Petri Nets for the Iterative Development of Interactive Robotic Systems

Pragathi Praveena[1], **Andrew Schoen**[1], **Michael Gleicher**[1], **David Porfirio**[2], **Bilge Mutlu**[1]
[1]Department of Computer Sciences, University of Wisconsin–Madison
[2]NRC Postdoctoral Research Associate, U.S. Naval Research Laboratory
pragathi@cs.wisc.edu, schoen@cs.wisc.edu, gleicher@cs.wisc.edu, david.porfirio.ctr@nrl.navy.mil, bilge@cs.wisc.edu

Abstract

We argue for the use of Petri nets as a modeling language for the iterative development process of interactive robotic systems. Petri nets, particularly Timed Colored Petri nets (TCPNs), have the potential to unify various phases of the development process — *design*, *specification*, *simulation*, *validation*, *implementation*, and *deployment*. We additionally discuss future directions for creating a domain-specific variant of TCPNs tailored specifically for HRI systems development.

Introduction

Imagine the following *coffee shop* scenario.

> *A robot assists the owner of a coffee shop in daily operations, involving friendly banter with patrons and reorganizing tables to accommodate large groups. The robot also works with the pastry chef to arrange cupcakes in the display case. As a large group deliberates over what to order, the robot offers suggestions and manages the group's multiple requests.*

Developing an interactive system for the *coffee shop* scenario involves coordinating the robot's spoken dialogue with social gestures, facilitating joint action between the owner and the robot while reorganizing tables, managing access to shared workspaces as the robot and chef arrange cupcakes, and mediating interaction in group settings during the order process. In the field of Human-Robot Interaction (HRI), the development of such complex systems typically draws on the expertise of a diverse array of *application developers*, including domain experts, designers, programmers, and research professionals, each employing methods, tools, and representations tailored to their specific stage of the development cycle. Additionally, given the iterative nature of systems development, there is often a significant amount of refinement and cross-communication between these phases.

The representations and tools that application developers utilize for their respective phases of system development—such as low-fidelity prototypes by designers and high-fidelity simulators by algorithm developers—may not be effectively interoperable across different phases. We propose that a unifying representation can facilitate a smoother transition across the various stages of the iterative development process. While representations and tools tailored for individual phases are indispensable, we posit that an accompanying, unified representation can enhance this design process by encoding information that can be used to coordinate between phases. We believe that Petri nets, particularly Timed Colored Petri nets (TCPNs), hold the potential to unify various phases of the development process. In what follows, we describe the benefits of TCPNs in terms of their ability to unify all phases of systems development and aid application developers to iteratively cycle between phases.

Related Work

Petri nets are a class of state machines that specialize in modeling the flow and dependencies between events and resources in distributed and concurrent systems (Peterson 1977). Petri nets are defined as a set of *place* nodes that hold *tokens*, *transition* nodes that indicate actions that move tokens from place to place, and directed *arcs* between places and transitions that define the logic of where and which tokens are consumed and produced by transitions. *Firing* a transition, *i.e.,* consuming tokens to execute a transition, depends on the availability of sufficient tokens on incoming arcs from places to each transition. The *marking* of the Petri net is the arrangement of tokens across places in the net and indicates its current state. *High-level* Petri nets represent a class of Petri nets with additional features that allow for a more compact representation of dependencies and behaviors while producing a more semantically meaningful graph (Jensen 1983). A *Colored* Petri net is a High-level Petri net that includes data values of different types (*colors*) in individual tokens, such as integers, strings, or user-defined types, and *guard* functions that evaluate whether the tokens present in incoming arcs are sufficient for the transition to execute. A *Timed* Colored Petri net introduces the concept of time, *e.g.,* to indicate the time a token must remain in a place before it can move or the time a transition will take to fire once it is enabled. A full description of the formal specification of TCPNs can be found in Jensen and Kristensen.

Although Petri nets have had a relatively modest adoption in robotics (Costelha and Lima 2007; Chao and Thomaz 2016; Brooks 2017), they have demonstrated utility in wide-ranging applications that span game design, biological systems, and industrial processes. Game narratives and player interactions mirror the complexities found in robotic systems, and Petri nets have been successfully used to model and analyze game mechanics (Balas et al. 2008; Araújo and Roque 2009; Muratet, Carron, and Yessad 2022). Variants of Petri nets have been used to model complex biological processes such as viral infection and 2D diffusion (Assaf 2022), molecular transport across cell membranes (Liu 2012), and

metabolic reactions between enzymes and substrates (Baldan et al. 2010). Biological processes share many characteristics with industrial and workflow-based processes, where multiple actors may come together to transform components or resources into products. Van der Aalst created a variant of Petri nets called *Workflow-Nets (WF-nets)*, and Petri nets have been used to optimize agent-task allocations and scheduling between humans and robots (Casalino et al. 2019; Cheng, Sun, and Fu 1994; Drakaki and Tzionas 2017).

Value Proposition of the Representation

We posit that TCPNs hold much untapped potential within HRI. Specifically, TCPNs can yield cumulative benefits by serving as a unifying representation for all phases of systems development. We find it useful to think of systems development in terms of six phases as shown in Figure 1: (1) *design*, (2) *specification*, (3) *simulation*, (4) *validation*, (5) *implementation*, and (6) *deployment*. In what follows, we provide details about how TCPNs can enhance each of these phases.

Design. Design is the conceptualization phase of systems development where user requirements are gathered through observations and participatory design, scenarios are created to envision system interactions, and the system architecture is structured. A high-level plan is established at this stage for how the system will meet user needs and technical considerations, and produces tangible outcomes such as wireframes, user flows, personas, and low-fidelity prototypes.

A key property that enables Petri nets to enhance the *Design* phase is their ability to be used at different levels of abstraction. In the early stages of design, Petri nets support high levels of abstraction, *e.g.,* modeling the robot's high-level dialogue, joint action, and decision-making behavior. This model can then be gradually refined to yield a more detailed and precise description of the system under consideration, ultimately resulting in an executable prototype of the system (see *Implementation*). The ability for the representation to have various levels of abstraction could make it an appropriate shared language for communication between domain experts, end-users, designers, and programmers. This was demonstrated by (Jensen and Kristensen 2009) in a case study where a suitably abstracted Petri net-based model was used to engage end-users (hospital nurses) for elicitation, negotiation, and agreement of user requirements.

Specification. Specification involves defining the requirements, behavior, features, and constraints of the system being developed. Specifications expressed in natural language or in visual formats like flowcharts and activity diagrams are prone to ambiguity. Additionally, other representations (*e.g.,* state machines) suffer from an explosion of state spaces as the complexity of the system increases, resulting in representations that are less comprehensible for human interpretation. In contrast, TCPNs can model the same phenomenon in a much more compact way. Hence, they provide a precise, concise, and human-friendly way to represent system behavior, yielding models that are both mathematically sound and accessible to systems developers through their visual and graphical nature (Jensen and Kristensen 2009).

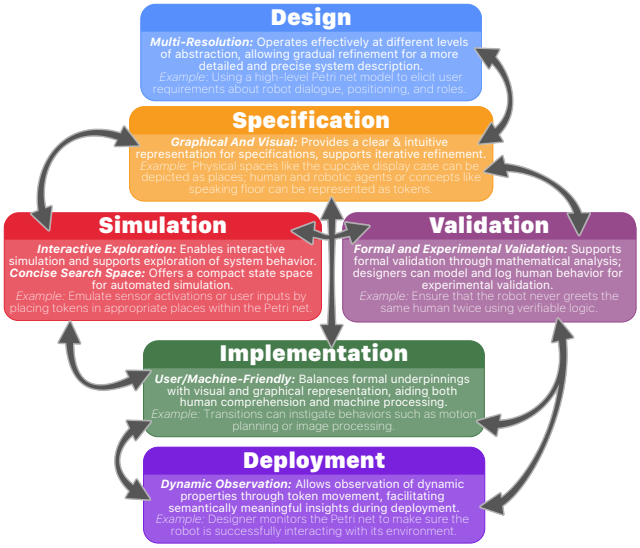

Figure 1: Flow between development phases and properties and examples of how Petri nets are useful at each.

The Petri net representation can be designed to map to semantically meaningful domain properties (*e.g.,* distinct physical areas within a setting, human and robotic agents, and abstract concepts like speaking floor). For instance, physical spaces like tables or the cupcake display case can be depicted as *places*, and the robot can be represented by a *token*. The movement of the robot can be specified and visualized as the movement of the *token* through the Petri net. When the robot *token* occupies the cupcake display case *place*, we can semantically infer that the robot is positioned at the display case, potentially assisting the pastry chef. Additionally, Petri nets can facilitate the development of complex systems by enabling users to specify smaller subprocesses, which can then be automatically combined using mathematical properties into a behavior model of a larger and more intricate system. Finally, the formal specification of system behavior using Petri nets is beneficial for the downstream process of *Validation* of the system through the application of formal techniques.

Simulation. Simulation involves viewing how a system behaves under different conditions via a simplified representation of real-world phenomena. Any valid Petri net will support simulation, and visual inspection can be enabled in two ways: (1) *interactive*, where users have the ability to actively engage with the Petri net model to explore hypothetical scenarios by placing *tokens* or triggering *transitions*; and (2) *automated*, where system behavior is analyzed without user intervention based solely on the design of the system and possible initial markings. The results of these simulations could prompt the designer to consider alterations to the *Specification*, *Implementation*, or prompt *Validation*.
Interactive: By placing *tokens* in appropriate *places* within the Petri net, developers can emulate real-world events such as sensor activations and user commands. Due to the visual and graphical nature of the representation, developers can

visually track the movement of *tokens* as they traverse *transitions* and *places*. The comprehensibility of the interactive simulation is enhanced when the Petri net elements hold semantic significance (as elaborated in *Specification*).

Automated: Petri nets can facilitate the methodical exploration and discovery of optimal system parameters via the use of automated simulations. This can be particularly valuable for the design of complex systems where there are potentially many choices for possible transitions given a certain *marking*, and in which naively selecting or handcrafting actions and managing token consumption can lead to suboptimal outcomes. Additionally, in interactive robotics systems, it is important to account for actions from both controlled (*e.g.,* robots and other automated systems) and uncontrolled (*e.g.,* workers, customers, and pets) agents, as these actions collectively impact interaction quality.

The exploration of these options and the determination of optimal strategies can be accomplished through a combination of simulation and methods like search or reinforcement learning. These methods can be relatively simple, as with (Cheng, Sun, and Fu 1994), who used A^* search to determine an optimal firing sequence of transitions in a Petri net that modeled manufacturing systems. More recent methods have utilized reinforcement learning and deep learning to derive optimal strategies, *i.e.,* policies, to support a nuanced set of interactions (Drakaki and Tzionas 2017; Hu et al. 2020).

The focus of our work is on high-level control or behavior modeling, thereby reducing the need for many off-the-shelf robotics simulators that instead focus on realistic physics, robot control, and computer vision (*e.g.,* NVIDIA's Omniverse (Mittal et al. 2023) or Habitat (Manolis Savva et al. 2019; Szot et al. 2021)). Only the features specified as *tokens* and the structure of the Petri net need to be encoded, reducing the state vector size. Furthermore, the action space is generally more constrained given specific markings, so methods such as action masking can be used to improve learning in these action spaces (Huang and Ontañón 2020). As such policies can be trained prior to deployment, they require minimal computational power to execute in real-time. However, this does not preclude online learning if the developer wants to support *in-situ* adaptation.

Validation. Validation involves assessing whether the system is built correctly according to its design and performs its intended functions accurately and reliably. HRI systems can benefit from both *formal* and *experimental* validation techniques. Formal techniques can be used to prove whether programs adhere to specific properties (Wing 1990), such as ensuring a robot never greets the same human twice (Porfirio et al. 2018). On the other hand, experimental techniques are useful for validation of user behavior and ergonomic properties (Ait-Ameur and Baron 2006), such as ensuring that an individual is able to effectively communicate with the robot in the presence of ambient noise. Petri nets are *formal* models that can be shared between formal and experimental validation, and bridge the gap between these techniques. Here we elaborate on formal validation and discuss experimental validation under *Deployment*.

It is often useful to verify the temporal or probabilistic behavior of complex systems. For example, developers may need to ensure that the duration of a specific task probabilistically falls within a certain time bound if performed collaboratively with a robot. This and similar phenomena can be captured within verifiable logic, such as *Signal Temporal Logic* (Donzé and Maler 2010), *e.g.,* $\mathrm{P} = ? \, \mathbf{F}_{[10,20]} \, \texttt{[success]}$, or in plain English, *what is the probability that the task will take between 10 and 20 seconds?*

While expressing such properties is straightforward, their verification is challenging. Example prior work in robotics involves sampling simulated trajectories through methods such as Monte Carlo (Scher, Sadraddini, and Kress-Gazit 2023). Combined with high-fidelity off-the-shelf simulators (*e.g., RoboSuite*, Zhu et al. 2020), such techniques have been shown to be successful, though they are restrictive to a small set of domains and are expensive due to the large number of samples required and simulator performance limitations.

The concise and semantically meaningful properties of Petri nets are also well-suited for validation. Sampling a single trajectory in a Petri net is much less computationally intensive than if done in simulators with high realism, *e.g.,* those with photorealism and physics engines. Although this higher level of abstraction in modeling by the Petri net as compared to high-fidelity simulation may constrain users to verifying coarse approximations of phenomena, we hypothesize that it facilitates system modeling at a level of abstraction that requires less effort compared to equivalent representations (*e.g.,* probabilistic timed automata).

Implementation. The *Design*, *Specification*, *Simulation*, and *Validation* stages of the systems development process produce a precise, concise, and human-friendly model in the form of a Petri net, which can be integrated into the system's implementation. This integration eliminates any loss of information or unintended alterations that might occur during translations between different representations at various phases. As a result, the specified behaviors and interactions captured within the Petri net model can be faithfully translated into the program's execution. If, at any point, the designer realizes changes need to be made to the implementation, since the model is still represented as a Petri net, this can seamlessly be introduced into the previous *Specification*, *Simulation*, and *Validation* steps.

Existing Petri net libraries can facilitate the implementation of Petri net models. For instance, the SNAKES library (Pommereau 2015) supports a wide range of Petri net models and allows the use of Python objects as *tokens* and Python expressions for *transition* guards or *arc* conditions. If a developer uses SNAKES for *Design*, *Specification*, *Simulation*, and *Validation*, the same SNAKES models can then be used as the skeleton upon which the full implementation can be built. For example, *transitions* within Petri nets, which often signify system actions, can instigate system behaviors (such as motion planning or image processing) when they are fired. Further, sensor activations and user commands can introduce *tokens* into the Petri net. Utilizing a Petri net-based representation for modeling system behavior allows the separation of high-level logic from low-level processes (*e.g.,* sensing and motion planning).

Deployment. Petri nets can be beneficial to monitor and visualize the real-time execution of finished systems. This can prove challenging with alternative representations—state machines tend to be large and difficult to visualize, while block-based representations lack an explicit portrayal of movement, progression, or transfer of system entities (*e.g.,* tracking the location of a cupcake or monitoring occupancy of a physical space). The compact and graphical nature of Petri nets provides a clear depiction of the system flow, especially with the use of tokens to model dynamic behaviors and allow for tracing an entity through the system.

Petri nets are ideal for run-time analysis and experimental validation during deployment because they model complex interactions, capture dynamic behaviors, and provide a visual presentation of system behavior. Complex systems are characterized by many free interactions, leading to elaborate interaction traces that are challenging for analysis and interpretation by people. (Muratet, Yessad, and Carron 2016) use a Petri net representation to *algorithmically* analyze and label player behaviors during a game. However, even for the *manual* generation of labels, qualitative coding of Petri net traces could serve as a complement to, or even a potential substitute for, video coding. Moreover, Petri net elements could be time-stamped or tagged with data for dynamic behavior (*e.g.,* when transitions are fired) to facilitate downstream analysis such as understanding response times, waiting times, and throughput. This approach could enable a more direct translation of empirical insights from deployment into tangible improvements in the design, specification, or implementation of system behavior. We emphasize that the efficacy of this approach hinges on the model effectively capturing all aspects that the developer seeks to analyze. This is achievable using Petri nets because of the representation's versatility in expressing a variety of concepts, including user-activity model, task model, and context model.

Case Study: *Arranging Cupcakes*

Returning to the *coffee shop* scenario in which the robot assists the pastry chef in arranging cupcakes, we present three processes and observe how they can be modeled through a Petri net-based representation (see Figure 2).

Cupcakes transfer: Both the chef and the robot can transfer cupcakes from a tray to a display case. We model this by two places: *Tray Space* and *Case Space* . Both the human and the robot start outside this Petri net module and are represented by tokens of distinct color values but the same *Agent* color type. An agent transitions into the module from an external one and occupies the tray to collect a cupcake token . Then, both the agent and the cupcake would transfer to the case , resulting in a cupcake displacing an empty spot in the case .

Single-occupancy enforcement: To ensure safety, we add a *Resource Place* , which contains tokens indicating the occupancy of the tray or case. An agent can occupy a space if the agent possesses the requisite resource token for either the tray or the case . This design enforces the robot to follow the single-occupancy rule. However, if the human accesses the space while the robot is already present , the model assigns an error token to the human, which can then

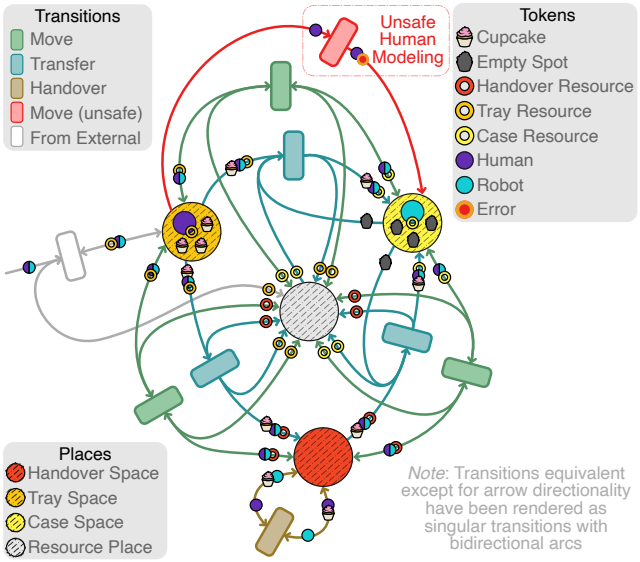

Figure 2: An example model of processes related to the scenario where the robot assists the chef in arranging cupcakes.

be utilized by subsequent processes to either stop the robot or withdraw it from the space. This approach allows us to model the way humans might violate expectations within the same representation used to specify robot behavior, thereby enabling simulation or validation of such behaviors.

Double-occupancy for handovers: To model handovers, where one agent delivers a cupcake to another, who then transports it to the tray, a place called *Handover Space* is introduced. One agent arrives with a cupcake token, another without a cupcake token, and both agents would have to be in possession of the handover resource tokens to enable cupcake handover . Upon exiting the transition, the agent, initially without a cupcake token, now possesses one.

Discussion

Based on the value proposition that we discuss above, we argue that Petri nets could be an indispensable tool for iterative systems development. However, we believe that there is a need for an extended, domain-specific TCPN that is tailored for HRI systems development. An extended representation might enforce, for instance, that physical locations in the robot's environment are represented as *places*, or that *tokens* are drawn from a predefined set of *colors* that represent the robot, the human, or the availability of physical space (see the *Case Study* above). Thus, our domain-specific instantiation will inherit from TCPNs, contain semantic significance to HRI systems, and be accessible to developers lacking experience with Petri nets. Given an extended TCPN representation, we also advocate the need for an end-user development tool that selectively exposes aspects of the underlying representation to developers in an intuitive, graphical way.

Our future work will involve creating a new representation specifically tailored to HRI that extends TCPNs and evaluating its effectiveness in terms of developer accessibility. We believe that Petri nets can enable the development of more complex and nuanced human-robot interactions.

Acknowledgements

This work was supported by a NASA University Leadership Initiative (ULI) grant awarded to the UW–Madison and The Boeing Company (Cooperative Agreement #80NSSC19M0124) and National Science Foundation (NSF) awards 1925043 and 2026478. This work was conducted while David Porfirio was an NRC Postdoctoral Research Associate at the Naval Research Laboratory. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the US Navy or NSF.

References

Ait-Ameur, Y.; and Baron, M. 2006. Formal and Experimental Validation Approaches in HCI Systems Design Based on a Shared Event B Model. *International Journal on Software Tools for Technology Transfer*, 8: 547–563.

Araújo, M.; and Roque, L. 2009. Modeling Games with Petri Nets. In *Digra conference*. Citeseer.

Assaf, G. 2022. Fuzzy Coloured Petri Nets for Modelling Biological Systems with Uncertain Kinetic Parameters. Doctoral thesis, BTU Cottbus - Senftenberg.

Balas, D. B. D.; Brom, C.; Abonyi, A.; and Gemrot, J. 2008. Hierarchical Petri Nets for Story Plots Featuring Virtual Humans. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, 2–9.

Baldan, P.; Cocco, N.; Marin, A.; and Simeoni, M. 2010. Petri Nets for Modelling Metabolic Pathways: A Survey. *Natural Computing*, 9: 955–989.

Brooks, N. 2017. *Situational Awareness and Mixed Initiative Markup for Human-Robot Team Plans*. Ph.D. thesis, Carnegie Mellon University.

Casalino, A.; Zanchettin, A. M.; Piroddi, L.; and Rocco, P. 2019. Optimal Scheduling of Human–Robot Collaborative Assembly Operations with Time Petri Nets. *IEEE Transactions on Automation Science and Engineering*, 18(1): 70–84.

Chao, C.-M.; and Thomaz, A. 2016. Timed Petri Nets for Fluent Turn-Taking over Multimodal Interaction Resources in Human-Robot Collaboration. *The International Journal of Robotics Research*, 35(11): 1330–1353.

Cheng, C.-W.; Sun, T.-H.; and Fu, L.-C. 1994. Petri-Net Based Modeling and Scheduling of a Flexible Manufacturing System. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 513–518. IEEE.

Costelha, H.; and Lima, P. 2007. Modelling, Analysis and Execution of Robotic Tasks using Petri Nets. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1449–1454.

Donzé, A.; and Maler, O. 2010. Robust Satisfaction of Temporal Logic Over Real-Valued Signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 92–106. Springer.

Drakaki, M.; and Tzionas, P. 2017. Manufacturing Scheduling Using Colored Petri Nets and Reinforcement Learning. *Applied Sciences*, 7(2): 136.

Hu, L.; Liu, Z.; Hu, W.; Wang, Y.; Tan, J.; and Wu, F. 2020. Petri-net-based Dynamic Scheduling of Flexible Manufacturing System via Deep Reinforcement Learning with Graph Convolutional Network. *Journal of Manufacturing Systems*, 55: 1–14.

Huang, S.; and Ontañón, S. 2020. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. *arXiv preprint arXiv:2006.14171*.

Jensen, K. 1983. High-Level Petri Nets. In *Applications and Theory of Petri Nets: Selected Papers from the 3rd European Workshop on Applications and Theory of Petri Nets Varenna, Italy, September 27–30, 1982 (under auspices of AFCET, AICA, GI, and EATCS)*, 166–180. Springer.

Jensen, K.; and Kristensen, L. M. 2009. *Coloured Petri Nets*. Berlin, Heidelberg: Springer.

Liu, F. 2012. Colored Petri Nets for Systems Biology. Doctoral thesis, BTU Cottbus - Senftenberg.

Manolis Savva; Abhishek Kadian; Oleksandr Maksymets; Zhao, Y.; Wijmans, E.; Jain, B.; Straub, J.; Liu, J.; Koltun, V.; Malik, J.; Parikh, D.; and Batra, D. 2019. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Mittal, M.; Yu, C.; Yu, Q.; Liu, J.; Rudin, N.; Hoeller, D.; Yuan, J. L.; Tehrani, P. P.; Singh, R.; Guo, Y.; Mazhar, H.; Mandlekar, A.; Babich, B.; State, G.; Hutter, M.; and Garg, A. 2023. ORBIT: A Unified Simulation Framework for Interactive Robot Learning Environments. .

Muratet, M.; Carron, T.; and Yessad, A. 2022. How to Assist Designers to Model Learning Games with Petri Nets? In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, FDG '22. New York, NY, USA: Association for Computing Machinery. ISBN 9781450397957.

Muratet, M.; Yessad, A.; and Carron, T. 2016. Understanding Learners' Behaviors in Serious Games. In Chiu, D.; Marenzi, I.; Nanni, U.; Spaniol, M.; and Temperini, M., eds., *Advances in Web-Based Learning – ICWL 2016*, volume 10013 of *Lecture Notes in Computer Science*. Springer.

Peterson, J. L. 1977. Petri Nets. *ACM Computing Surveys (CSUR)*, 9(3): 223–252.

Pommereau, F. 2015. SNAKES: A Flexible High-level Petri Nets Library. In *Proceedings of PETRI NETS'15*, volume 9115 of *Lecture Notes in Computer Science*. Springer.

Porfirio, D.; Sauppé, A.; Albarghouthi, A.; and Mutlu, B. 2018. Authoring and Verifying Human-Robot Interactions. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, 75–86. New York, NY, USA: Association for Computing Machinery. ISBN 9781450359481.

Scher, G.; Sadraddini, S.; and Kress-Gazit, H. 2023. Probabilistic Rare-Event Verification for Temporal Logic Robot Tasks. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 12409–12415. IEEE.

Szot, A.; Clegg, A.; Undersander, E.; Wijmans, E.; Zhao, Y.; Turner, J.; Maestre, N.; Mukadam, M.; Chaplot, D.;

Maksymets, O.; Gokaslan, A.; Vondrus, V.; Dharur, S.; Meier, F.; Galuba, W.; Chang, A.; Kira, Z.; Koltun, V.; Malik, J.; Savva, M.; and Batra, D. 2021. Habitat 2.0: Training Home Assistants to Rearrange their Habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Van der Aalst, W. M. 1998. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(01): 21–66.

Wing, J. 1990. A Specifier's Introduction to Formal Methods. *Computer*, 23(9): 8–22.

Zhu, Y.; Wong, J.; Mandlekar, A.; Martín-Martín, R.; Joshi, A.; Nasiriany, S.; and Zhu, Y. 2020. RoboSuite: A Modular Simulation Framework and Benchmark for Robot Learning. *arXiv preprint arXiv:2009.12293*.