Cross–layer Formal Verification of Robotic Systems

Sylvain Raïs, Julien Brunel, David Doose, Frédéric Herbreteau

Cross-Layer Formal Verification of Robotic Systems*

Sylvain Raïs[1,2], Julien Brunel[1], David Doose[1] and Frédéric Herbreteau[2]
[1] ONERA DTIS, Université de Toulouse, France
[2] Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, 33400, Talence, France
[1] firstname.lastname@onera.fr, [2] firstname.lastname@u-bordeaux.fr

Robotic systems are widely used to interact with humans or to perform critical tasks. As a result, it is imperative to provide guarantees about their behavior. Due to the modularity and complexity of robotic systems, their design and verification are often divided into several layers. However, some system properties can only be investigated by considering multiple layers simultaneously. We propose a cross-layer verification method to verify the expected properties of concrete robotic systems. Our method verifies one layer using abstractions of other layers. We propose two approaches: refining the models of the abstract layers and refining the property under verification. A combination of these two approaches seems to be the most promising to ensure model genericity and to avoid the state-space explosion problem.

1 Introduction

The design and development of modern robotic systems is a complex issue, as it brings together many fields of research. Moreover, these robotic systems are intended to interact with humans or to be deployed in critical sites. Therefore, it is essential to provide guarantees for the operation of these systems. Formal methods are widely used to assert the reliability of critical systems. They provide strong proof-based guarantees that the verified system behaves accordingly to the specifications. In the context of robotic systems, several modeling tools and formalisms have been developed to verify properties, either online [7] or offline [4, 6].

On the other hand, in order to improve the design of robotic systems, state-of-the-art approaches rely on multi-layer architectures as they provide powerful abstraction to develop each layer independently of the others. Such a design facilitates the development of robotic systems, improves their modularity and enables each layer to be (formally) verified separately. These advantages help to implement complex behaviors such as fault tolerance [9] and facilitate the reuse of robotic system code. Note that several multi-layer design standards exist within the robotics research community: five-layer pyramid design [3], four-layer design [13], three-layer pyramid design [10, 14], and more. Among these classical designs, the three-layer architecture shown in Figure 1 is a promising and widely used approach because it provides a modular design while minimizing the number of layers. In this architecture, the decision layer deals with the robot's decision-making and planning processes (e.g. a user interface or a "smart" program). The executive layer provides an abstract interface to the functional layer via the concept of skills [1, 13, 14, 10]. And the functional layer corresponds to low-level task processing.

*This study has received financial support from the French government in the framework of the France 2030 programme IdEx université de Bordeaux / RRI ROBSYS

Matt Luckcuck and Mengwei Xu (Eds.):
Sixth International Workshop on Formal Methods
for Autonomous Systems (FMAS 2024)
EPTCS 411, 2024, pp. 143–150, doi:10.4204/EPTCS.411.9

Listing 1: An example of RobotLanguage design

```
skillset custom_robot {
    resource {
        motion { state { On Off } initial Off transition all }
        battery { state { Normal Critical } initial Normal transition all }
    }
    skill goto {
        input {distance: Integer}
        output position: Position
        precondition { (motion == Off) && (battery != Critical) }
        start motion -> On
        invariant {in_movement { guard motion == On }}
        interrupt {effect {motion -> Off}}
        success {arrived {effect {motion -> Off}}}
        failure {blocked {effect {motion -> Off}}}
    }
}
```

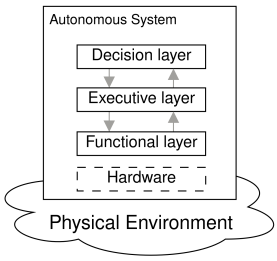


Figure 1: Three-layer architecture

In practice, it is impossible to verify the whole system at once, due to the complexity of robotic systems, or the incompatibility of certain theories that make verification undecidable. Specific formalisms and techniques have been developed for the design and the verification of each layer separately. However, the compartmentalization of the different analyses is an obstacle to complete system analysis because these formalisms cannot always be combined. In fact, some of the operating characteristics of the system must consider multiple layers in order to be studied.

The present work, which is part of a Ph.D. thesis, aims to provide an offline cross-layer verification method based on the three-layer design in Figure 1. Our method uses RobotLanguage [1] [5, 2, 1], an interesting framework for designing reliable robotic systems. RobotLanguage provides a formal language to model the executive layer, a formal offline verification of predefined properties on this model, and an automatic code generation from the model to implement this layer. Our approach is based on a RobotLanguage model of the executive layer, and extends it with abstract models of the other layers in order to verify properties of the whole system. In general, abstract models are not refined enough to verify robotic systems. We introduce two complementary approaches: one consists in refining the model, and the other consists in refining the property. We illustrate the relevance of our techniques on an example. Our method is not specific to the three-layer architecture in Figure 1, and can be used with other multi-layer designs.

2 Related Works

Several formal frameworks have been defined to support the design and the verification of robotic systems, such as RobotLanguage. PROSKILL [8] gathers the specifications of the decisional layer, the executive layer and a part of the functional layer (see Figure 1), and allows to verify temporal and timed properties both offline and online. However, PROSKILL provides a monolithic design for robotic systems and does not benefit from the advantages of a multi-layer design. Our method is based on the multi-layer design, preserving the modularity gained by this design, and thus fits well to our real robotic systems.

On the other hand, RobotLanguage comes with a tool, SkiNet [11], which provides a translation to Petri nets to perform offline formal verification of temporal properties. This tool has also been extended [12] to verify temporal properties online in order to address the state-explosion problem. However, SkiNet only verifies properties of the executive layer only, while our work aims to provide a multi-layer verification method.

[1] https://onera-robot-skills.gitlab.io/index.html

3 Cross-Layer Verification

RobotLanguage has been developed to design the executive layer of robotic systems. After a brief introduction, we describe the formalism used to model these systems. Next, we explain how to model each system layer and how to incorporate all models for formal verification. Finally, we present a method for systematically verifying multi-layer systems, illustrated with an example.

3.1 Introduction to RobotLanguage

Modern approaches to formal robotic system design are based on skills and resources[8, 5, 13, 10]. Skills are basic actions provided by the executive layer to implement complex behaviors in the decision layer. For example, Listing1 defines one skill: `goto`, which moves a robot a given distance. Resources represent physical features used by skills, such as `motion` and `battery` in Listing 1. The resource `battery` tracks levels, while `motion` monitors movement. In RobotLanguage, each group of skills and their shared resources forms a *skill set*, such as `custom_robot` in Listing 1.

The skill `goto` is an abstraction of the actual code executed at the functional layer. In RobotLanguage the system designer specifies conditions for starting a skill (`precondition`), conditions that should remain true during execution (`invariant`), and resource updates (`start`, `effect`).

RobotLanguage includes a toolset[2] that translates models into executable C++ code using the ROS2 middleware. This code creates one ROS2 node per skill set and several topics to manage communication between the executive and decision layers. In addition, the generated code verifies conditions (`precondition`, `invariant`) and applies effects (`start`, `effect`) specified in RobotLanguage. The programmer is responsible for implementing the functional layer in specific hook functions, whose prototypes are generated from the RobotLanguage design.

3.2 Modeling Formalism

In this paper, a model consists of a finite set $M = \{S_1, \ldots, S_k\}$ of finite labeled transition systems. Each transition system $S_i = (Q_i, q_i^0, \Sigma_i, T_i)$ consists of a finite set of states Q_i, a distinguished initial state q_i^0, a finite alphabet of events Σ_i and a transition relation $T_i \subseteq Q_i \times \Sigma_i \times Q_i$ where edges are labeled by events from Σ_i. Note that the transition systems may have common events on which they synchronize. Let $\Sigma = \bigcup_{i\in[1;k]} \Sigma_i$. A global state of M is a tuple $(q_1, \ldots, q_k)$ of states, one for each transition system in M. The initial global state is $(q_1^0, \ldots, q_k^0)$. There exists a global transition $(q_1, \ldots, q_k) \xrightarrow{a} (q_1', \ldots, q_k)$ with $a \in \Sigma$ if for each S_i such that $a \in \Sigma_i$, there exists a transition $(q_i, a, q_i') \in T_i$, and $q_i' = q_i$ for every S_i such that $a \notin \Sigma_i$. A global run is a sequence of global transitions starting from the initial global state.

As an example, consider the model consisting of two transition systems: S in Figure 2 and F in Figure 3a. These two transition systems synchronize on their common labels. Thus, any run in this model consists of asynchronous solid and zigzag transitions from S, or dotted and dashed transitions that synchronize S and F.

3.3 Executive Layer Modeling

First, we explain how to model the executive layer by describing the execution of a skill through the transition system in Figure 2. During its execution, the skill transitions through several states, depending on internal actions (plain transitions), or on interactions with the decision layer (zigzag transitions) or

[2] `https://onera-robot-skills.gitlab.io/`

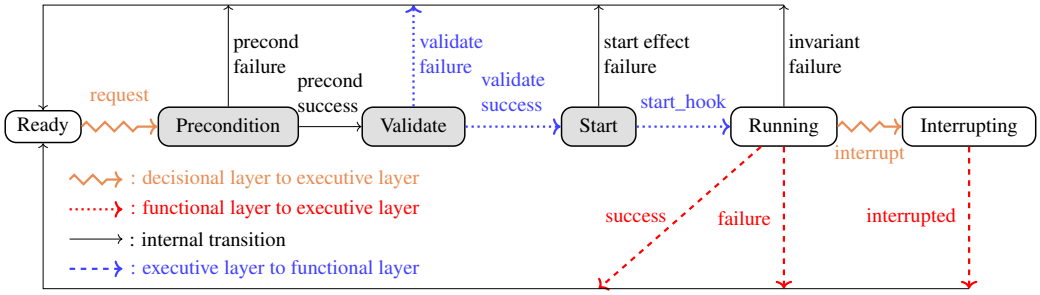


Figure 2: Control flow graph of a skill

with the functional layer (dashed/dotted transitions). The execution begins in the state `Ready`, and spans into three phases. First, on reception of `request` from the decision layer, the state of the system is checked at the executive layer (`precond`) and the functional layer (`validate`). If these conditions are satisfied, `start_hook` triggers the execution of the functional layer, switching the state to `Running`. Finally, the execution can terminate in a success or a failure, triggered by the functional layer, or it can be interrupted by the decision layer. In each case, the functional layer notifies the executive layer by calling `success`, `failure`, or `interrupted`. The execution can also stop if an invariant is violated. These invariants are monitored by the code that is automatically generated from the RobotLanguage design. We refer the reader to [1] for more details on the semantics of invariants in RobotLanguage which is beyond the scope of this paper.

For a given skill set like in Listing 1, a model using instances of the transition system in Figure 2 for each skill can formally verify some properties at the executive layer, such as "skill `goto` can be executed." However, this model is not refined enough to verify more specific properties, such as "skill `goto` cannot be executed infinitely often", which is expected to hold, since our RobotLanguage design in Listing 1 lacks a skill to recharge the battery.

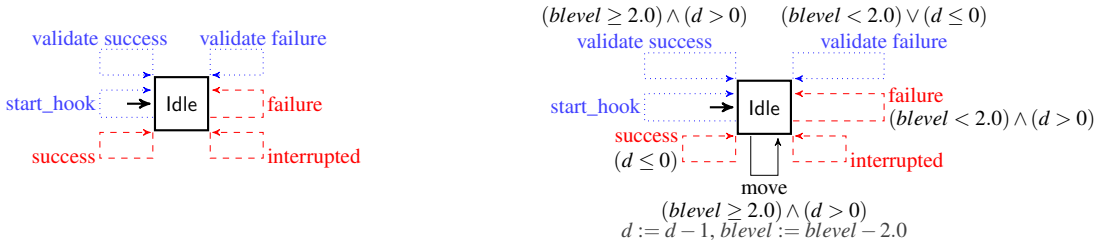
3.4 Multi-Layer Modeling

We aim to extend the model in Figure 2 (called S in the sequel) with models of the functional and decision layers. For now on, we will concentrate on the functional layer, as the approach that we present hereafter straightforwardly applies to the decision layer.

From Section 3.2, it comes that a model of the functional layer should conform to a *synchronization interface* that will enable communication between the model of the functional layer, and the model of the executive layer, through event synchronizations. More specifically, a model of the functional layer should synchronize on events validate success, validate failure, start hook, success, failure and interrupted with the transition in Figure 2.

The transition system F in Figure 3a shows a very abstract model of a functional layer that conforms to this synchronization interface. Note that F allows any sequence of the above mentioned events since its transitions can be crossed unconditionally. Hence, any sequence of events that is possible in S is also possible in the model $\{S,F\}$ where S and F synchronize on common labels. F can be seen as the generic most abstract functional layer model.

Figure 3b shows another model of the skill `goto` at the functional layer. Observe that this model also conforms to the synchronization interface. It further has an internal action `move` that does not synchronize with S: it is asynchronous. This model is described as a control graph with two variables: d which is the distance to travel, and *blevel* which is the battery level (both variables have finite domains). Note



(a) Default functional layer model

(b) Concrete functional layer model for skill `goto`

Figure 3: Transition systems modeling the skill `goto` at the functional layer

that the variables used at the functional layer partly model the robot's state, while the RobotLanguage resources used in the executive layer (Listing 1) are abstract knowledge of the robot's state, updated by monitoring the robot. The model for updating the `battery` resource according to the actual value of *blevel* is not shown for the sake of simplicity. Following our settings described in Section 3.2, we consider the transition systems F' that defines the semantics of control graph in Figure 3b. Its states are pairs $(d, blevel)$ of values of the two variables, and transitions $(d, blevel) \xrightarrow{a} (d', blevel')$ take into account the guards and updates on the variables. Now, observe that due to variables d and *blevel*, the model F' restricts the sequences of events that can occur in a run. For instance, validate success is not possible if the battery level is less than 2.0. F' can thus be seen as a refinement of F. As a result, some runs that exist in S do not exist any more in the model $\{S, F'\}$ that synchronizes S and F'.

Similarly, we can model the decision layer, with a synchronization interface that is defined by the events request and interrupt. We thus obtain a multi-layer model, that consists in transition systems for each skill (as in Figure 2) at the executive layer, for each skill at the functional layer (as in Figure 3a or 3b), as well as transition systems for each resource (as defined in Listing 1) and a transition system modeling the decision layer.

3.5 A Method for Cross-layer Verification

We aim at verifying that "skill `goto` cannot be executed infinitely often" taking into account a model of the functional layer. This property can be expressed in LTL as:

$$\mathsf{F}\,\mathsf{G}\ \texttt{not Running} \tag{1}$$

This formula specifies that after some finite amount of time, the robot will never be running. It does not hold when the functional layer is modeled as in Figure 3a. We present two approaches for verifying such specifications requiring a multi-layer model.

A first approach consists in considering the refined model of the functional layer in Figure 3b, where d represents the distance to travel, and *blevel* tracks the battery level. In Figure 3b, the black loop moves the robot one meter ahead, consuming two battery units at the same time. At some point, either the distance d reaches 0 which leads to a `success`, or the battery level gets below 2.0 which leads to a `failure`. Observe that the battery level *blevel* is set at the initialization of the model. Hence, the battery level eventually becomes insufficient to execute skill `goto`: it only allows "`validate failure`" and "`failure`" transitions. As a result, property (1), that is "skill `goto` cannot execute infinitely often", holds on the refined model.

A second approach consists in refining the specification. In this approach, we aim at verifying our property: "skill `goto` cannot execute infinitely often" on the model including the abstract representation

of the functional layer from Figure 3a, but *with some extra assumptions*. Coming back to our example, since our RobotLanguage design in Listing 1 does not include a skill to recharge the battery, we can expect the resource `battery` to be in state `Critical` after some finite amount of time. Hence, we can verify that "skill `goto` cannot execute infinitely often" *under the assumption* "eventually the `battery` is forever in state `Critical`". This approach consists in refining the LTL formula in (1) to verify the property only on runs which satisfy this assumption. This is formalized in (2), where `Critical` corresponds to the state of the resource `battery` in Listing 1. This formula ensures that if the `battery` eventually stays in state `Critical` forever, then, the skill `goto` is not executed infinitely often.

$$\mathsf{F}\,\mathsf{G}\,\texttt{Critical} \implies \mathsf{F}\,\mathsf{G}\,\text{not}\,\texttt{Running} \tag{2}$$

Observe that due to the precondition in Listing 1 the transition labeled "`precond success`" in Figure 2 can only be taken a finite number of times on any run such that the `battery` eventually stays in state `Critical` forever. As a result the property in (2) holds on the abstract model of the system with the functional layer modeled by the transition system in Figure 3a. Observe that this model does not need any extra variable and is thus much smaller than the model obtained with the first approach.

To validate our approaches, we have translated the transition systems and specifications corresponding to the two approaches, as formulas for the *Tatam* model-checker[3]. The RobotLanguage design in Listing 1 as well as the Tatam models underlying the two approaches above are available on a public repository[4]. As expected, we have first observed that the property "skill `goto` cannot execute infinitely often" does not hold on the abstract model of the functional layer in Figure 3a as the discharge of the battery is not taken into account. On the other hand, the two approaches above allow to prove that the property holds, either by providing a refined model of the functional layer, or by refining the specification.

We see these two approaches as complementary tools for cross-layer verification of robotic systems. Refining the property keeps the model small and simple. It also yields a simpler counter-example when a property is not satisfied. However, some properties require a more precise knowledge of the state of the robot. Then, the first approach should be used to refine (parts of) the model with as few details as possible in order to be able to verify the property under consideration.

4 Conclusion

This paper presents a method for cross-layer verification of robotic systems. Our approach consists in verifying one layer using abstractions of the others. We have proposed two approaches to prove a property. One consists in refining the models of the abstract layers, the other consists in refining the property. In practice, the combination of the two approaches seems to be the most promising since it allows to consider as few implementation details as possible in the model, while mitigating the state-space explosion problem.

As future work, we plan to implement our approach in a tool to formally verify RobotLanguage designs using a precise model of the executive layer and abstract models of the decision and functional layers. To obtain a full guarantee approach, we plan to extend our technique to prove that these abstract models correspond to the implementation of the corresponding layers.

[3]Tatam git repository: `https://github.com/DavidD12/tatam`

[4]`https://gitlab.com/sylvain.rais24/fmas_2024_s_rais_models`

References

[1] Alexandre Albore, David Doose, Christophe Grand, Jérémie Guiochet, Charles Lesire & Augustin Manecy (2023): *Skill-based design of dependable robotic architectures* 160, p. 104318. doi:10.1016/J.ROBOT.2022.104318.

[2] Alexandre Albore, David Doose, Christophe Grand, Charles Lesire & Augustin Manecy (2021): *Skill-Based Architecture Development for Online Mission Reconfiguration and Failure Management*, pp. 47–54. doi:10.1109/ROSE52553.2021.00015.

[3] V. Alcácer & V. Cruz-Machado (2019): *Scanning the Industry 4.0: A Literature Review on Technologies for Manufacturing Systems*. *Engineering Science and Technology, an International Journal* 22(3), pp. 899–919, doi:10.1016/j.jestch.2019.01.006. Available at `https://www.sciencedirect.com/science/article/pii/S2215098618317750`.

[4] Renato Carvalho, Alcino Cunha, Nuno Macedo & André Santos (2020): *Verification of system-wide safety properties of ROS applications*. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*, IEEE, pp. 7249–7254, doi:10.1109/IROS45743.2020.9341085.

[5] Christophe Grand Charles Lesire, David Doose (2020): *Formalization of Robot Skills with Descriptive and Operational Models*, pp. 7227–7232. doi:10.1109/IROS45743.2020.9340698.

[6] Lukas Johannes Dust, Rong Gu, Cristina Seceleanu, Mikael Ekström & Saad Mubeen (2023): *Pattern-Based Verification of ROS 2 Nodes Using UPPAAL*. In Alessandro Cimatti & Laura Titolo, editors: *Formal Methods for Industrial Critical Systems - 28th International Conference, FMICS 2023, Antwerp, Belgium, September 20-22, 2023, Proceedings*, 14290, Springer, pp. 57–75, doi:10.1007/978-3-031-43681-9_4.

[7] Jeff Huang, Cansu Erdogan, Yi Zhang, Brandon M. Moore, Qingzhou Luo, Aravind Sundaresan & Grigore Rosu (2014): *ROSRV: Runtime Verification for Robots*. In Borzoo Bonakdarpour & Scott A. Smolka, editors: *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, 8734, Springer, pp. 247–254, doi:10.1007/978-3-319-11164-3_20.

[8] Félix Ingrand (2024): *PROSKILL: A formal skill language for acting in robotics*. *CoRR* abs/2403.07770, doi:10.48550/ARXIV.2403.07770.

[9] André Leite, Andry Maykol Pinto & Aníbal Matos (2018): *A Safety Monitoring Model for a Faulty Mobile Robot*. *Robotics* 7(3), p. 32, doi:10.3390/ROBOTICS7030032.

[10] Mikkel Rath Pedersen, Lazaros Nalpantidis, Rasmus Skovgaard Andersen, Casper Schou, Simon Bøgh, Volker Krüger & Ole Madsen (2016): *Robot skills for manufacturing: From concept to industrial deployment*. *Robotics and Computer-Integrated Manufacturing* 37, pp. 282–291, doi:10.1016/j.rcim.2015.04.002.

[11] Baptiste Pelletier, Charles Lesire, David Doose, Karen Godary-Dejean & Charles Dramé-Maigné (2022): *SkiNet, A Petri Net Generation Tool for the Verification of Skillset-based Autonomous Systems*. In Matt Luckcuck & Marie Farrell, editors: *Proceedings Fourth International Workshop on Formal Methods for Autonomous Systems (FMAS) and Fourth International Workshop on Automated and verifiable Software sYstem DEvelopment (ASYDE), FMAS/ASYDE@SEFM 2022, and Fourth International Workshop on Automated and verifiable Software sYstem DEvelopment (ASYDE)Berlin, Germany, 26th and 27th of September 2022*, *EPTCS* 371, pp. 120–138, doi:10.4204/EPTCS.371.9.

[12] Baptiste Pelletier, Charles Lesire, Christophe Grand, David Doose & Mathieu Rognant (2023): *Predictive Runtime Verification of Skill-based Robotic Systems using Petri Nets*. In: *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, IEEE, pp. 10580–10586, doi:10.1109/ICRA48891.2023.10160434.

[13] Francesco Rovida, Matthew Crosby, Dirk Holz, Athanasios Polydoros, Bjarne Großmann, Ronald Petrick & Volker Krueger (2017): *SkiROS—A skill-based robot control platform on top of ROS*, pp. 121–160. doi:10.1007/978-3-319-54927-9_4.

[14] Casper Schou, Rasmus Skovgaard Andersen, Dimitrios Chrysostomou, Simon Bøgh & Ole Madsen (2018): *Skill-based instruction of collaborative robots in industrial settings*. Robotics and Computer-Integrated Manufacturing 53, pp. 72–80, doi:10.1016/j.rcim.2018.03.008. Available at https://www.sciencedirect.com/science/article/pii/S0736584516301910.