

# Safe Adaptation of Cobotic Cells based on Petri Nets

Sebastian Ebert

sebastian.ebert@tu-dresden.de

Technische Universität Dresden

Dresden, Germany

## ABSTRACT

Collaborative robotic cells combine human skills with the latest advancements in robotic accuracy and reliability. Cobotic cell parts are distributed and adapt their behavior to changing tasks and environments. The specific missions of cobotic cells, depend on their field of application, but are critical for human safety, which introduces complexity, increasing testing and development effort. Component-based software engineering is used to manage complexity, but ensuring safety and correctness requires verification and validation, which is complex and demanding to re-ensure, when composed behavior changes. This also applies to the widely used middleware Robot Operating System (ROS), where existing approaches only model high level communication or integrate models. Also, verification of cobotic cell must reflect their context-adaptivity, to check safety critical reactions to contexts-changes. To overcome these inhibitors, a model-driven development approach based on Petri nets is proposed, modeling central aspects of ROS-based cobotic cells. By using formal models, the testing effort at development time is reduced, because global behavior remains formally proven, and only local components have to be retested. Within this work, the plans for this model-driven software approach are reported.

## CCS CONCEPTS

• **Software and its engineering** → **Petri nets; Model-driven software engineering**; *Abstraction, modeling and modularity*; • **Human-centered computing** → Collaborative interaction.

## KEYWORDS

Robotics, Robot Operating System, Petri Nets, Context Adaptation

### ACM Reference Format:

Sebastian Ebert. 2022. Safe Adaptation of Cobotic Cells based on Petri Nets. In *17th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '22)*, May 18–23, 2022, PITTSBURGH, PA, USA. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3524844.3528075>

## 1 INTRODUCTION

Design, development and extension of robotic software systems with human co-working is a complex task. Such cobotic cells [13] consist of physically distributed components [20] sharing space with human co-workers, necessitating adaptation to changing environments and tasks [15]. Consequently, cobotic software must be guaranteed to function correctly (following system specification) and safe, by not causing harm to humans or machines while also self-adapting to changing contexts [26, 27]. To ensure certifiability of cobotic systems, safety and correctness have to be verified and validated. Dealing with these two aspects introduces complexity into

cobotic software, increasing needed development effort and amount of testing [7]. Component-based software engineering (CBSE) provides decoupled component models, communication middlewares and libraries [6] to deal with the introduced complexity. Such a popular component based framework is the Robot Operating System (ROS), facilitating the development of robotic applications [32]. ROS allows the creation of components controlling robotic hardware and pure software components, by providing a structured communications layer, with various communication patterns such as a publish-subscribe. A ROS-based process is called a node, and inside nodes data-processing and control takes place. However, CBSE and therefore also ROS focuses on improving reusability of implementations, and on making the validation process done by software tests on component level more efficient by enabling reusing validation results of a component when changing other components. If component interfaces or communication patterns are changed, re-ensuring validity of composed behavior is a complex and time-intensive task. Model-driven software development based on formally defined models addresses this complexity, by enabling the specification of reusable and implementation independent system behaviors, requirements and possible run-time adaptations, providing means for analysis, verification purposes through system life-cycles, minimizing testing efforts. Existing approaches, such as [11, 24, 25], model only parts of ROS-based systems or focus on specific programming or communication aspects. Therefore, a model-driven approach reflecting the complete ROS communication, adaptation-aspects and high-level control flow, enabling verifiable, efficient development and safe adaptation of ROS components, is proposed here.

## 2 RESEARCH CHALLENGES

Selection and adaptation of a suitable modeling approach, grants the abilities to unify modeling and simplify verification, but also requires solving several challenges. Structural aspects of ROS systems such as distributed nodes containing models, packages and parameters, as well as communication mechanisms, must be part of this approach to enable analysis of incorrect system behavior.

**RC1: Integration of Safety in Cobotic CBSE.** Enabling model-based integration of safety requires dealing with two classes of safety issues. Firstly, a safe system has to handle external events, occurring expected or unexpected [34], within the proximity of a system. Expected external events occur at run-time and are already known at development time. Unexpected external events are unknown at development time, which means that one has to monitor system models during run-time, to correct these them based on the results. Secondly, safe systems have to handle internal expected and unexpected events [37]. Expected internal events are anticipated and handled explicitly. Unexpected events, cover incorrect system behavior, to be handled on divergence from the specification. It is necessary to integrate a modeling approach, handling these issues,

into a model-driven chain, to guarantee that verified properties are still valid within the final software.

**RC2: Formal Interfaces for Safe Self-Adaptation.** Any computationally accessible information can be considered as context, providing a basis for the aforementioned events [19, 35]. Thus, a cobotic cell must be aware of context changes and has to adapt its behavior to it. A safety-focused modeling approach for cobotic cells, has to provide sound interfaces for contextual information, so that a system and adaptations to it remain verifiable. These spots of adaptation specifications must be kept generic, so that any adaptation controlling technology can be connected, and thus act as a framework for safe self-adaptations, verified independently of the decision to carrying them out. Secondly, adaptations also happen within the development time of cobotic cells on the basis of code changes. Software is constantly being further developed, which is subject of re-verification. If this is not facilitated, needed development effort and amount of testing to be done is increased.

**RC3: Compatibility for External Components.** Using a single modeling approach allows covering several aspects of cobotic cells, such as communication and application workflows. However, in real-world applications, because not every aspect of cobotic cells can be modeled with the same technique, integrating existing systems and components is necessary to use their tested abilities. In the following, both of these types are cumulated to *external components*, which need to be integrated, in a tightly coupled manner into verification to increase the meaningfulness of it, so that the handling of safety issues as seen in challenge RC1 is still possible. This is in contrast to the formal interfaces of challenge RC2, providing generic integration for contextual information.

**Development of Safe Cobotic Cells.** Based on the challenges identified, requirements can be concluded, which are necessary for the development of safe adaptive cobotic cells. **R1: Model-driven development** of cell structure and behavior, to facilitate inheriting verified model properties to the final cobotic cell software (RC1 and RC2). **R2: Integration of robotic middleware concepts** such as communication and structure within the development phase, to have a more meaningful verification (RC1). **R3: Integration of context-adaptation** into the development process, to enable verified run-time adaptation based on formal interfaces (RC2). **R4: Ensuring compatibility** with external systems and models to enable efficient integration of components into the verification process (RC3). **R5: Reduce development effort** by reducing testing effort with verification.

### 3 RELATED WORK

Modeling techniques and model-driven systems for ROS based applications are already facilitated by various approaches. Therefore, formal and specifically Petri net-based approaches are analyzed in the following, by applying the previously defined criteria to ROS, as shown in table 1.

**Formal Model-based ROS Approaches.** Halder et al. [17] models and verifies ROS systems using real time properties, based on timed automata models, whose model checking is done based on Uppaal [2]. More in detail, the approach allows modeling the internal concepts of ROS nodes, such as the spinning mechanism, queues and control loops. However the approach focuses only on

**Table 1: Related approaches, where full support is denoted with ■, partial support with ▣ and no support with □.**

Approach	ROS-Structure Communication	Context- Adaptation	External Components	Model-driven Development	Formal Approach
Halder [17]	■ / ▣	□	□	□	Automata
Wang [36]	■ / ▣	□	□	■	Automata
Cheng [5]	▣ / □	■	■	■	GSN
Kortik [22]	▣ / ▣	□	■	□	Linear Logic
Zander [38]	▣ / □	□	□	■	Ontologies
Lesire [24]	□ / ▣	□	■	■	Petri nets
Dondrup [11]	□ / ▣	□	□	■	Petri nets
DiNeROS	■ / ■	■	■	■	Petri nets

topic-based communication between nodes and also not on context adaptation and integration of external components. Wang et al. [36] proposes a model-based design method for ROS based systems, that automatically generates executable C++ ROS code, based on specification realized by a network of timed automata. Again, the verification is done with Uppaal, to check for formalized safety requirements. The overall modeling power in terms of ROS is similar to Halder et al., but additionally a model-driven chain supports the development of applications. Cheng et al. [5] considers especially self-adaptive behavior and ensuring satisfaction of safety requirements across adaptations. Structuring Notation (GSN) models are used to specify safety requirements. Based on the GSN specification, ROS launch files are generated that are coupling an adaption controlling node to application nodes and GSN based models. While system run-time adaptation actions are published to topics subscribed by nodes executing the adaptations. However, nodes are only supported abstractly and communication only in terms of topics. Formal verification based on linear logic is realized by Kortik et al. [22], focusing on verification of correctness and not on designing safe systems. The desired behavior of each node is encoded within linear logic and then verified, by checking computational graph consistency and type consistencies. Thus, structural aspects in terms of nodes and communication aspects in terms of topics are only modeled in highly abstract way. Zander et al. [38] comes up with a model-driven chain based on ontological semantics. Developers first create a local, not formal model, by specifying capabilities and interfaces of for robotic software components. A generator parses the model and generates executable code skeletons, containing nodes, to be completed by developers.

**Petri Net-based ROS Approaches.** Petri nets formally model behavior and structure of systems [8]. A Petri net is a bipartite graph consisting of places and transitions, connected by directed arcs [8]. Places may contain a discrete number of tokens, consumed by firing transition and recreated within their outputs. Pages encapsulate hierarchically parts of Petri nets, connected by references. Petri nets are able to model concurrent, asynchronous, distributed, parallel and non-deterministic properties of systems, which are also subject of cobotic applications [21], where potentially parallel and collaborative performed actions [23] contribute to a globally, between

applications parts distributed, state. Non-determinism is introduced in cobotic cells by their human parts and environment [14]. Cobot cells feature context-depended behavior for changing context such as human presence [16], system evolution, environments, and non-autonomous behavior. Thus, Petri nets are a good candidate for modeling and developing cobotic, and therefore also ROS-based systems, including verification, validation and code generation, which is already addressed by multiple works [26]. ASPiC by Lesire et al. [24] allows modeling of robotic skills using control-flow Petri net models called skill Petri nets, which are parameterized by required resources (locks, inputs, outputs). The approach is an acting system based on the composition of these Skill Petri nets with Petri nets acting as sound compositional control flow operators to action plans. To be able to connect to the ROS middleware, the approach includes the modeling and, thus, binding of ROS action interfaces. Unfortunately, no structural aspects are included, and the whole approach runs on one ROS node. Dondrup et al. [11] deals with handling interaction between agents (devices or robots) and humans, by modeling interaction patterns as Petri nets mapping to deterministic finite state machines. A developer specifies Petri nets, based on a modeling language, which is used for the automatic generation of the mapped Petri nets. Transitions, of the Petri nets, are bound to ROS action servers and triggered whenever a bound transition fires. Summarized, all analyzed approaches do not fully support ROS features and only a few provide context-adaptation and external components, but model-driven chains are common.

**Petri Nets in Robotics.** Beside approaches, which are modeling ROS concepts, Petri nets are widely used in modeling and controlling distributed robotic systems. Moutinho [28] proposes distributed locally synchronous Petri nets connected with asynchronous communication channels. The work of Bera [3] describes an architectural framework for distributed and communicating Petri nets based on fixed interaction patterns. Robotic Task Models [30] are Petri nets controlled by central coordinators running robotic workflows consisting of primitive tasks. The approach for Petri net based task planning by Kotb [23] allows distributed robotic agents to collaborate based on Petri net modeled abilities. Figat [12] presents in his work a way to describe robotic workflow execution, and how to generate a distributed ROS system based on it, although itself does not model ROS concepts.

## 4 PROPOSED SOLUTION

In the following, an approach is presented, treating the pointed out requirements, by providing a modeling formalism, a model-driven development chain and a software architecture collectively referred to as *Distributed Petri Nets to ROS* (DiNeROS).

**Modeling Approach** ROS-based Systems specialize the aforementioned cobotic cell properties in terms of communication, where ROS provides synchronous communication (services) and asynchronous communication (topics, actions), and distribution by nodes contained in packages, configured by parameters. Thus, the Petri net formalism is also a good candidate for ROS-based systems. However, there are missing aspects, but fortunately Petri nets are extensible [18, 31, 33]. Modeling ROS-based communicating nets, is done with channel submodels, parameterized by capacities and encapsulated to hide their complexity. Therefore, specific submodels

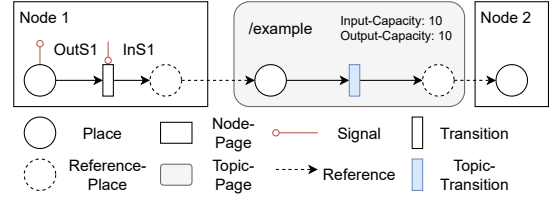


Figure 1: Example DiNeROS Global ROS Petri Net

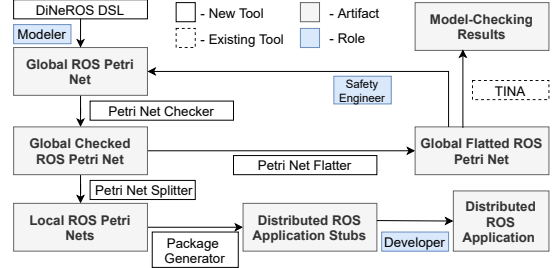


Figure 2: DiNeROS Development Chain

for each communication type, such as topics exemplary shown in figure 1, are included, where the topic transition maps to a p/t-net, representing the underlying ROS mechanisms. Parallel communication type specific behavior of message creation and processing is realized by fixed Petri net patterns. Distribution specifics, such as physical locations, packages and nodes, are modeled by pages with attached metadata. Context-specific and non-autonomous behavior, is added based on toggling transitions with binary input signals and giving feedback based on integer valued output signals on places providing a generic interface for adaptations, shown within left node in figure 1, treating challenge R3. Including new concepts, such as channels and signals, is done in a sound way, by providing mappings to place-transitions-nets (P/T-nets) [9], for which formal properties can be guaranteed, and a tooling landscape exists, solving R2. Sending primitive tokens between node-contained distributed Petri nets based on channels does not provide a powerful tool, because applications need to exchange data. Using colored Petri nets [31], would enable tokens containing data, but fewer properties are guaranteed. Thus, binding data to tokens is done similar to balloon types [1], by following a fixed, defined data format, representing a shared state. Therefore, this tokens have no influence on execution semantics and model checking.

**Development Chain.** The modeling formalism just described forms the basis for a model-driven development chain, which is shown in figure 2 and that allows a safe, because verified, ROS applications to be developed on the basis of Petri nets. Within the chain three development roles are existing: the *modeler*, a domain expert creating Petri net models of the cobotic cell. Second, the *developer*, implementing application logic, and third the *safety-engineer*, an expert in safety which analysis possible safety threats and suggests solutions. At the beginning of the chain, models, such as workflows and safety models are defined as a global Petri net, encapsulating ROS aspects as part of its formalism. The specification is created with a Domain Specific Language (DSL) facilitating development speed, and containing run-time adaptation spots based on signals as interface, allowing to couple knowledge bases, such as hybrid Petri nets [10], teaching data and simulations, for adaptation actions.

Based on this DSL-created model, the *global ROS Petri net* is automatically generated, which is input for the *Petri Net Checker* validating the model to exclude possible syntactic errors. After that, the global net is flattened to a P/T-net by the *Petri Net Flatter* tool and is input to the model checking tool TINA [4]. Based on this, global system behavior is evaluated by a *safety-engineer* against safety properties, based on state-reachability, liveness-, path-, boundedness-, and invariant- analysis. A guidance tool helps the engineer to understand the outputs of TINA within the context of DiNeROS. If a model does not fulfill all safety requirements, it is updated by the *safety-engineer*, and the chain is restarted with it. Subsequently, the global net is automatically split into multiple *local ROS Petri nets* contained in ROS nodes, depending on location metadata attached to Petri net elements, by among other things splitting transitions representing communication channels. In the end, implementation stubs for ROS nodes are automatically generated, with the *Package Generator*, containing local ROS Petri nets as run-time models, based on templates provided either by the *developer* or by DiNeROS. This Petri net stub code is connected by a *developer* to application code based on automatically generated callbacks and signals, providing the end of the model-driven chain resolving R1, by coupling verifiable models and code, reducing development effort (R5).

**Architecture.** The system resulting from the model-driven chain follows a centralized architecture with a cobotic coordinator as central server and peer-to-peer communication between ROS-based nodes. First of all, the architecture provides nodes, which are either DiNeROS or adaptation nodes. *DiNeROS nodes* are basically local Petri net controlled ROS-Nodes, created as output of the *Package Generator*. Hence, these nodes have their own state, contained within tokens, which can only be accessed via the Petri net modeled ROS communication interfaces and signals. They deal with various concerns like controlling robotic workflows or processing sensor data. *Adaptation nodes*, provide control and knowledge for run-time self-adaptations of cobotic cells, treating challenge R3. They are coupling self-adaptation control mechanisms [29], to DiNeROS nodes via the cobotic coordinator. The input for adaptation nodes are automatically updated output signals and their output are possible new input signal values on detected changes. The *cobotic coordinator* (CC) is the architecture's central element extending ROS core functionalities. Firstly, the CC handles signals and self-adaptation processes by connecting adaptation- and DiNeROS-nodes, enabling combining and applying adaptation node outputs via input-signals, resulting in modified model behavior. Secondly, liveness guarantees for all nodes and connected components are provided, based on a distributed Petri net collecting node states, preventing for example global deadlocks. As third functionality, the CC provides integration for client-libraries based on the mentioned generic models relying on CC-side. Thus, the *developer*, is responsible to write adapter code gluing client-connected components to CC-contained models, solving requirement R4. Additionally, the CC provides registration and connection functions between nodes, resolved by the ROS Core. *Client-connected components*, cover external component, not generated by DiNeROS, and thus may follow other modeling or implementation techniques. They are connected to the overall system based on generic connections models, and a client library. The reason for the existence of these connectors is that not every aspect of cobotics can be modeled equally well with Petri nets. The

proposed architecture is well integrated into ROS, follows the CBSE principals, and thus inherits its advantages, such as reusability and improved development speed handling challenge R5.

## 5 PLAN FOR EVALUATION AND VALIDATION

The basis for evaluating the proposed approach is a multi-robot assembly line, with workflow, world, safety and contextual models represented as DiNeROS Petri nets. More in detail, a chemistry laboratory is simulated, where chemicals are dynamically mixed, analyzed and directly handed over to humans by robots based on recipes, which is dangerous for humans. Additionally, the robot operations are context-dependent, and thus include adaptive behavior when a human enters robot-operated areas. The evaluation will be executed based on four aspects. **Feasibility:** It must be shown that the developed architecture solves the identified challenges R1-R5 and that the approach works with ROS 1 and 2, to support ROS features of both existing versions (R2). **Performance:** This aspect is concerned with evaluating the performance of single components and the overall system. Especially the cobotic coordinator, system generation, node interaction, model transformation and analysis are crucial. **Soundness:** The meaningfulness of the modeling approach and thus which formal properties are guaranteed, by providing a mapping to P/T-nets. **Interaction:** Validation of the envisioned architecture, realized by evaluating the interactions between components and environment.

## 6 EXPECTED CONTRIBUTIONS

The overall research contribution will be the envisioned model-driven architecture for cobotic cells, a framework for the safe development of cobotic applications. The first part of this is the DiNeROS Petri net based modeling approach for ROS-based systems, including a description format. The second part is the DiNeROS development chain, facilitating model-driven development, including the stub-code generation for Petri net based actors. The third part is concerned with the context-adaptivity provided by the generic signal-based interfaces and the adaptation nodes. This includes the integration of hybrid Petri nets as simulation knowledge base. The final part combines all the envisioned parts by providing the cobotic coordinator as central integrator for context-adaption, connections to external components and liveness guarantees.

## 7 CURRENT STATUS

After two years, this PhD thesis is now within the central phase of implementing its main contributions. An overview of relevant literature and approaches is in the final phase of construction, and will be finished within this year. The overall modeling approach, the model-driven development chain and architecture in its requirements and structure is being considered conceptually. The work on a core model-driven development chain will be finished this year, dealing with DiNeROS-nodes. This will be the basis, to be extended by the self-adaptation aspects including an adaptation node framework, the integration of external components and guiding tools such as a DSL and analysis guidance. Once this is completed, the envisioned concepts and components, are brought together within the overall architecture and evaluated based on the presented plans.

## ACKNOWLEDGMENTS

This work was funded by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy – EXC 2050/1 – Project ID 390696704 – Cluster of Excellence “Centre for Tactile Internet with Human-in-the-Loop” (CeTI) of Technische Universität Dresden.

## REFERENCES

- [1] Paulo Sérgio Almeida. 1997. Balloon types: Controlling sharing of state in data types. In *European Conference on Object-Oriented Programming*. Springer, 32–59.
- [2] Gerd Behrmann, Alexandre David, and Kim G Larsen. 2004. A tutorial on uppaal. *Formal methods for the design of real-time systems* (2004), 200–236.
- [3] Debjyoti Bera et al. 2014. *Petri nets for modeling robots*. Ph.D. Dissertation.
- [4] Bernard Berthomieu, F Vernadat, and SD Zilio. 2015. TINA: Time Petri Net Analyzer.
- [5] Betty HC Cheng, Robert Jared Clark, Jonathon Emil Fleck, Michael Austin Langford, and Philip K McKinley. 2020. AC-ROS: assurance case driven adaptation for the robot operating system. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 102–113.
- [6] Stefan-Gabriel Chitic. 2018. *Middleware and programming models for multi-robot systems*. Ph.D. Dissertation. Université de Lyon.
- [7] Philippa Conmy and Iain Bate. 2014. Assuring safety for component based software engineering. In *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*. IEEE, 121–128.
- [8] René David and Hassane Alla. 2010. *Discrete, continuous, and hybrid Petri nets*. Vol. 1. Springer.
- [9] Jörg Desel and Wolfgang Reisig. 1996. Place/transition Petri nets. In *Advanced Course on Petri Nets*. Springer, 122–173.
- [10] Jerry Ding, Jeremy H Gillula, Haomiao Huang, Michael P Vitus, Wei Zhang, and Claire J Tomlin. 2011. Hybrid systems in robotics. *IEEE Robotics & Automation Magazine* 18, 3 (2011), 33–43.
- [11] Christian Dondrup, Ioannis Papaioannou, and Oliver Lemon. 2019. Petri Net Machines for Human-Agent Interaction. *arXiv preprint arXiv:1909.06174* (2019).
- [12] Maksym Figat and Cezary Zielinski. 2020. Robotic System Specification Methodology Based on Hierarchical Petri Nets. *IEEE Access* 8 (2020), 71617–71627.
- [13] Frank HP Fitzek, Shu-Chen Li, Stefanie Speidel, and Thorsten Strufe. 2021. Tactile Internet with Human-in-the-Loop: New frontiers of transdisciplinary research. In *Tactile Internet*. Elsevier, 1–19.
- [14] Thibault Gateau, Caroline P Carvalho Chancel, Mai-Huy Le, and Frédéric Dehais. 2016. Considering human's non-deterministic behavior and his availability state when designing a collaborative human-robots system. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 4391–4397.
- [15] John C Georgas and Richard N Taylor. 2008. Policy-based self-adaptive architectures: a feasibility study in the robotics domain. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*. 105–112.
- [16] Sami Haddadin, Michael Suppa, Stefan Fuchs, Tim Bodenmüller, Alin Albu-Schäffer, and Gerd Hirzinger. 2011. Towards the robotic co-worker. In *Robotics Research*. Springer, 261–282.
- [17] Raju Halder, José Proença, Nuno Macedo, and André Santos. 2017. Formal verification of ROS-based robotic applications using timed-automata. In *2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormalISE)*. IEEE, 44–50.
- [18] Christophe Harel, Bruno Tuffin, and Kishor S Trivedi. 2000. Spnp: Stochastic petri nets. version 6.0. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer, 354–357.
- [19] Robert Hirschfeld, Pascal Costanza, and Oscar Marius Nierstrasz. 2008. Context-oriented programming. *Journal of Object technology* 7, 3 (2008), 125–151.
- [20] Parita Jain, Puneet Kumar Aggarwal, Poorvi Chaudhary, Kshirja Makar, Jaya Mehta, and Riya Garg. 2021. Convergence of IoT and CPS in Robotics. In *Emergence of Cyber Physical System and IoT in Smart Automation and Robotics*. Springer, 15–30.
- [21] Hideharu Kashima and Ryosuke Masuda. 2001. Cooperative control of mobile robots based on petri net. In *The 10th international conference on advanced robotics (ICAR)*. 339–404.
- [22] Sitar Kortik and Tejas Kumar Shastha. 2021. Formal Verification of ROS Based Systems Using a Linear Logic Theorem Prover. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 9368–9374.
- [23] Yehia Kotb, Steven Beauchemin, and John Barron. 2007. Petri Net-Based Cooperation In Multi-Agent Systems. 123–130.
- [24] Charles Lesire and Franck Pommereau. 2018. ASPiC: an Acting system based on Skill Petri net Composition. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 6952–6958.
- [25] Joaquín López, Alejandro Santana-Alonso, and Miguel Diaz-Cacho Medina. 2019. Formal verification for task description languages. a petri net approach. *Sensors* 19, 22 (2019), 4965.
- [26] Matt Luckcuck, Marie Farrell, Louise A Dennis, Clare Dixon, and Michael Fisher. 2019. Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys (CSUR)* 52, 5 (2019), 1–41.
- [27] Robyn R Lutz. 1993. Analyzing software requirements errors in safety-critical, embedded systems. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*. IEEE, 126–133.
- [28] Filipe Moutinho. 2014. *Petri net based development of globally-asynchronous locally-synchronous distributed embedded systems*. Ph.D. Dissertation. Faculdade de Ciências e Tecnologia (FCT).
- [29] Henry Muccini, Mohammad Sharaf, and Danny Weyns. 2016. Self-adaptation for cyber-physical systems: a systematic literature review. In *Proceedings of the 11th international symposium on software engineering for adaptive and self-managing systems*. 75–81.
- [30] Torre Norte. 2002. Petri Net Views of a Robotic Task. *International Conference on Robotics and Automation* (2002), 0–5.
- [31] James L Peterson et al. 1980. A note on colored Petri nets. *Inf. Process. Lett.* 11, 1 (1980), 40–43.
- [32] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. Kobe, Japan, 5.
- [33] Olivier Henri Roux and Anne-Marie Déplanche. 2002. A t-time Petri net extension for real time-task scheduling modeling. *European Journal of Automation* 36, 7 (2002), 973–987.
- [34] John Rushby. 2008. How Do We Certify For The Unexpected?. In *AIAA Guidance, Navigation and Control Conference and Exhibit*. 6799.
- [35] Guido Salvaneschi, Carlo Ghezzi, and Matteo Pradella. 2012. Context-oriented programming: A software engineering perspective. *Journal of Systems and Software* 85, 8 (2012), 1801–1817.
- [36] Rui Wang, Yong Guan, Houbing Song, Xinxin Li, Xiaojuan Li, Zhiping Shi, and Xiaoyu Song. 2018. A formal model-based design method for robotic systems. *IEEE Systems Journal* 13, 1 (2018), 1096–1107.
- [37] N. Yakymets, S. Dhouib, H. Jaber, and A. Lanusse. 2013. Model-driven safety assessment of robotic systems. *IEEE International Conference on Intelligent Robots and Systems* (2013), 1137–1142.
- [38] Stefan Zander, Georg Heppner, Georg Neugschwandtner, Ramez Awad, Marc Essinger, and Nadia Ahmed. 2016. A model-driven engineering approach for ros using ontological semantics. *arXiv preprint arXiv:1601.03998* (2016).