g):** The global terminability of a net A is the terminability of the whole system H_A, considering the influence of A and all its subnets, which means that each one of the subnets of A, if any, affects the global terminability of A.

$$\tau_g[A] = \begin{cases} \tau_i[A] \cdot \prod_{m=1}^{n_1} \left(\tau_g[N_A(1)_m]\right) & \text{, if } SN_A \neq \varnothing \\ \tau_i[A] & \text{, if } SN_A = \varnothing \end{cases} \tag{7}$$

Sub-hierarchy terminability can be obtained recursively from Equation 6, computing the productory of the individual terminability of each net in SN_A set, if any, or 1 if A has no subnets.

$$\begin{aligned} \tau_{sh}[A] &= \prod_{m=1}^{n_1} \left(\tau_i[N_A(1)_m] \cdot \tau_{sh}[N_A(1)_m]\right) \\ &= \prod_{m=1}^{n_1} \Big(\tau_i[N_A(1)_m] \cdot \prod_{p=1}^{n_2} \left(\tau_i[N_A(2)_p] \cdot \tau_{sh}[N_A(2)_p]\right)\Big) \\ &= = \cdots = \prod_{m=1}^{n-1} \left(\tau_i[SN_{A,m}]\right) \quad \text{, } \mathit{if} \quad SN_A \neq \varnothing. \end{aligned} \tag{8}$$

Corollary 1 (Relation between τ_g and τ_i): Global terminability can also be obtained recursively from Equation 7, computing the productory of the individual terminability of each net in N_A set.

$$\begin{aligned} \tau_g[A] &= \tau_i[A] \cdot \prod_{m=1}^{n_1} \left(\tau_g[N_A(1)_m]\right) \\ &= \tau_i[A] \cdot \prod_{m=1}^{n_1} \Big(\tau_i[N_A(1)_m] \cdot \prod_{p=1}^{n_2} \left(\tau_g[N_A(2)_p]\right)\Big) \\ &= \prod_{m=1}^{n} \left(\tau_i[N_{A,m}]\right) \end{aligned} \tag{9}$$

From a practical point of view, the conclusion is that if an MIPN A is globally terminable ($\tau_g^s[A]$), then the hierarchic system H_A is free of deadlocks and its termination is always possible. Also, the global terminability can be obtained from the individual terminability of all the nets in the hierarchy using Equation 9.

As a consequence of the previous conclusions, the terminability analysis of an MIPN can be based on the RG of each individual net. If all the individual RG hold the terminability condition (Theorem 1), the global MIPN is terminable.

3) LIVENESS

For the kind of applications that MIPNs are designed, compliance with quasi-liveness avoids design problems related to transitions that are unreachable. The quasi-liveness property of a global MIPN can be assessed from the RGs using the following theorem:

Theorem 3: A hierarchic MIPN is quasi-live if the RGs fulfill the following two conditions:

1) *All the RGs for the nets of the MIPN hold the quasi-live condition.*
2) *Every subnet should be included in at least a place of a higher level net and that place should be included in some marking of the RG corresponding to that net.*

Proof: The quasi-liveness condition for the global MIPN requires that all the transitions t_i in all the nets N_j are quasi-live. Therefore, $\forall\ t_i \in N_i$, there exists a sequence of transition firings in different nets that grants t_i can be enabled and fired. This can be proven using both conditions:

1) The first condition implies that within the net N_i that includes the transition $t_i \in N_i$ there is a sequence

[3]For simplicity, simple notation "terminable" implies "individually terminable" (Definition 1), otherwise should be specified. Thus: $\tau = \tau_i$

of transition firings from the initial marking $\vec{m}_0$ that grants t_i can be enabled and fired.

2) The second condition, implies that there is a net N_j that includes the execution of net N_i and therefore the initial marking $\vec{m}_0$ of N_i can be reached. This condition can be applied recursively until the main net N_0 is reached.

□

In case the condition does not hold, the model checker will generate the list of transitions not reachable (transitions not included in the RG) and nets not executable (nets not included in another net).

IV. RESULTS

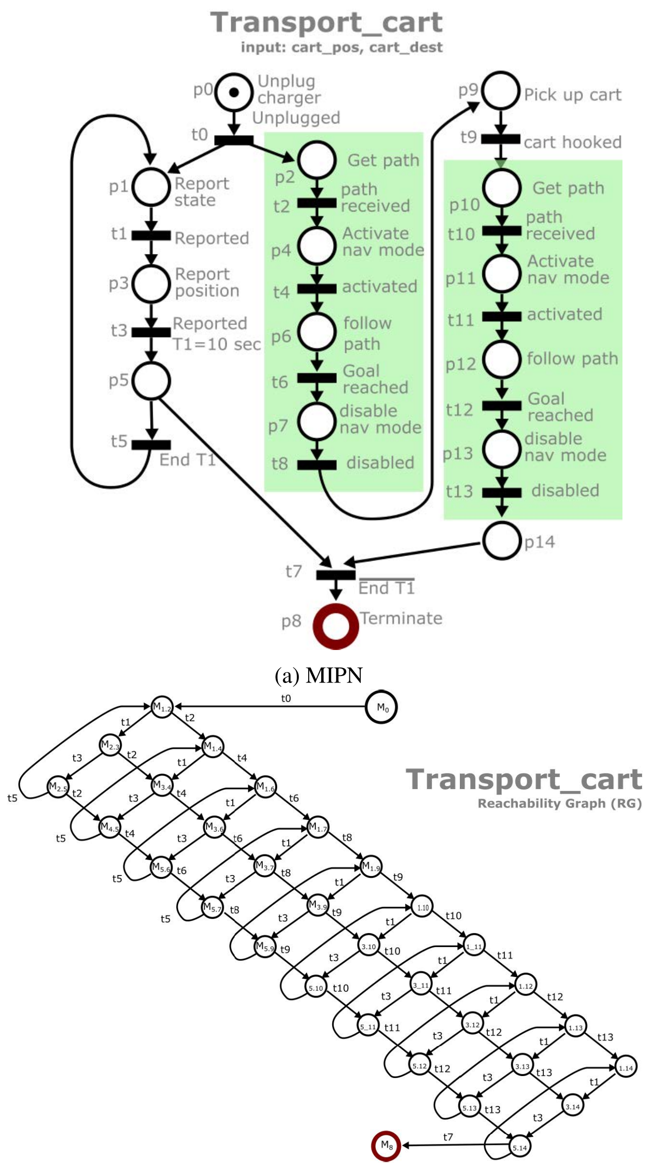
The MIPN theoretical model has been motivated by the requirements of actual message-based communications hierarchic systems that exceeded the capabilities of current PN extensions. MIPNs are modular and hierarchic because it is the most natural way to model the behavior of these complex systems but at the same time, these atomic properties simplify the analysis as we pointed out in Section III-C. This section includes first a simple example that illustrates these advantages. Then we introduce the IDE that we have developed to implement these systems and some of the applications where it has been used.

A. A SIMPLE EXAMPLE

Let's use the simple example of FIGURE 8 to get an idea of the reduction of the RG provided by the hierarchical nature of MIPNs. The equivalent flat net without hierarchy is shown in FIGURE 10a and the corresponding RG is represented in FIGURE 10b. The number of vertices (nodes) in the RG for the hierarchical case (FIGURE 9) is 19 while for the non-hierarchical case (FIGURE 10b) is 32. As we can see, the number of vertices for this simple case with only one subnet is reduced substantially. TABLE 1 shows the reduction of nodes in the RG as a function of the number of places in the subnet. As the complexity of the subnet increases, there is a higher benefit of using hierarchy. That reduction is even more significant as the number of simultaneous sequences increases because there is a higher number of possible combinations of markings and more calls to the subnets. TABLE 2 shows this reduction when the branch of the MIPN that calls the subnet in FIGURE 8 is replicated several times. It can be observed that the reduction in this case as the number of branches in parallel increase is exponential.

B. IMPLEMENTATION

There are many software systems so complex that need to be divided into different modules. A popular solution, especially in distributed systems, is to implement different parts of the system in several, independent processes that use some message exchange mechanism to share information. Most of the autonomous vehicle software architectures are among those systems. The executive module should coordinate the activity of the rest of the modules to carry out some specific task. The process to build this component includes two steps: the



(a) MIPN

(b) RG

FIGURE 10. Non-hierarchical MIPN and RG of the task defined in FIGURE 9.

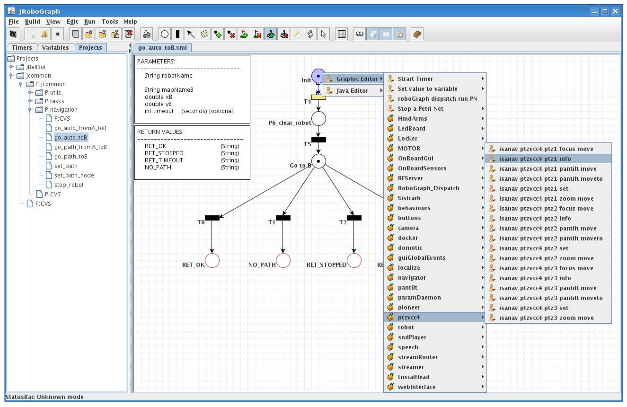
first step consists of building the model as some sequence of actions and events and the second step consists of generating the code that implements the model. Throughout the last twenty years we have been modeling this executive module as a finite state machine first, then as different kinds of PNs and finally using MIPNs. The software generated in the second step includes instructions to send messages to other modules to carry out different actions and to subscribe to messages containing information about the events. To eliminate the second step, we built the tool RoboGraph [27] where we can define the MIPN model that serves at the same time as the program to control and coordinate the other modules. RoboGraph adds two modules to the architecture: RoboGraph GUI (RG_GUI) that allows to edit and debug the MIPNs

TABLE 1. Number of vertex of the reachability graph versus the number of places in the subnet when using a model with (MIPN) and without (IPN) hierarchy.

Type of model	Number of places in the subnet						
	3	5	7	9	11	13	15
IPNs	20	32	44	56	68	80	92
MIPNs	17	19	21	23	25	27	29

TABLE 2. Number of vertex of the reachability graph versus the number of branches in the main net when using a model with (MIPN) and without (IPN) hierarchy.

Type of model	Number of branches in the main net					
	1	2	3	4	5	6
IPNs	32	302	3002	30002	300002	3000002
MIPNs	19	55	199	775	3079	12295



```
PARAMETERS:
---------------------------------
   String robotName

   String mapNameB
   double xB
   double yB
   int timeout    (seconds) [optional]

RETURN VALUES:
---------------------------------
   RET_OK                    (String)
   RET_STOPPED               (String)
   RET_TIMEOUT               (String)
   NO_PATH                   (String)
```

FIGURE 11. RoboGraph in editor mode displaying the list of modules and messages of a module.

models and RoboGraph dispatch (RG_dispatch) that executes the nets.

RG_dispatch implements the executive layer of the architecture executing the tasks defined as MIPNs. When RG_dispatch executes the task, the messages assigned to places will be published as the net progresses. On the other side, when RG_dispatch loads a new MIPN subscribes to all messages included in conditions associated with transitions.

RG_GUI can work in three different modes: Editor, Monitor, and Play Logger. In editor mode (FIGURE 11) the user can create new tasks using a simple and intuitive MIPN graphical editor. First, the MIPN structure is created by selecting and dragging the different elements: places, transitions, arcs, and marks. Then actions, associated with places (publish messages, start a timer, start other MIPN,...), and conditions associated with transitions (conditions on the arrival of messages, end of a subnet,...) must be defined. All the messages can be selected from a menu list automatically generated by the GUI as shown in FIGURE 11.

Once the tasks are defined, RoboGraph provides several tools to analyze the MIPNs using the methods described in this paper. These tools are integrated into RoboGraph [51] along with the programming tools, so the developer has immediate access to design/debug/analysis tools in the same IDE, thus enhancing the designer's workflow and making easier the design task.

RG_GUI in monitor mode is used in execution time to watch the evolution of the running MIPNs. While executing a net, every time RG_dispatch fires a transition, it publishes a message to report it. RG_GUI in monitor mode is subscribed to this kind of messages to show every running Petri net in a different tab with the current marking. Because of the hierarchical nature of MIPNs, most of the applications have several nets running at the same time and it is quite difficult to monitor the execution in real time. For this purpose, an XML log file with dispatch IPC/JIPC/ROS messages is created in running time. The programmer can then run RG_GUI in play-logger mode, open the log file and play it back. Besides the regular play option, the user can monitor the log file step by step or jump to a specific place in ''execution''.

C. APPLICATIONS

The MIPN model has been used and polished in several robot control applications, providing an easy way to design, verify, and debug complex architectures that include multiple modules. RoboGraph is currently working in three different message-related communication systems: IPC [39], JIPC [24], and the ROS (Robot Operating System) communication system [41].

1) APPLICATIONS THAT USE IPC AND JIPC

Applications that use IPC and JIPC include the following (see also FIGURE 12):

- **WatchBot** [26]. A multi-robot surveillance application that allows for scheduled patrolling, video camera control, robot teleoperation, alarm managing, etc.
- **HospBot** [52]. A robotic automated delivery system for hospital facilities.
- **BellBot** [53]. A hotel assistant system using mobile robots that guide guests through the hotel and deliver small items such as drinks or the newspaper to their rooms.
- **GuideBot** [54]. A tour-guide robot for fairs and museums. The robot participated in several editions of the ''Xuventude Galicia Net'' event.

All these applications have been created using **RIDE** (**R**obotic **I**ntegrated **D**evelopment **E**nvironment) [22], [24]. The control architectures developed using RIDE are based on a middleware layer with several independent modules that implement primitive actions and report events about their state. Project developers use **RoboGraph** [27] to define tasks (RG_GUI) as MIPNs and store them in XML files. The RoboGraph dispatcher (RG_dispatch) is just another running module of the control architecture that loads the tasks from the XML files and executes them. Therefore, RG_dispatch coordinates the activities of the middleware modules according to the sequences defined in the MIPNs.

FIGURE 13 shows the general architecture for the tour-guide robot application developed using RIDE. The modules of the architecture are connected on two different

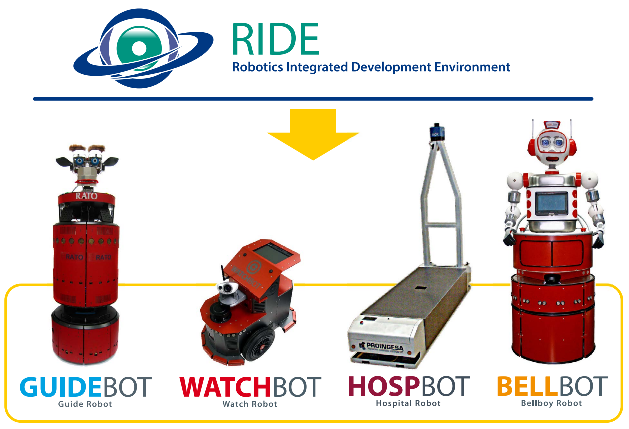


FIGURE 12. GuideBot, WatchBot, HospBot, and BellBot.

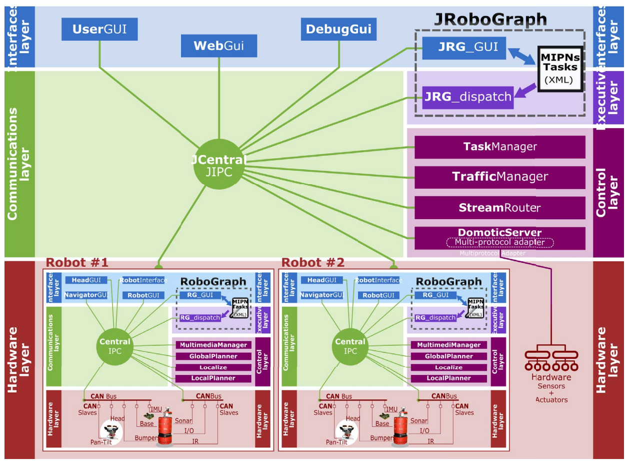


FIGURE 13. RIDE multi-robot system architecture.

levels. The lowest level includes the on-board modules of each robot connected using IPC (FIGURE 3) and the highest level is the global system connected using JIPC where each robot is considered as one single module. Each set of modules has its own RoboGraph modules because there are tasks defined on both levels. Global tasks might include several robots, other elements of the application, and users. Robot tasks are executed by only one robot and most of them are part of a global task. This means that some MIPNs in the hierarchy can be executed by different RoboGraph dispatch modules distributed in different machines by the system. Both kinds of tasks are defined in a similar way using the RG_editor as MIPNs. However, the robot tasks use IPC messages while the global tasks use JIPC messages.

In both communication systems (IPC and JIPC), modules share information using a publish/subscribe messages paradigm. However, while IPC uses broadcast messages, JIPC can publish messages to and subscribe to messages from a specific module. In IPC, modules subscribed to a kind of message will receive all the messages of that kind that any module publishes. This is a nice feature when thinking of a

ROS
Robotics Operating System
ROBLE
Robotización y Logística Inteligente
SmartElderlyCar
Smart Car for Elderly People
ColRobot
Collaborative Robotics for Assembly

FIGURE 14. ROBLE, SmartElderlyCar, and ColRobot projects developed under ROS.

single robot control. However, due to the symmetry of the multi-robot multi-user framework (FIGURE 13), a problem arises when dispatch wants to send a command to only one robot since all the robots will get the message. JIPC was designed to avoid that problem in multi-robot systems.

2) APPLICATIONS THAT USE ROS

Applications that use ROS include the following (see also FIGURE 14):

- **SmartElderlyCar** [55]. An autonomous car designed for elderly or disabled people. MIPNs were used to define the behavior expected for the car according to the traffic rules in different scenarios.
- **ROBLE** [56]. In this case, tugger trains were used to transport stock material to supply different working stations with different materials efficiently. The first version was developed under RIDE and a final version was implemented on ROS.
- **ColRobot** [57]. European Project centered on using collaborative mobile manipulators for assembly and kitting in smart manufacturing.
- **PROFETA**. Application centred on the manipulation of preimpregnated composite materials in the manufacturing of aircraft parts.
- **Mari4_YARD** [58]. This is an ongoing European Project were different robotics technologies will be tested in shipbuilding industry.
- **5R Network** [59]. Collaboration network between competence centres in robotics to foster the technology transfer to manufacturing companies.

Even though RoboGraph was initially developed inside RIDE, the communication capabilities were implemented in an external layer that facilitates the extension to other architectures. As in most robot control frameworks, ROS provides an inter-process communication interface with different type of connections between modules that include persistent connections to a service when high-throughput is needed. However, only the publish/subscribe paradigm was used to exchange information between RoboGraph and other ROS modules. Besides the communication interface, the rest of RoboGraph is the same with two modules (processes); one to load and execute the nets and subnets (RG_dispatch) and the other to edit, validate and debug them (RG_GUI).

For all these applications, MIPNs model only the executive layer. For example, in the SmartElderlyCar project, the main net identifies the main driving situations according to the section of the road and events perceived by the perception layer. Some of these scenarios include "*follow a lane*", "*overtake a vehicle*", "*enter an intersection*", etc. At the same time, each situation can be modeled by one or more subnets. In the ROBLE project, the main net identifies different events that start different tasks. For example, the event that "*there is a new empty cart to remove*" or "*some work station needs new material*". As in the previous case, each task is modeled by one or more subnets. In the ColBot project, RoboGraph was used to orchestrate the ROS-based control architecture of a mobile manipulator operating inside a van to screw the partition wall. Finally, Mari4_YARD is still an ongoing project where RoboGraph is planned to be used to orchestrate the control architecture of a mobile manipulator performing outfitting operations inside superblocks.

V. CONCLUSION

In this paper, a new extended hierarchic PN model has been presented. The model expands the functionality provided by the current PN-based models. It was designed for complex applications whose control functions are distributed in several processes that use a well established message-based communications to exchange information. This is the case for example of the applications developed under the Robotics Operating System (ROS).

Once the hierarchic model was defined, the means to analyze it have been provided. Aside from the MIPN model itself, new properties such as the concept of *terminability* have been defined. Some methods to evaluate these properties in the MIPN models were also developed. The evaluation process is simplified using the mathematical relations between global properties of a hierarchic MIPNs system and individual properties of each MIPN. This allows to atomically analyze each net separately and then build up the properties of the whole system. This capability for building up properties through hierarchy levels simplifies the analysis of any complex multi-level MIPN system and enhances the reusability of subtasks modeled with MIPNs. Some properties similar to *terminability*, have been previously proposed by other researchers. The closest one is the concept of *soundness* in WF-nets. However, WF-nets design implications are different [13], [60]. A WF-net requires the Petri net to have (i) a single Start place, (ii) a single End place, and (iii) each node must be on one path from Start to End. Those restrictions make WF-nets less versatile than MIPNs and its termination rule, which allows for arbitrary final markings, as needed by the designer, and has an established behavior on termination (publish a termination message).

Software tools to check the MIPNs properties have been added to our IDE (RoboGraph). Most of these tools are analysis modules imported directly from PIPE [61] such as the classification, reachability graph and state space analysis. These modules provide the analysis and properties of each individual net. Using the results on Section III-C we have added new modules that provide the global properties of a hierarchic MIPN based on the individual properties of each subnet. The MIPNs are edited using the RoboGraph Editor and, once the model is defined, it can be analyzed providing a report. The report in the case of problems such as deadlocks includes the sequence of events that leads the system to the deadlock.

Beyond the modeling and analysis, MIPN model is aimed for auto generation of code for the modeled tasks. Several methods and tools for automatic generation of control code after a formal model such as GRAFCET [34] have been designed. Here the model itself (MIPN) is the code that is executed by our tool (RoboGraph). In this way, RoboGraph can be seen as the "Virtual Machine" that can execute the nets included in the MIPN model.

The actual utility of the model was motivated and verified by the use in platforms such as **RIDE** [22], [24] and ROS [41]. Several applications have been developed using both platforms as described in Section IV-C. In these applications, besides task modeling, verification, and analysis, MIPNs have also been used for task control definition, becoming an efficient visual task programming language, task sequencing, and task monitoring. These are some of the main advantages of this approach:

- Reduce development time. Once the model is created, it can also serve as the control program. For example, in our case, RoboGraph is in charge to execute directly the nets. There is no need to generate extra code.
- Simple and intuitive programming. Almost everybody that has worked or learn to use IEC 61131-3 compliant programming environments (Siemens S7 Graph, Graphcet, etc.) will be able to program new tasks using MIPNs with minimal training.
- Automatic verification and analysis. The MIPN properties described in this paper can be easily implemented in a graphical IDE tool. We have included this functionality in RoboGraph so that after editing the nets, it can be automatically analyzed producing a report with counterexamples to identify the error source when some structural problem is detected.
- Intuitive graphical debugging. The same MIPN model is also used to follow the state of the system because the RoboGraph GUI shows the current marking of all the running nets. If the system gets stuck, it is quite easy to identify the event or events is waiting for. Those events correspond to the conditions of the enabled transitions.
- Simulation and report generation. Since a net models a task or subtask, statistics on the execution of the net

or sections of the same are easily generated through a simple event simulation. We have been using this feature to establish the minimum size of a fleet of robots in multi-robot applications.

ACKNOWLEDGMENT

The authors would like to thank all the people that contributed in the different projects, where MIPNs were used through the RoboGraph tool. Their feedback helped not only to fix some initial problems but also to improve and adapt the model to new requirements making it more versatile.

REFERENCES

[1] H. Ergin, E. Syriani, and J. Gray, "Design pattern oriented development of model transformations," *Comput. Lang., Syst. Struct.*, vol. 46, pp. 106–139, Nov. 2016.

[2] A. R. da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Comput. Lang., Syst. Struct.*, vol. 43, pp. 139–155, Oct. 2015.

[3] P. Gómez-Abajo, E. Guerra, and J. D. Lara, "A domain-specific language for model mutation and its application to the automated generation of exercises," *Comput. Lang., Syst. Struct.*, vol. 49, pp. 152–173, Sep. 2017.

[4] D. Méndez-Acuña, J. A. Galindo, T. Degueule, B. Combemale, and B. Baudry, "Leveraging software product lines engineering in the development of external DSLs: A systematic literature review," *Comput. Lang., Syst. Struct.*, vol. 46, pp. 206–235, Nov. 2016.

[5] I. Kovalenko, D. Tilbury, and K. Barton, "The model-based product agent: A control oriented architecture for intelligent products in multi-agent manufacturing systems," *Control Eng. Pract.*, vol. 86, pp. 105–117, May 2019.

[6] F. Basile, P. Chiacchio, and E. Di Marino, "An auction-based approach to control automated warehouses using smart vehicles," *Control Eng. Pract.*, vol. 90, pp. 285–300, Sep. 2019.

[7] M. F. Geronimo, E. G. H. Martinez, E. D. F. Vazquez, J. J. F. Godoy, and G. F. Anaya, "A multiagent systems with Petri net approach for simulation of urban traffic networks," *Comput., Environ. Urban Syst.*, vol. 89, Sep. 2021, Art. no. 101662.

[8] Y.-W. Si, V.-I. Chan, M. Dumas, and D. Zhang, "A Petri nets based generic genetic algorithm framework for resource optimization in business processes," *Simul. Model. Pract. Theory*, vol. 86, pp. 72–101, Aug. 2018.

[9] D. F. Simon, M. Teixeira, and J. P. da Costa, "Availability estimation in photovoltaic generation systems using timed Petri net simulation models," *Int. J. Electr. Power Energy Syst.*, vol. 137, May 2022, Art. no. 106897.

[10] J. B. Oliveira, M. Jin, R. S. Lima, J. E. Kobza, and J. A. B. Montevechi, "The role of simulation and optimization methods in supply chain risk management: Performance and review standpoints," *Simul. Model. Pract. Theory*, vol. 92, pp. 17–44, Apr. 2019.

[11] W. M. Zuberek and I. Bluemke, "Hierarchies of place/transition refinements in Petri nets," in *Proc. IEEE Conf. Emerg. Technol. Factory Automat. (ETFA)*, Nov. 1996, pp. 355–360.

[12] M. Figat and C. Zielinski, "Methodology of designing multi-agent robot control systems utilising hierarchical Petri nets," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 3363–3369.

[13] C. Dufourd, A. Finkel, and P. Schnoebelen, "Reset nets between decidability and undecidability," in *Proc. 25th Int. Colloq. Automata, Lang. Program.* London, U.K.: Springer-Verlag, 1998, pp. 103–115.

[14] K. Jensen, "Coloured Petri nets and the invariant-method," *Theor. Comput. Sci.*, vol. 14, no. 3, pp. 317–336, 1981.

[15] J. Wang, *Timed Petri Nets: Theory and Application*, vol. 9. Springer & Business Media, 2012.

[16] M. Moalla, J. Pulou, and J. Sifakis, "Synchronized Petri nets: A model for the description of non-autonomous systems," in *Proc. MFCS*, 1978, pp. 374–384.

[17] G. Frey, "Analysis of Petri net based control algorithms–basic properties," in *Proc. Amer. Control Conf. (ACC)*, Jun. 2000, pp. 3172–3176.

[18] R. König and L. Quäck, *Petri-netze in der steuerungs und digitaltechnik*. Oldenbourg Verlag, 1988.

[19] I. Grobelna and M. Adamski, "Model checking of control interpreted Petri nets," in *Proc. 18th Int. Conf. Mixed Design Integr. Circuits Syst.*, Jun. 2011, pp. 621–626.

[20] M. Uzam and A. H. Jones, "Discrete event control system design using automation Petri nets and their ladder diagram implementation," *Int. J. Adv. Manuf. Technol.*, vol. 14, no. 10, pp. 716–728, Oct. 1998.

[21] I. A. Fernández, J. C. M. Cortabarría, L. E. Echeverría, "Petri net implementation in programmable logic controllers: Methodology for development and validation," in *Proc. IEEE 19th World Symp. Appl. Mach. Intell. Inform. (SAMI)*, Jan. 2021, pp. 000015–000020.

[22] *RIDE: Robotics Integrated Development Environment*, Mobile Robots Intell. Syst. Group, Univ. Vigo, Vigo, Spain, 2010.

[23] J. López, E. Zalama, and J. Gómez-García-Bermejo, "A simulation and control framework for AGV based transport systems," *Simul. Model. Pract. Theory*, vol. 116, Apr. 2022, Art. no. 102430.

[24] J. López, D. Pérez, and E. Zalama, "A framework for building mobile single and multi-robot applications," *Robot. Auto. Syst.*, vol. 59, nos. 3–4, pp. 151–162, Mar. 2011.

[25] A. C. Gaona, J. M. Chavez, and C. R. Vazquez, "RCPetri: A MATLAB app for the synthesis of Petri net regulation controllers for industrial automation," in *Proc. 26th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2021, pp. 1–7.

[26] J. López, D. Pérez, E. Paz, and A. Santana, "Watchbot: A building maintenance and surveillance system based on autonomous robots," *Robot. Auton. Syst.*, vol. 61, no. 12, pp. 1559–1571, Dec. 2013.

[27] J. L. Fernandez, R. Sanz, E. Paz, and C. Alonso, "Using hierarchical binary Petri nets to build robust mobile robot applications: RoboGraph," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2008, pp. 1372–1377.

[28] M. Buehler, K. Iagnemma, and S. Singh, Eds., *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic* (Springer Tracts in Advanced Robotics), 56th ed. Springer, 2010.

[29] C. Urmson, J. A. Bagnell, C. R. Baker, M. Hebert, A. Kelly, R. Rajkumar, P. E. Rybski, S. Scherer, R. Simmons, and S. Singh, "Tartan racing: A multi-modal approach to the darpa urban challenge," Robot. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., 2007.

[30] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, and D. Johnston, "Junior: The Stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569–597, 2008.

[31] A. Kurt and Ü. Özgüner, "Hierarchical finite state machines for autonomous mobile systems," *Control Eng. Pract.*, vol. 21, no. 2, pp. 184–194, 2013.

[32] P. Ogren, "Increasing modularity of UAV control systems using computer game behavior trees," in *Proc. AIAA Guid., Navigat., Control Conf.*, Aug. 2012, p. 4458.

[33] M. Colledanchise and P. Ogren, "How behavior trees modularize robustness and safety in hybrid systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2014, pp. 1482–1488.

[34] F. Schumacher and A. Fay, "Formal representation of GRAFCET to automatically generate control code," *Control Eng. Pract.*, vol. 33, pp. 84–93, Dec. 2014.

[35] J. M. B. Braman and R. M. Murray, "Bisimulation conversion and verification procedure for goal-based control systems," *Formal Methods Syst. Des.*, vol. 38, no. 1, pp. 62–95, Feb. 2011.

[36] J. Li, X. Dai, Z. Meng, J. Dou, and X. Guan, "Rapid design and reconfiguration of Petri net models for reconfigurable manufacturing cells with improved net rewriting systems and activity diagrams," *Comput. Ind. Eng.*, vol. 57, no. 4, pp. 1431–1451, Nov. 2009.

[37] M. Vierhauser, R. Rabiser, P. Grünbacher, K. Seyerlehner, S. Wallner, and H. Zeisel, "ReMinds : A flexible runtime monitoring framework for systems of systems," *J. Syst. Softw.*, vol. 112, pp. 123–136, Feb. 2016.

[38] H. Bruyninckx, "Robotics software framework harmonization by means of component composability benchmarks. The manifolds of four," RICS Repository, BRICS Deliverable D8.1, Most, Tech. Rep., 2010, pp. 1–12.

[39] R. Simmons and D. James, "Inter-process communication (IPC). A reference manual," School Comput. Sci./Robot. Inst., CMU (Carnegie Mellon Univ.), Pittsburgh, PA, USA, Tech. Rep., Aug. 2001.

[40] J. López, D. Pérez, I. Vaamonde, E. Paz, A. Vaamonde, and J. Cabaleiro, "Building a warehouse control system using ride," in *Proc. Robot: 2nd Iberian Robot. Conf.*, Cham, Switzerland: Springer, 2016, pp. 757–768.

[41] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, Kobe, Japan, vol. 3, 2009, p. 5.

[42] P. Buchholz, "Hierarchical high level Petri nets for complex system analysis," in *Application and Theory of Petri Nets* (Lecture Notes in Computer Science), vol. 815, R. Valette, Ed. Berlin, Germany: Springer, 1994, pp. 119–138.

[43] G. Frey and L. Litz, "A measure for transparency in net based control algorithms," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 1999, pp. 887–892.

[44] S. Klein, X. Weng, G. Frey, J.-J. Lesage, and L. Litz, "Controller design for an FMS using signal interpreted Petri nets and SFC: Validation of both descriptions via model-checking," in *Proc. Amer. Control Conf.*, vol. 5, May 2002, pp. 4141–4146.

[45] A. M. Atto, C. Martinez, and S. Amari, "Control of discrete event systems with respect to strict duration: Supervision of an industrial manufacturing plant," *Comput. Ind. Eng.*, vol. 61, no. 4, pp. 1149–1159, Nov. 2011.

[46] A. Chandler, A. Heyworth, L. Blair, and D. Seward, "Testing Petri nets for mobile robots using groebner basis," *Proc. 21st Int. Conf. Appl. Theory Petri Nets*, Aarhus, Denmark, Jul. 2000.

[47] M. D. Jeng, "Petri nets for modeling automated manufacturing systems with error recovery," *IEEE Trans. Robot. Autom.*, vol. 13, no. 5, pp. 752–760, Oct. 1997.

[48] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[49] P. Buchholz and P. Kemper, "Hierarchical reachability graph generation for Petri nets," *Formal Methods Syst. Des.*, vol. 21, no. 3, pp. 281–315, Nov. 2002.

[50] R. M. Karp and R. E. Miller, "Parallel program schemata," *J. Comput. Syst. Sci.*, vol. 3, no. 2, pp. 147–195, May 1969.

[51] J. L. Fernández, D. P. Losada, M. G. Prado, and A. P. Domonte, "Monitoring and debugging distributed autonomous systems using Petri nets," in *Proc. 15th World Multi-Conf. Systemics, Cybern. Inform. (WMSCI)*, vol. 1, Jul. 2011, pp. 216–221.

[52] J. L. Fernández, D. P. Losada, R. Pinillos, S. Domínguez, E. Zalama, and J. Gómez-García-Bermejo, "Diseño y desarrollo de un sistema de transporte reconfigurable para entornos hospitalarios," *Revista Iberoamericana de Automática e Informática Industrial.*, vol. 9, no. 1, pp. 57–68, 2012.

[53] J. López, D. Pérez, E. Zalama, and J. Gómez-García-Bermejo, "BellBot—A hotel assistant system using mobile robots," *Int. J. Adv. Robotic Syst.*, vol. 10, no. 1, p. 40, Jan. 2013.

[54] J. López, D. Pérez, M. Santos, and M. Cacho, "GuideBot. A tour guide system based on mobile robots," *Int. J. Adv. Robotic Syst.*, vol. 10, no. 11, p. 381, Nov. 2013.

[55] J. López, P. Sánchez-Vilariño, R. Sanz, and E. Paz, "Implementing autonomous driving behaviors using a message driven Petri net framework," *Sensors*, vol. 20, no. 2, p. 449, Jan. 2020.

[56] R. Samaniego, R. Rodríguez, F. Vázquez, and J. López, "Efficient path planing for articulated vehicles in cluttered environments," *Sensors*, vol. 20, no. 23, p. 6821, Nov. 2020.

[57] (Nov. 17, 2020). *Collaborative Robotics for Assembly and Kitting in Smart Manufacturing*. Accessed: Jan. 4, 2021. [Online]. Available: https://cordis.europa.eu/project/id/688807

[58] (Sep. 6, 2019). *User-Centric Solutions for a Flexible and Modular Manufacturing in Small and Medium-Sized Shipyards*. Accessed: May 18, 2022. [Online]. Available: https://cordis.europa.eu/project/id/101006798

[59] (2021). *Network of Spanish Competence Centres in Robotic Technologies for Manufacturing*. Accessed: May 18, 2022. [Online]. Available: https://red5r.es/

[60] C. Li, J. Ge, L. Huang, H. Hu, B. Wu, H. Hu, and B. Luo, "Software cybernetics in BPM: Modeling software behavior as feedback for evolution by a novel discovery method based on augmented event logs," *J. Syst. Softw.*, vol. 124, pp. 260–273, Feb. 2017.

[61] P. Bonet, C. M. Lladó, R. Puijaner, and W. J. Knottenbelt, "Pipe v2. 5: A Petri net tool for performance modelling," in *Proc. 23rd Latin Amer. Conf. Informat. (CLEI)*, 2007, pp. 1–12.

JOAQUÍN LÓPEZ received the M.S. degree in telecommunications engineering from the University of Vigo, Spain, in 1992, and the Ph.D. degree from the Department of Systems Engineering and Automation, University of Vigo, in 2000. He spent two years as a Visiting Researcher (first year) and a Special Faculty (second year) at Carnegie Mellon University's Robotics Institute. He is currently an Associate Professor with the School of Industrial Engineering, University of Vigo. He is the author and the coauthor of over 40 journal articles in the field of mobile robotics and artificial intelligence. He has participated in several funded research projects during the last 15 years.

ALEJANDRO SANTANA-ALONSO received the M.S. degree in electrical engineering from the University of Vigo, Spain, in 2011. He is currently pursuing the Ph.D. degree, with a focus on complex tasks definition, analysis, and implementation for mobile robotics applications. He worked at the System Engineering Department, University of Vigo, where he is developing mobile robotics applications and control systems. He also worked at ERP Development as a CTO for three years. From 2016 to 2019, he was employed at the AIMEN Technology Centre in research and development in robotics and control area. His research interest includes mobile and collaborative robotics.

DIEGO PÉREZ LOSADA received the M.S. degree in electrical engineering and the Ph.D. degree in robotics and control engineering from the University of Vigo, in 2006 and 2012, respectively. His work focuses on robotics architectures, mobile robot applications, and hardware integration in mobile robot control architectures, with more than 16 years of experience in the design of applications-based in mobile robots and industrial manipulators. He has worked on several national and European research and development projects within the University of Vigo and the AIMEN Technology Centre. Currently, he is leading the Advanced Robotics Technologies and Applications Research Team (ARTA), Smart Systems and Smart Manufacturing Unit, AIMEN Technology Centre.