Predictive Runtime Verification of Skill-based Robotic Systems using Petri Nets

Baptiste Pelletier, Charles Lesire, Christophe Grand, David Doose, Mathieu Rognant

▶ To cite this version:

Baptiste Pelletier, Charles Lesire, Christophe Grand, David Doose, Mathieu Rognant. Predictive Runtime Verification of Skill-based Robotic Systems using Petri Nets. 2023 IEEE International Conference on Robotics and Automation (ICRA), May 2023, Londres, United Kingdom. pp.10580-10586, 10.1109/ICRA48891.2023.10160434 . hal-04168590

HAL Id: hal-04168590
https://hal.science/hal-04168590v1

Submitted on 21 Jul 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Predictive Runtime Verification of Skill-based Robotic Systems using Petri Nets

Baptiste Pelletier and Charles Lesire and Christophe Grand and David Doose and Mathieu Rognant

Abstract— **This work presents a novel approach for the online supervision of robotic systems assembled from multiple complex components with skillset-based architectures, using Petri nets (PN). Predictive runtime verification is performed, which warns the system user about actions that would lead to the violation of safety specifications, using online model-checking tools on the system PNs.**

I. INTRODUCTION

The development of system composed of multiple complex components comes with two difficulties: designing elements which are modular and have independently safe behaviors, and ensuring that their interaction once assembled will respect safety specfications during operations. While the former can be relatively easy to do, many problems can arise in the latter that would normally not be considered if components were to stay as single-working entities. For instance, mounting a robotic arm on a mobile robot will yield new challenges with regards to positionning or collision-avoidance, so a safety specification would be to only allow long distance movements of the mobile robot if the arm is in a safe position. This issue of verification complexity becomes even more important in the context of semi-autonomous systems. Human operated phases require the operator to focus on a lot of factors: individual robot health and system status, coordination of movements/actions, monitoring of safety properties, and so on. The use of a high-level of abstraction for the components can make their development, assembly and use more approachable. Their formal verification also becomes more accessible, which helps inspect reachability and/or deadlock properties of the system. For instance, complex components can be modelled as discrete-event systems such as finite state-machines (FSMs), but this comes with scalability and controllability issues, which have already been present in the early literature, as seen in the work of Ramadge and Wonham [1]. Furthermore, the state-space explosion leading to the assembly of these complex components makes a-priori verification impossible, or dangerous states are unavoidable by construction. This means the verification needs to be made during operations, by analyzing the state of the system to detect the satisfaction or violation of properties, and reacting accordingly. This method, called Runtime Verification (RV), can be made on-board the system or on a remote platform [2], [3], [4]. On top of RV, model-checking (MC) can be performed if a model of the system has been established, to better predict and react to specifications violation [5], [6], [7], which can be seen in Runtime Assurance techniques (RTA) [8], [9]. In this work, we will perform on-the-fly MC for predictive RV, using Petri net (PN) models of the components.

This research was supported by the Occitanie region, France.
ONERA/DTIS, University of Toulouse, France
firstname.lastname@onera.fr

Prior to this work, we developed a high-level formalization with a skill-based architecture, based on the work of Albore et al. [10], to model the components of the system as a set of finite-state machines (FSMs). Their respective executions are modelled using PNs, as they are well suited for modelling compositions of concurrent execution of FSMs. Formal verification of each component was done a-priori, offline, using MC on the PN model, which helped raise initial design concern [11]. In this paper, we propose an online monitoring system which observes the behavior of each component, checks the satisfaction of specifications set by the user, and predicts actions that would inevitably violate them. The monitoring system performs MC on the PN models to calculate the future possible paths of the system, based on the combination of the PNs of each component in their current states, and predict future specification violation, on a possibly finite-horizon, depending on system scale.

The present work showcases the monitoring tool on a semi-autonomous Spot® quadruped robot with a Kinova Jaco manipulator arm, shown in Fig. 1, that deactivates traps while being remotely operated by a human. Section II will go over the related work on RV, supervision and control of robotic systems using PN, before moving on to the relevant definitions of skillsets and skillset PN in Section III. Section IV will showcase the SkiNet Live tool and its internal mechanisms, and Section V will present how our tool helped the user while operating the system. Finally, Section VI will conclude and open the discussion. Readers can try out a simple version of the tool at: https://gitlab.com/onera-robot-skills/skinet-release.

Fig. 1. Spot® quadruped robot equipped with a Kinova Jaco manipulator arm and a mounted computer executing controller code. The system has to deactivate traps, represented as wooden cubes with a panel that needs to be pushed to neutralize the trap.

II. RELATED WORKS

To ensure a system complies with specifications during its execution, two methods are usually employed: either creating a safe controller which will limit the execution of the system [1], [12], [13], called controller synthesis, or monitoring its behavior during runtime and reacting to specification violation [2], [3], [4], called runtime verification (RV). We will focus on the latter, as model-checking on individual components was already performed in a previous work [11], and performing controller synthesis and a-priori verification on complex systems has limitations with increasing scale [1].

A. Runtime Verification

The goal of RV is to check whether the current execution of a system respects specifications, by only looking at the current state and/or the sequence of actions that led to it. It was first developped for software monitoring, such as with the work of Havelund [2], where RV was used to control complex systems by only looking at the current behavior and implementing predictive algorithms to avoid violating specifications in the future. In the context of robotics, Lotz [3] performs RV by monitoring the components of a robotic system, where each component is a black-box whose behavior can only be deducted from partial observation of its state, making the prediction of future events more challenging. Blech [4] pushes this further by studying the use of RV in safety critical systems, and how much a RV system can be trusted, offering a way to certify RV monitors. In the context of the middleware ROS (robot operating system), Huang [14] proposes ROSRV, a tool that monitors exchanged ROS messages to perform RV, with no intrusion in the original system. The work of Foughali [12], [13] tackled the issues of correctness and scalability of RV for concurrent robots, using formal methods and a custom framework called GenoM3. Finally, Colledanchise [15] showcases the use of behavior trees (BT) for RV of deliberative policies, building runtime monitors using a formal property specification language. These approaches can be combined with MC, so that the monitored components are no longer black-boxes, but well defined and formally verified models, such as the skill-based architectures from Albore [10] we use in this work. Desai [7], [8] combines offline a-priori MC for RV and uses signal temporal logic (STL) for online monitoring to construct safe motion plans with the SOTER framework. The authors mention that this method can have scalability issues during offline calculations, but can perform predictions on infinite horizons. Our approach will perform MC online, during runtime, but at the cost of a possibly finite horizon for large systems. This allows the users more flexibility with the system, as there is no need to perform offline calculations everytime the safety specifications are changed. As for scalability, because our approach deals with high-level components that are already an abstraction of their respective system, we believe the use cases for the tool would rarely exceed a few components.

B. Predictive RV

Coupling RV with MC improves analysis, prediction and reaction capabilities by using MC tools on a complete or partial model of the system execution. This was proposed by Leucker [5], using Linear Temporal Logic (LTL) specifications, combining the verification of an abstract model of the software program and the verification of its current run, using LTL_3, introduced by Bauer [16]. In the same manner, Pinisetty [6] uses a-priori knowledge of the system model to perform predictive RV of timed properties, using timed automata.

The approaches combining MC and RV are the most interesting for the challenge at hand: providing guidance to a user that is potentially unfamiliar with the complex system they are using, by interrogating models of the components to avoid the violation of specifications in the future. Approaches like Leucker and Pinisetty have not yet been applied to robotic systems, to the best of the authors knowledge. The goal of our tool is to fill this need for predictive RV for robotics, where the need for a reactive system is important, both for operator guidance and system autonomy. Because we model the components execution as PNs, we will use them as the interrogated models.

C. Petri nets for Control and Runtime verification

PNs have been used for the monitoring, supervision and control of DES, as well as RV. The early work of Giua [17] illustrates how PN can be used for the control of discrete-events system purely mathematically, with linear inequalities to define the reachable markings of the net. The application to manufacturing systems becomes clear with the work of Der Jeng [18] by using control places and control transitions on top of the original net of the system to restrict its behavior, without the need for exhaustive state-space enumeration. PNs are also used for controller synthesis [19], [20], [21], [22], even if the model is only partially controllable or observable, such as in the work of Luo et al. [23], [24], [25]. Offline controller synthesis using model checking (MC) was studied by Rezig [26], as well as Lacerda [27] who used tools from the Tina toolbox [28], [29] and LTL specifications for controller synthesis of multi-robot teams, without the need for state-space enumeration. Finally, the approach of Lee [30], [31] tackles the human-in-the-loop problem, where a human operator can operate the system and make it go, purposefully or not, into a faulty state. Ding [32] continues on this work and uses place-invariant-control theory to design a controller that will find not only dangerous markings, but also markings that would inevitably lead to them after a finite number of transitions, using Computation Tree Logic (CTL). However, such approaches limit the controllability of the system, whereas in our context, the human operator should have full control of the system at all times, regardless of specification violation. This is important in the context of a dynamic, dangerous environment, where for instance the walking robot needs to be quickly moved to avoid a danger that would damage the whole system, without having to wait for other components to be in a configuration

that would respect the specification. Ramirez-Trevino [33] suggests the use of two PNs: a healthy one and a faulty one, to design an online diagnoser, where both PNs are compared during runtime to check for faulty behaviors in the system. We inspire from this later on by using two PNs, one for monitoring, and one for MC. Our approach performs the MC of the PNs, using tools from Tina, as introduced by Lacerda [27], for the state-space generation, but online, during runtime. We also base our work on Ding [32] where CTL is used to find dangerous markings, i.e. operations that would inevitably lead to property violation if performed.

III. Background

A. Skillset

This section summarizes the elements of a Skillset used for the modelling and programming of autonomous systems, as defined by Albore [10], with some elements omitted as they are not used in this paper. A Skillset can represent both hardware and software elements of the system, and their interaction/execution. A skillset contains resources, resource guards and resource effects. These elements are assembled to create events and skills. The example system used in the experimental work of this paper contains two skillsets: a `spot` skillset, used on the Boston Dynamics Spot® quadruped robot, and a `manipulator` skillset, used on the Kinova Jaco manipulator arm, which was mounted on Spot®. It is important to note that the skillsets of both components were made independantly for other applications, so their assembly must be done without modifying these skillsets. Finally, a computer is also mounted, used to execute the controller code of both components. Figure 1 shows the assembled system as used for experimental validation. Skillset files are available in the GitLab repository: https://gitlab.com/onera-robot-skills/skinet-release.

A tool called "robot language" generates C++ code based on the written skillset specifications that follows this execution [10], with part of the execution code to be filled by the user, such as skills functions, exit conditions, events triggering, etc. A ROS2 [34] skillset manager node is then deployed as the controller, and our tool monitors the messages of this manager, as shown later in Fig. 2. More information on the skillset execution semantic can be found in [10].

B. Skillset Petri net

This section will sum up the definitions and notions of skillset Petri nets [11]. Skillset Petri nets are conventional Petri nets, with the addition of a transition function to set a priority order between transitions. Skillset Petri nets have places representing resources and skills states, and transitions that move tokens between places as the execution of the skillset goes. A Skillset Petri net $SkN = \langle N, m_0 \rangle$ is a tuple $N = (P, T, F, \succ)$ and an initial marking m_0, where:

- P and T are two non-empty, disjoint and finite sets of places and transitions, respectively.
- P is the set of places, with one place for each resource state, and each skill $s \in S$ is represented by an idle place e_s, a running place i_s, and exit places $x_{s,k}$, representing the various termination modes (success, failure...). At most one token is present in each place, and one token is shared at all times between each place for a given skill or resource.
- T is the set of transitions, composed of three subsets T_{events}, T_{skills} and T_{reset}. T_{events} is the set of transitions associated with events, firing when events occur during execution and their resource guard is satisfied, applying their resources effects as an exchange of tokens between resources places. T_{skills} is the set of transitions associated with skills. When a skill s successfully starts, the transition $t_{s,start}$ is fired, and upon termination, the associated exit transition $t_{s,k}$ is fired. Tokens are moved depending on the various resources guards and effects of the skill execution elements they are associated to. Finally, T_{reset} transfers the tokens from the exit places $x_{s,k}$ to the idle place e_s, so that the skill can start again.
- $F \subseteq (P \times T) \cup (T \times P)$ a set of directed, unitary arcs between places and transitions.
- $\succ$ is the priority relation, meaning that if two transitions $t1, t2 \in T$ are enabled, i.e. their input places have at least one token, and $t1 \succ t2$, then only $t1$ is firable. Here, the transitions $T_{invariants} \in T_{skills}$ have more priority than all the other transitions $T - T_{invariants}$, because invariant failures cause an immediate stop in skill execution before anything else can happen.
- $m_0 \in M$ the initial distribution of tokens, called the initial marking of the net, and $M = \{m_0, m_1, ..., m_n\}$ the set of all possible markings of N.

For any transition $t \in T$, the sets of its input and output nodes are ${}^\bullet t$ and $t^\bullet$ respectively. Let $p \in P$ and $t \in T$ be a place and a transition. The marking of a place p is noted $m[p]$. The firing of an enabled transition $t \in T$, with $\forall p \in^\bullet t,\ m[p] \geq 1$, leads to a new marking, or reachable state, m'. All the input places of t loose a token, i.e. $\forall p \in^\bullet t,\ m'[p] = m[p] - 1$, and all the output places gain one token, i.e. $\forall p \in t^\bullet,\ m'[p] = m[p] + 1$.

IV. SkiNet Live Tool

A. Overview of the tool

In this paper, we present the SkiNet Live tool, Fig. 2, which performs predictive runtime verification of user-defined temporal logic specifications, using model-checking tools from Tina [29], [28] and Petri net models of the system components [11]. The controller code of the skillset managers is generated from their written specifications [10], with SkiNet Live monitoring the messages sent by each manager using ROS2 [34], as well as communicating with the human operator. The operator can observe the state of each skillset nets and transition firing history through ROS2 topics, as well as the state-space exploration results. The tool runs two processes: a fast process that runs direct RV, and a slow process that performs MC for predictive RV.

We will go over the two methods used to discover dangerous markings, i.e states that violate safety properties or would inevitably lead to their violation. We will use

the terms *explicit* and *implicit* constraints, introduced by Ding [32], which are, respectively, markings that violate a safety property, and markings that don't violate the property but inevitably lead to the violation of the property. Unsafe transitions are transitions that lead to dangerous markings.

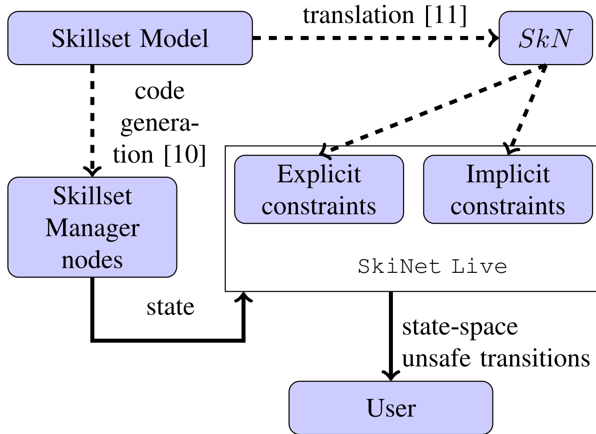

Fig. 2. SkiNet Live is a tool for predictive runtime verification of concurrent skill-based components in a robotic system. Dashed arrows represent operations on the model, while bold arrows represent ROS2 messages [34].

B. Explicit constraints

The tool runs two skillset PNs. Both PNs have their exit places $x_{skill,k}$ and transitions T_{reset} removed, as their existence is only useful for offline MC and analysis to represent intermediate states. However, they do not reflect the online execution of the skills, which can only be in an idle or running state. Moreover, they greatly increase the size of the state-space when performing MC.

During runtime, as the skillset managers of each component are executed, the respective skillset nets are updated. We note SkN_i the skillset net of component i, and I the set of all components. SkN_i is the skillset net whose transitions are fired. All transitions are considered observable, i.e. every action occurring in the skillset managers has a corresponding transition in the skillset PNs. The running marking of SkN_i is noted m_i.

To know if the current marking m_i violates any safety property ϕ, a simple function $safe$ is used, which evaluates the logical expression of ϕ in the combined marking $\sum_{i \in I} m_i$, returning $True$ if the safety property is satisfied, $False$ otherwise

To find out if the markings reachable from m_i are explicit constraints, we propose algorithm 1, which looks for unsafe transitions that lead to immediate safety violation. For each safety property ϕ, it checks if the marking m_i', reached upon firing one of the firable transitions at marking m_i of each skillset net SkN_i, would not violate the safety property. The $safe$ function is used with the new combined marking $m_i' + \sum_{j \in I-i} m_j$, putting the unsafe transition t in the list $unsafe_tr$ of ϕ if $safe$ returns false. This algorithm is fast and inexpensive but can only discover immediate explicit constraints.

Data: I, safety_properties
Result: Get the unsafe transitions $unsafe_tr$ that lead to explicit constraints.

```
1  unsafe_tr = dict();
2  for SkN_i, i ∈ I do
3      m_{I−i} = Σ_{j∈(I−i)} SkN_j.marking;
4      for t ∈ SkN_i.firable() do
5          SkN_i.fire(t);
6          for φ ∈ safety_properties do
7              if ¬safe(SkN_i.marking + m_{I−i}, φ) then
8                  unsafe_tr[φ] ⟵ unsafe_tr[φ] + t;
9              end
10         end
11         SkN_i.undo();
12     end
13 end
14 return unsafe_tr ;
```

Algorithm 1: Unsafe transitions search algorithm

C. Implicit constraints

To find implicit constraints, the eventless skillset net SkN_e is used, a version stripped of its event transitions T_{events}. The reason behind the existence of this modified net is that events are supposed uncontrollable: they only represent the functional layer activity of the component. For instance, in our example, the motor power state for the `spot` skillset can be controlled by the user using the `init_power` and `safe_poweroff` skills, but also by the battery depletion or emergency stop from a security operator, which would trigger the event transition `t_power_switchoff`. When performing predictive RV to guide the user during online operations, it would seem inappropriate to warn the user about events, as they are uncontrollable and only reflect hardware status. Therefore, an eventless net is made to focus the RV and MC on transition paths that only rely on skills.

In order to get the unsafe transitions leading to implicit constraints, we first explore the possible futures with the Sift tool, from the Tina toolbox [28], [29]. This tool allows for a breadth-first exploration of PN markings, where transitions are fired one by one in each layer before moving on to the next. The input net is the union of all SkN_i^e of the system components. We note $\mathcal{M} = \prod_{i \in I} M_i^e$ the set of all possible markings reachable by the union of all SkN_i^e.

This calculation becomes exponentially expensive as the amount of components in the system increases, so a timeout can be set to stop the exploration prematurely. We note $\mu \subseteq \mathcal{M}$ the potentially partial set of markings obtained, returned as a Kripke structure by Sift. In our context of semi-autonomous systems, we consider that, if timeout is reached, verification can be performed sufficiently so that a few operations can be made without worrying about specification violation. To know how many operations the user can perform, the maximum depth of μ, i.e. the largest number of transitions fired before timeout, is given, so that the user can judge how much actions they can perform safely.

μ is recalculated everytime a transition is fired in the system, until Sift finishes its exploration or the timeout is reached. That is why the initial marking of SkN_i^e, $m_{i,0}^e$, is

replaced by the current marking m_i of SkN_i at every transition firing, so that the initial state of μ is m_i. If transitions are fired before timeout, they are ignored until the previous exploration is finished. Nonetheless, the user can always change the timeout value depending on the execution speed of the system. A ROS2 topic called `\skinet\state_space` also sends a message, stating how long Sift took to calculate μ, if the timeout was reached or not (and if so, the depth of exploration), and the size of μ in terms of markings and transitions. These elements could be added to a user interface for the operator to evaluate the trust level of the exploration, but the interaction between the user and the tool through a user interface is left for future work.

To find the dangerous markings $\mu_! \subseteq \mu$ and the paths that could inevitably lead to them, the Kripke structure representing μ is loaded in Muse, a mu-calculus and CTL tool from the Tina toolbox. For each property ϕ, the formula $AF\neg\phi$, literally "for all paths finally, property ϕ is violated" is sent to Muse. This returns the markings $\mu_!$ of ϕ, which contains markings that violate ϕ, i.e. explicit constraints, as well as the markings which would inevitably lead to the violation of ϕ, i.e. implicit constraints.

With the implicit constraints now known, we can add $\mu_!$ as input of algorithm 1, and the condition on line 7 becomes:

$$\neg safe \ or \ SkN_i.marking \in \mu_! \tag{1}$$

With this list of unsafe transitions, and with the knowledge of the current state of each skillset, it could be possible to block the execution as to forbid actions that would trigger unsafe transitions. This is left for future work with fully autonomous systems, but our context focuses on simply informing a human operator, who needs full control of the system.

V. EXPERIMENTAL RESULTS

The Spot® + Kinova Jaco manipulator system was deployed for a long duration mission where a human operator would remotely control its movements and actions, based on the available skills in the `spot` and `manipulator` skillsets. The objective was to neutralize traps, represented as cubes, with a falling panels that needed to be pushed. If the panel falls, the trap is successfully deactivated.

A. Mission protocol

The mission starts with the system idle and the human operator waiting for orders to activate the system and send it at approximate locations where traps have been spotted. Upon receiving the first trap, the operator starts the Spot® robot, makes it stand up and sends coordinates for navigation with the `go_to_waypoint` skill. Once the operator has a visual on the trap thanks to the cameras mounted on Spot®, teleoperation starts to reach the trap, using the `teleop` skill. When sufficiently close to the trap to deactivate it, the operator deploys the robotic arm with `arm_ready` and teleoperates it with `arm_joystick` to push the panel and deactivate the cube. Upon success, the arm must be folded again with `arm_moving_to_RD` (RD=ready position) and then `arm_RD_to_HP` (HP=home position). When receiving a new trap location, the user teleoperates Spot® away from the cube, before letting the autonomous navigation reach the next location with `go_to_waypoint`. A total of four traps had to be deactivated for the full mission to be successful. The mission will be done twice: once with the operator unassisted by the tool, and a second time with knowledge of the unsafe transitions.

B. Safety Properties

The safety properties chosen for this mission are two logical expressions on the state of resources and skills of the skillsets. They are written with a simple notation `skillset_name`:*state* for these states (state names are unique and two resources cannot have similar state names, so specifying which resource or skill is monitored is not necessary). The basic operators *and*, *or* and *not* are used, so that the safety properties are intuitive to write for users.

$$\texttt{action_guard} = not\ (\texttt{spot} : Busy \\ and\ \texttt{manipulator} : Busy) \tag{2}$$

$$\texttt{safety_guard} = not\ (\texttt{spot} : i_go_to_waypoint \\ and\ not\ \texttt{manipulator} : HP) \tag{3}$$

Property (2) monitors the state of the resources `control_mode` of `spot` and `mutex` of `manipulator`. These resources, when equal to $Busy$, mean that a movement skill is running, so the property ensures that Spot® and the arm are not moving at the same time.

Property (3) is to guarantee the safety of the robotic arm during the `go_to_waypoint` skill. If the arm is unfolded, i.e. it is not in its home position HP, then the `go_to_waypoint` skill is forbidden. The goal is to avoid collisions with the environment, as well as making Spot® more stable during long distances walk between waypoints on potentially uneven terrains.

C. State-space calculation

Because our system is only composed of two components, the state-space exploration was complete in 0.014 seconds, with $\mu = \mathcal{M}$ at every calculation, for a total of 440 possible states. This means that all possible implicit and explicit constraints could be predicted at all times. A total of 192 dangerous markings for `action_guard` and 36 for `safety_guard` were found. The size of $\mu_!$ was thus 228, which means that around half of all possible states violate safety properties. This shows how easily the user can unintentionally break the safety properties.

D. Mission results with no guidance

In this first mission, the operator has to manage the system alone and respect the protocol and the safety specifications, without the help of SkiNet Live. An extract of the mission timeline is shown in Fig. 3. Upper half of the figure shows the resources and skills in their current states, while bottom half shows the safety properties status and total number of transitions in the skillset PNs that would lead to dangerous markings, i.e the size of the list retrieved with algorithm 1. As

we can see from the timeline, safety properties were violated

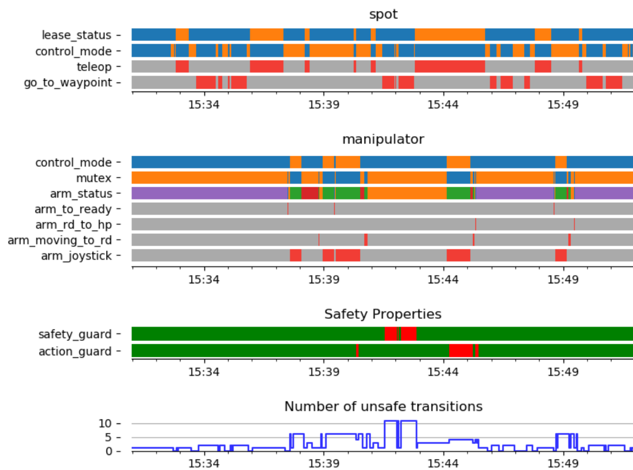

Fig. 3. Experimental results with the human operator with no help from the tool. First graph is an extract of the `spot` skillset state, and second is an extract of the `manipulator` skillset state. Skills are represented with grey (Idle) and red (Running) colors, while resources have various colors depending on their states. When violated, safety specifications appear red. The bottom graph shows the number of transitions that would lead to dangerous markings at a given time. Safety properties were violated several times, intentionally or not.

Upon mission start, the user received the first trap location and sent the robot close to it before starting teleoperation for the final approach, at 15:36. This first trap deactivation went smoothly as the operator was focused and protocol was respected. The user made sure no safety specifications were violated, successfully deactivating the trap. Upon arriving and attempting to deactivate the next trap at 15:40, the user realized the robot was too far from the trap for the arm to deactivate it. The user decided to use the `teleop` skill, but without terminating the `arm_joystick` skill first, leading to both the resources `control_mode` at state $Busy$ and `mutex` at state $Busy$, violating the safety property `action_guard` (2). This safety property is important as the user could inadvertedly move both components simultaneously, or move the wrong one, and endanger the system.

Later on, at 15:42, the user did not put the arm in its home position before sending the robot to the next trap. The resouce `arm_status` was not in the state HP while the skill `go_to_waypoint` was running, leading to the violation of safety property `safety_guard` (3). Since the area of the experiment had trees and ledges, the arm could have collided with them. Moreover, the surface of the ground was uneven at times, so the robot could have become unbalanced and would have fallen down.

The third trap deactivation at 15:45 again led to the violation of `action_guard`, as the user did not terminate the `teleop` skill before manipulating the arm. Finally, the fourth and last trap deactivation at 15:49 went smoothly and protocol was respected.

E. Mission results with SkiNet Live

The same mission was repeated, but this time with the user receiving assistance from SkiNet Live. The user could observe at all times the unsafe transitions that would lead to safety property violation. Figure 4 shows the timeline of this second mission, with a respected protocol and no safety violation throughout the mission duration. The second mission took around 25 minutes to complete, and at the end, the user has been operating the system for almost an hour.

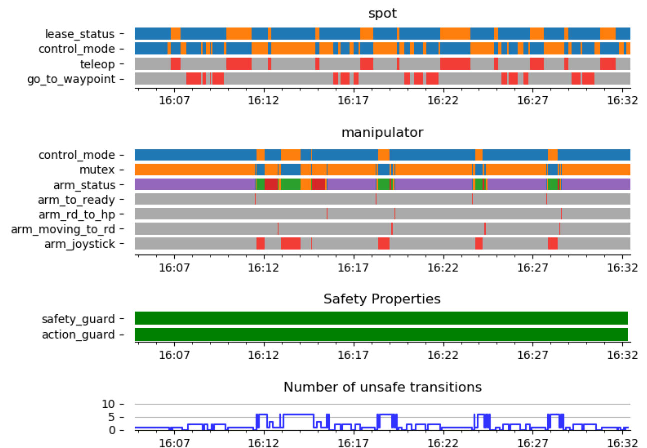

Fig. 4. Experimental results with the human operator being assisted with SkiNet Live during the second half of the mission. Safety properties were respected throughout the mission.

During the mission, the user noted that SkiNet helped him remember to perform some steps of the protocol, such as terminating teleoperation skills on one robot before switching to the other, or putting the arm in the correct position before starting one of its skills. After the second trap, at 16:20, the user was reminded by the tool to put the arm in a safe position before performing the skill `go_to_waypoint`, as the start transition of the skill was flagged as an unsafe transition for the `safety_guard` property by SkiNet Live. At the third trap, at 16:24, the user was warned that starting the `arm_to_ready` was an unsafe transition of `action_guard`, because the `teleop` skill was still runing.

Thanks to the tool, the user was able to follow more easily the mission protocol. The predictive RV performed by SkiNet Live was constantly giving the user guidance to avoid triggering unsafe transitions and reach dangerous markings.

VI. CONCLUSIONS

In this work, we presented a tool which performs RV and online MC on a semi-autonomous complex system to guide a human operator in performing a repetitive task. Experimental results showed that without the tool, safety specifications on the system behavior could be violated by the user, unintentionally or not, after a few task completions. The tool allowed to find and predict faulty states. A second run was made with the user being assisted by the tool, which helped to better follow the mission protocol and avoid safety risks. The next step for this work will be to develop a framework for the human operator to send high-level goals to a robotic system that will autonomously decide the steps to reach the goals, based on its components status and feedback data, while respecting safety specifications.

ACKNOWLEDGEMENTS

This research was supported by the Occitanie region, France.

REFERENCES

[1] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[2] K. Havelund and A. Goldberg, *Verify Your Runs*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 374–383. [Online]. Available: https://doi.org/10.1007/978-3-540-69149-5_40

[3] A. Lotz, A. Steck, and C. Schlegel, "Runtime monitoring of robotics software components: Increasing robustness of service robotic systems," in *2011 15th International Conference on Advanced Robotics (ICAR)*, 2011, pp. 285–290.

[4] J. O. Blech, Y. Falcone, and K. Becker, "Towards certified runtime verification," in *Formal Methods and Software Engineering*, T. Aoki and K. Taguchi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 494–509.

[5] M. Leucker, "Sliding between model checking and runtime verification," in *International Conference on Runtime Verification*. Springer, 09 2012, pp. 82–87.

[6] S. Pinisetty, T. Jéron, S. Tripakis, Y. Falcone, H. Marchand, and V. Preoteasa, "Predictive runtime verification of timed properties," *Journal of Systems and Software*, vol. 132, pp. 353–365, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121217301310

[7] A. Desai, T. Dreossi, and S. A. Seshia, "Combining model checking and runtime verification for safe robotics," in *Runtime Verification*, S. Lahiri and G. Reger, Eds. Cham: Springer International Publishing, 2017, pp. 172–189.

[8] A. Desai, S. Ghosh, S. A. Seshia, N. Shankar, and A. Tiwari, "Soter: A runtime assurance framework for programming safe robotics systems," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 138–150.

[9] J. D. Schierman, M. D. DeVore, N. D. Richards, and M. A. Clark, "Runtime assurance for autonomous aerospace systems," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 12, pp. 2205–2217, 2020. [Online]. Available: https://doi.org/10.2514/1.G004862

[10] A. Albore, D. Doose, C. Grand, C. Lesire, and A. Manecy, "Skill-based architecture development for online mission reconfiguration and failure management," in *2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE)*. IEEE, 6 2021, pp. 47–54.

[11] B. Pelletier, C. Lesire, D. Doose, K. Godary-Dejean, and C. Dramé -Maigné, "SkiNet, a petri net generation tool for the verification of skillset-based autonomous systems," in *Fourth Workshop on Formal Methods for Autonomous Systems*, vol. 371. Open Publishing Association, sep 2022, pp. 120–138. [Online]. Available: https://doi.org/10.4204/EPTCS.371.9

[12] M. Foughali, "Toward a correct-and-scalable verification of concurrent robotic systems: Insights on formalisms and tools," in *2017 17th International Conference on Application of Concurrency to System Design (ACSD)*, 2017, pp. 29–38.

[13] M. Foughali, S. Bensalem, J. Combaz, and F. Ingrand, "Runtime verification of timed properties in autonomous robots," in *2020 18th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 2020, pp. 1–12.

[14] J. Huang, C. Erdogan, Y. Zhang, B. Moore, Q. Luo, A. Sundaresan, and G. Rosu, "Rosrv: Runtime verification for robots," in *Runtime Verification*, B. Bonakdarpour and S. A. Smolka, Eds. Cham: Springer International Publishing, 2014, pp. 247–254.

[15] M. Colledanchise, G. Cicala, D. E. Domenichelli, L. Natale, and A. Tacchella, "Formalizing the execution context of behavior trees for runtime verification of deliberative policies," *CoRR*, vol. abs/2106.12474, 2021. [Online]. Available: https://arxiv.org/abs/2106.12474

[16] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for ltl and tltl," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, sep 2011. [Online]. Available: https://doi.org/10.1145/2000799.2000800

[17] A. Giua and F. DiCesare, "Petri net structural analysis for supervisory control," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 2, pp. 185–195, 1994.

[18] M. D. Jeng, "A petri net synthesis theory for modeling flexible manufacturing systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 27, no. 2, pp. 169–183, 1997.

[19] S. Jiang and R. Kumar, "Supervisory control of discrete event systems with ctl* temporal logic specifications," in *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, vol. 5, 2001, pp. 4122–4127 vol.5.

[20] J. Moody and P. Antsaklis, "Deadlock avoidance in petri nets with uncontrollable transitions," in *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, vol. 2, 1998, pp. 1257–1258 vol.2.

[21] ——, "Petri net supervisors for des with uncontrollable and unobservable transitions," *IEEE Transactions on Automatic Control*, vol. 45, no. 3, pp. 462–476, 2000.

[22] F. Basile, P. Chiacchio, and A. Giua, "Suboptimal supervisory control of petri nets in presence of uncontrollable transitions via monitor places," *Automatica*, vol. 42, no. 6, pp. 995–1004, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109806000963

[23] J. Luo, W. Wu, H. Su, and J. Chu, "Supervisor synthesis for enforcing a class of generalized mutual exclusion constraints on petri nets," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 39, no. 6, pp. 1237–1246, 2009.

[24] J. Luo and M. Zhou, "Petri-net controller synthesis for partially controllable and observable discrete event systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 3, pp. 1301–1313, 2017.

[25] J. Luo, W. Wu, M. Zhou, H. Shao, K. Nonami, and H. Su, "Structural controller for logical expression of linear constraints on petri nets," *IEEE Transactions on Automatic Control*, vol. 65, no. 1, pp. 397–403, 2020.

[26] S. Rezig, N. Rezg, and Z. Hajej, "Online activation and deactivation of a petri net supervisor," *Symmetry*, vol. 13, no. 11, 2021. [Online]. Available: https://www.mdpi.com/2073-8994/13/11/2218

[27] B. Lacerda and P. U. Lima, "Petri net based multi-robot task coordination from temporal logic specifications," *Robotics and Autonomous Systems*, vol. 122, p. 103289, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889019302441

[28] B. Berthomieu, P.-O. Ribet, and F. Vernadat, "The tool tina – construction of abstract state spaces for petri nets and time petri nets," *International Journal of Production Research*, vol. 42, pp. 2741–2756, 7 2004.

[29] B. Berthomieu, F. Vernadat, and S. dal Zilio, "The tina toolbox home page - time petri net analyzer - by laas/cnrs," LAAS/CNRS, 2004.

[30] J.-S. Lee, M.-C. Zhou, and P.-L. Hsu, "Petri net-based design of modular supervisors for remotely human control systems," in *SICE 2004 Annual Conference*, vol. 2, 2004, pp. 1271–1276 vol. 2.

[31] ——, "An application of petri nets to supervisory control for human-computer interactive systems," *IEEE Transactions on Industrial Electronics*, vol. 52, no. 5, pp. 1220–1226, 2005.

[32] Z. Ding, H. Qiu, R. Yang, C. Jiang, and M. Zhou, "Interactive-control-model for human–computer interactive system based on petri nets," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 4, pp. 1800–1813, 2019.

[33] A. Ramirez-Trevino, E. Ruiz-Beltran, I. Rivera-Rangel, and E. Lopez-Mellado, "Online fault diagnosis of discrete event systems. a petri net-based approach," *IEEE Transactions on Automation Science and Engineering*, vol. 4, no. 1, pp. 31–39, 2007.

[34] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074