Model-Based Adaptation for Robotics Software

Jonathan Aldrich, David Garlan, Christian Kästner, Claire Le Goues, Anahita Mohseni-Kabir, Ivan Ruchkin, Selva Samuel, Bradley Schmerl, Christopher Timperley, Manuela Veloso, and Ian Voysey, Carnegie Mellon University

Joydeep Biswas, Arjun Guha, and Jarrett Holtz, University of Massachusetts, Amherst

Javier Cámara, University of York

Pooyan Jamshidi, University of South Carolina

// We developed model-based adaptation, an approach that leverages models of software and its environment to enable automated adaptation. The goal of our approach is to build long-lasting software systems that can effectively adapt to changes in their environment. //

Digital Object Identifier 10.1109/MS.2018.2885058
Date of publication: 22 February 2019

MOBILE ROBOTICS IS a growing area of research, but the long-term deployment of such robots has had limited success. A major challenge in the field is that robotics software is often purpose built for a given robot, task, and environment. Adapting that software to accommodate even small changes in the robot's hardware, mission, or ecosystem is difficult. At Carnegie Mellon University (CMU), CoBots have been assisting humans for the last seven plus years by escorting visitors, delivering messages, and carrying out other tasks. But getting these robots to perform as intended requires a good deal of human effort. So the CoBots remain mostly confined to the controlled environment of the Computer Science building.

The Model-Based Adaptation for Robotics Software project (http://www.cs.cmu.edu/~brassmars) seeks to change the way mobile robotics software is made so that it can be more easily adapted to different hardware, tasks, and environments. We want to build software so flexible that it can adapt itself to changing environments over a period of years or even decades. Our primary intellectual leverage comes from models: formal descriptions of the structure and properties of robotic software, and how these respond to environmental change.[1] Using these models, our approach will enable robots to automatically explore potential adaptations to the system architecture and code and then choose the adaptation that best meets system objectives in the current environment.

Change and Adaptation in Robotics Software

When a robot is in service for years, it is likely to change in many ways. Sensors are replaced or removed, software is upgraded, and parts show wear and tear. All of these changes affect how

the robot senses the world, makes decisions, moves, reasons, and interacts with its environment. Meanwhile, as time goes on, the robot may have to operate in new environments and be expected to perform new tasks.

These changes require time-consuming manual adaptation in today's mobile robotics systems. For example, the person installing or updating a sensor on a robot must learn how that new sensor uses system resources, such as power, and how the data it provides affect downstream components. Today, this requires painstaking data gathering and parameter tuning, much of which is done manually. Upgrading individual software components—not to mention software frameworks, such as the Robotic Operating System (ROS)—may result in incompatibilities that prevent the software from running correctly, consequently requiring extensive programmer time to debug the system.

While developing and maintaining our CoBots, we have observed that addressing these issues takes up enormous amounts of time, thus illustrating how this problem is a significant barrier to the more widespread deployment of similar systems.

Our Approach: Intelligent Model-Based Adaptation

We propose intelligent model-based adaptation (Figure 1), an approach in which developer-specified and automatically discovered models are leveraged to enable robotics software to autonomously adapt to changes in the ecosystem. Our models capture the intent of the system and its components at a higher level of abstraction compared to the source code. For example, a model of software architecture represents the high-level decomposition of the system into components and captures how those components communicate, providing a tool for reasoning through system-level adaptation. Other models represent the system and its environment, state-machine control, triggers for replanning, utility, system behavior, and the current plan.

Once models have been specified by developers or automatically discovered via observations of the system, we can leverage them in a monitoring component that detects the need for adaptation, then use them to search intelligently among possible adaptations and select an appropriate adaptation for a given change in the environment. Our adaptation approaches include replanning and architectural adaptations as well as code-level adaptations.

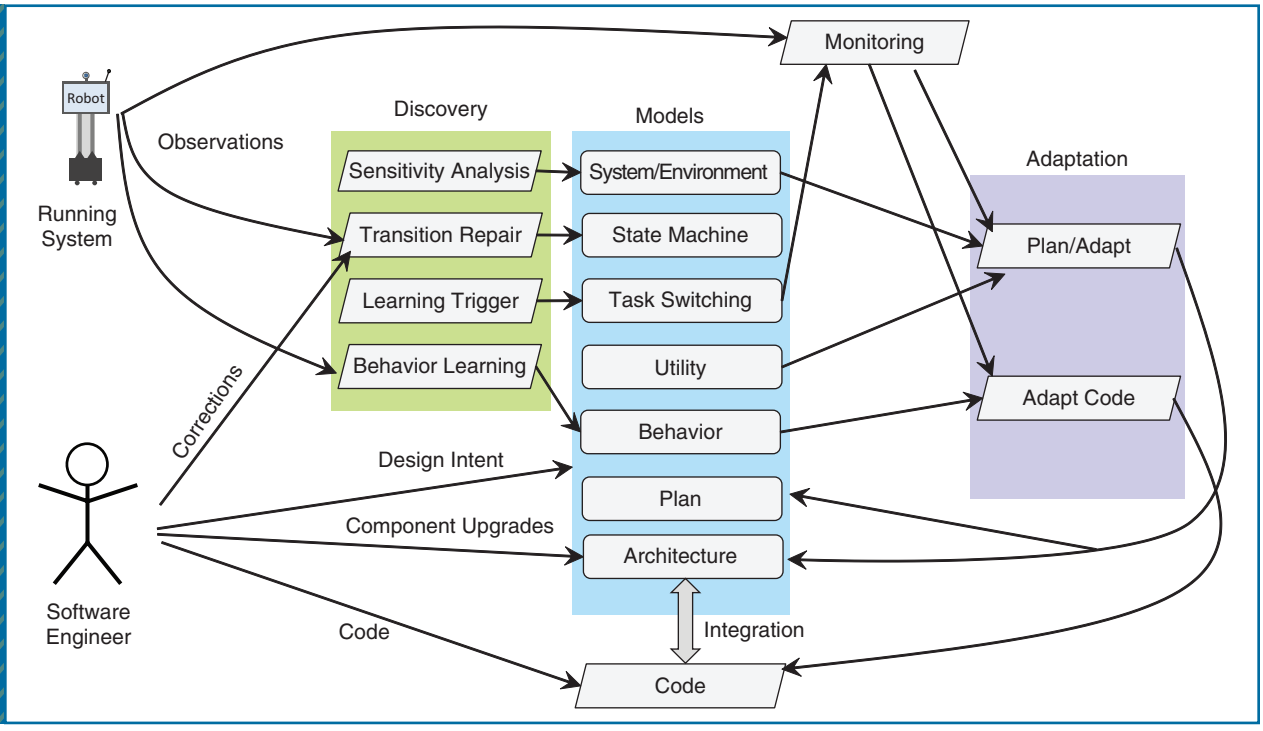


FIGURE 1. A diagram illustrating the intelligent model-based adaptation approach.

Sensitivity Analysis

Currently, environmental changes typically require time-consuming manual reconfiguration or tuning; we would prefer robots to adapt to these changes automatically by discovering models that explain the effects of all sorts of configuration changes in a given environment. These models would then be applied to adapt the software, e.g., by choosing among algorithms or sensors to find the most effective tradeoff between the quality of service and use of resources.

We apply sensitivity analysis, a technique based on sampling and machine learning, to automatically discover models governing how the robot's resource use and performance depend on its configuration. Model discovery is expensive, particularly in long-lasting robots that may have a large configuration space; e.g., four sensors that have five Boolean configuration options each yield a million combinations. It is not feasible to test all combinations, so we have developed techniques both to learn the effect of individual configuration options and to discover when multiple options interact.[2] We have also reduced learning-related costs via transfer learning, which combines large amounts of data cheaply gathered from simulations with carefully selected data from the real robot to predict how the robot will operate in different situations.

State Transition Repair

Complex robot tasks are typically modeled as state machines, where each state encapsulates a feedback controller. Transitions between states are triggered when the observed state of the world matches thresholds in a multidimensional parameter space. However, these thresholds are hard to get right, even for experienced roboticists. It is common for parameter values to work in one physical environment but fail in another or to work on one robot but fail with another. In long-running robotic software, it is important for nontechnical users to be able to give the robot feedback on its behavior—e.g., telling a robotic assistant that it should have asked a human for help earlier instead of wandering the halls for an hour—and have the robot automatically adjust its state transition thresholds accordingly.

We have introduced satisfiability modulo theories (SMT)-based Robot Transition Repair (SRTR),[3] an approach to adjusting the parameters of robot state machines using sparse, partial specifications of corrections from humans. This approach is general to any robot action as long as it fits the model of a robot finite-state machine. During implementation, we log execution traces of the transition function. Afterward, the human corrects one or more transitions. SRTR then takes the transition function source code and the corrections as input and then produces adjustments to the parameter values—automatically isolating parameters that are repairable and translating a set of resulting constraints to an off-the-shelf MaxSMT solver that produces adapted parameter values.

Learning Triggers for Task Switching

Our CoBot robotic assistants can autonomously deliver messages and escort visitors while avoiding obstacles. However, they cannot respond to their surroundings, e.g., reporting a problem such as a water spill. Long-lasting service robots will be assigned many new tasks throughout their lifetime. They must learn how to automatically switch among such tasks.

We evaluated a baseline approach to task switching by modeling tasks as a Markov decision process (MDP) with rewards and composing all tasks together into one MDP. This approach becomes computationally intractable as the number of tasks increases, as expected in long-lasting systems, because the state space grows combinatorially. Furthermore, the global MDP approach necessitates that sensors involved in all tasks run constantly, requiring too much energy and processing power.

We developed a new approach that adds the ability to interrupt one task and switch to another based on new observations. With our method, the robot learns policies and rewards for each task, then learns a task-selection policy that incorporates synergies between different tasks.[6] For example, reporting a spill can be done conveniently if it is observed while performing another task that is not urgent. We also learn the most important observations for triggering the interruption of one task and consequent switch to another. Focusing on these narrow stimuli substantially saves on the estimated costs of sensing. Through our stimuli-based task switching, robotics software can easily scale up to more tasks and seamlessly adapt to changes in the environment.

Architectural Adaptation: Models and Code Integration

A key aspect of our approach is architectural adaptation—enabling the robot to consider a variety of available algorithms and sensors and

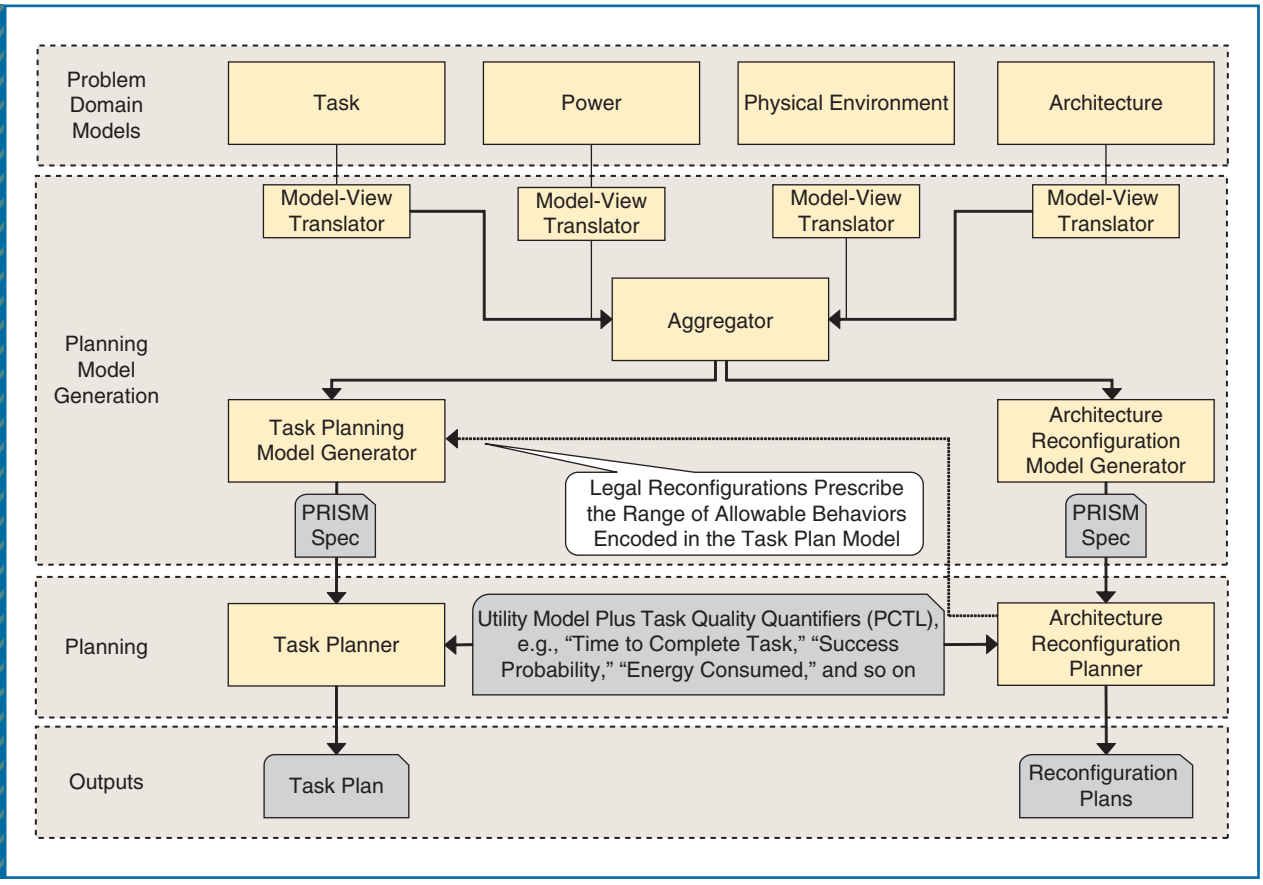


FIGURE 2. A diagram illustrating our approach to model integration.

automatically find and enact a configuration of the system that meets end-user needs. This approach requires a model of the system's architecture that is not just traceable to code but also integrated with it so that changes to the code are reflected in the model and vice versa.

Our approach enforces component isolation, ensuring that components by default cannot communicate with each other or access system resources.[5] This creates loose coupling between components and facilitates architectural adaptation. An architecture description language specifies how to bind those components together:

```
architecture SenseAct
    component s:Sensor
    component a:Actuator
    connector senseData:ROSTopic
    connect s.out, a.in with senseData.
```

Our robots are based on ROS, so the (greatly simplified) previously described architecture uses an ROS topic connector to pass sensor input to an actuator that uses it for further processing. The ROS topic connector generates the appropriate communication boilerplate so that the sensor and actuator components can be written without knowing the details of the connector, allowing a connector with different characteristics or even a connector from a later version of ROS to be substituted without affecting the component code.

Multimodel Integration and Planning

As described, our approach leverages multiple models, including models of the robotic system's tasks, power usage, and physical environment as well as its software architecture. In a long-running system, new models must be added to reflect new software, hardware, and resources as well as new environments and tasks. Such long-lasting systems require a systematic approach to integrating multiple models and using them to

reason about how to adapt the robot's configuration and behavior.

In our approach to model integration, illustrated in Figure 2, we use a translator to project each model to a view in a common language, where the views are checked for consistency.[7] An aggregator then composes this information. Models for task planning and architectural reconfiguration are generated in the input language for the PRISM model checker.[9] A planner leverages probabilistic model checking together with probabilistic computation tree-logic formulas describing safety properties and utility to produce plans for carrying out individual tasks and for reconfiguring the system when needed. The key benefits of this approach include extensibility (new types of models can be added), generality (the planning mechanism supports an arbitrary number of quantifiable quality dimensions), assurance (the probabilistic planner provides quantitative guarantees about behavior), and automation (system reconfiguration actions and task planning can be directly synthesized from models).

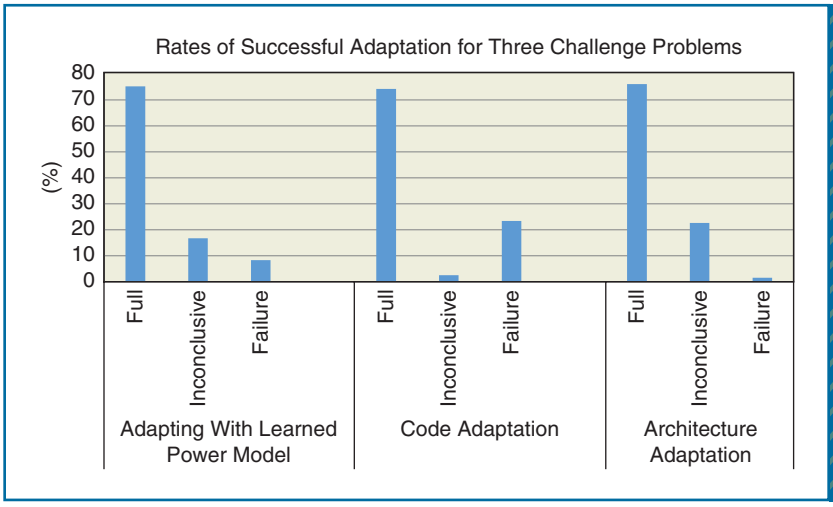


FIGURE 3. A graph showing experimental results for the three adaptation scenarios. Full means that the adaptation enabled normal operation after a perturbation. Inconclusive means we were not able to assess whether the adaptation was successful. Failure means normal operation was not restored.

Adapting Source Code

Many classes of adaptations require changes to source code. Our goal is to automatically generate adapted source code in many situations—e.g., when a new component is added to the system but contains a defect or an interface mismatch compared to a previous version of the component.

Unfortunately, long-lasting robotics software poses challenges of scale and complexity that significantly outpace the ability of state-of-the-art patch generation techniques, which typically target bugs that can be fixed locally and require a comprehensive repeatable test suite to localize the fix and evaluate repairs. Robotics code tends to have nonlocal effects, making it difficult to identify where a patch should be applied. Furthermore, the requirement to simulate both the robot and its environment makes validation significantly more complex; constructing large, repeatable test suites is rarely feasible, and doubts have been raised about whether typical robotics defects can be found in simulation.

To address these challenges, we have built a test generation framework called *Houston* (https://github.com/squaresLab/Houston), which captures a robotics system as a black box with an explicit configuration space, a space of possible incoming events, and observable behaviors. Houston's abstractions allow the systematic generation of tests for constructing the models of behavior in Figure 1 as well as for guiding the selection of code-level transformations and evaluating their effectiveness. Leveraging this framework, we demonstrated that our techniques can find, verify, and correct many recently found real-world defects in Ardu-based robotics systems using purely software-in-the-loop simulation.[8]

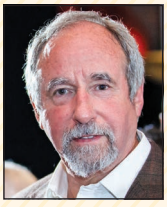
Evaluation

In partnership with an independent third-party evaluator, the Massachusetts Institute of Technology's Lincoln Laboratory, we are evaluating how well robots adapt with our approach. Our evaluation uses the Gazebo simulator to model a TurtleBot delivering messages on a floor of Wean Hall at CMU. The evaluation uses three scenarios, each with a different kind of perturbation to test our system's ability to adapt:

1. New hardware affects the power usage of the TurtleBot. We use sensitivity analysis to discover a model of power consumption, which is then used by our adaptive planner, e.g., for changing the plan to visit charging stations when battery power runs low.

ABOUT THE AUTHORS

JONATHAN ALDRICH is a professor in the School of Computer Science at Carnegie Mellon University, Pittsburgh. His research interests include improving software quality and programming productivity by expressing design within source code. Aldrich received a Ph.D. in computer science and engineering from the University of Washington, Seattle. Contact him at jonathan.aldrich@cs.cmu.edu.



DAVID GARLAN is a professor and associate dean in the School of Computer Science at Carnegie Mellon University, Pittsburgh. His research interests include autonomous and self-adaptive systems, software architecture, formal methods, explainablity, and cyberphysical systems. Garlan received a Ph.D. in computer science from Carnegie Mellon University. Contact him at garlan@cs.cmu.edu.

CHRISTIAN KÄSTNER is an associate professor in the School of Computer Science at Carnegie Mellon University, Pittsburgh. His research interests include the limits of modularity and complexity caused by variability in software systems. Kästner received a Ph.D. in computer science from the University of Magdeburg, Germany. Contact him at kaestner@cs.cmu.edu.

CLAIRE LE GOUES is an assistant professor of computer science in the Institute for Software Research at Carnegie Mellon University, Pittsburgh. Her research interests include automatically reasoning about and improving software quality in real-world, evolving systems. Le Goues received a Ph.D. in computer science from the University of Virginia, Charlottesville. Contact her at clegoues@cs.cmu.edu.

ANAHITA MOHSENI-KABIR is a robotics Ph.D. student at Carnegie Mellon University, Pittsburgh. Her research interests include robotics and artificial intelligence with a focus on robot learning through interaction with people and the environment. Mohseni-Kabir received her master's degree in computer science from Worcester Polytechnic Institute, Massachusetts, in 2015. Contact her at anahitam@andrew.cmu.edu.

IVAN RUCHKIN is a software engineering Ph.D. student in the Institute for Software Research at Carnegie Mellon University, Pittsburgh. His research interests include the modeling and verification of cyberphysical systems. Contact him at iruchkin@cs.cmu.edu.

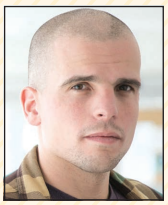
2. An upgraded software component is installed, but it has (seeded, in our simulation) defects that adversely affect behavior. We use our code-level adaptation engine to produce patches that enable the robot to complete its task with the new software component.
3. Environmental changes, such as a dark corridor or a blocked path, force the robot to alter its plan. We combine task adaptations (e.g., taking an alternate path)

ABOUT THE AUTHORS

SELVA SAMUEL is a software engineering Ph.D. student in the Institute for Software Research at Carnegie Mellon University, Pittsburgh. His research interests include programming languages and software engineering. Contact him at ssamuel@cs.cmu.edu.

MANUELA VELOSO is the Herbert A. Simon University professor in the School of Computer Science at Carnegie Mellon University, Pittsburgh, currently on leave at J.P. Morgan AI Reserach. Her research interests include artificial intelligence and robotics. Veloso received a Ph.D. in computer science from Carnegie Mellon University. Contact her at mmv@cs.cmu.edu.

BRADLEY SCHMERL is a principal systems scientist in the Institute for Software Research at Carnegie Mellon University, Pittsburgh. His research interests include software architecture, self-adaptive systems, and software engineering tools. Schmerl received a Ph.D. in computer science from Flinders University, Adelaide, Australia. Contact him at schmerl@cs.cmu.edu.



IAN VOYSEY is a research programmer in the Institute for Software Research at Carnegie Mellon University, Pittsburgh. His research interests include type theory and theorem proving. Contact him at iev@cs.cmu.edu.

CHRISTOPHER TIMPERLEY is a systems scientist in the Institute for Software Research at Carnegie Mellon University, Pittsburgh. His research interests include automated program repair, cyberphysical systems, program analysis, and search-based software engineering. Timperly received a Ph.D. in computer science from the University of York, United Kingdom. Contact him at ctimperley@cmu.edu.

JOYDEEP BISWAS is an assistant professor in the College of Information and Computer Sciences at the University of Massachusetts, Amherst. His research interests include robot perception, motion planning, control systems, artificial intelligence, and deployed robot systems. Biswas received a Ph.D. in computer science from Carnegie Mellon University, Pittsburgh. Contact him at joydeepb@cs.umass.edu.

with architectural adaptations (e.g., using different sensors and algorithms that operate better in the new environment) to enable the robot to succeed in its task.

Figure 3 shows the results of our evaluation. In each scenario, our adaptations were able to restore full functionality in response to perturbations in most tests.

Mobile robotics systems have the potential to play a greater role in society, assisting with tasks in the workplace and the home. But this revolution can

ABOUT THE AUTHORS

ARJUN GUHA is an assistant professor in the College of Information and Computer Sciences at the University of Massachusetts, Amherst. His research interests include programming languages, formal methods, and systems. Guha received a Ph.D. in computer science from Brown University, Providence, Rhode Island. Contact him at arjun@cs.umass.edu.

JARRETT HOLTZ is a computer science Ph.D. student at the University of Massachusetts, Amherst. His research interests include robotics, computer vision, artificial intelligence, and software engineering. Contact him at jaholtz@cs.umass.edu.

JAVIER CÁMARA is a lecturer in the Department of Computer Science at the University of York, United Kingdom. His research interests include autonomous and self-adaptive systems, software architecture, formal methods, and cyberphysical systems. Cámara received a Ph.D. in computer science from the University of Málaga, Spain. Contact him at javier.camaramoreno@york.ac.uk.

POOYAN JAMSHIDI is an assistant professor at the University of South Carolina, Columbia. His research interests include software engineering, systems, and machine learning, with a focus on the areas of machine-learning systems. Jamshidi received a Ph.D. from Dublin City University, Ireland. Contact him at pjamshid@cse.sc.edu.

only take place if the robots can easily adapt to new environments and maintain themselves with little human effort. Our preliminary results suggest that models are an important technical enabler for the kind of adaptation that will enhance the benefits of mobile robotics systems for society.

References

1. N. Bencomo, R. France, B. H. C. Cheng, U. Aßmann, Eds., *Models@run.time: Foundations, Applications, and Roadmaps*. New York: Springer-Verlag, 2014.
2. P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," in *Proc. IEEE /ACM Int. Conf. Automated Software Engineering*, 2011, pp. 497–508.
3. J. Holtz, A. Guha, and J. Biswas, "Interactive robot transition repair with SMT," in *Proc. Int. Joint Conf. Artificial Intelligence*, 2018, pp. 4905–4911.
4. M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. Int. Conf. Computer Aided Verification*, 2011, pp. 585–591.
5. D. Melicher, Y. Shi, A. Potanin, and J. Aldrich, "A capability-based module system for authority control," in *Proc. European Conf. Object-Oriented Programming*, 2017, pp. 20:1–20:27.
6. A. Mohseni-Kabir and M. Veloso, "Robot task interruption by learning to switch among multiple models," in *Proc. Int. Joint Conf. Artificial Intelligence*, 2018, pp. 4943–4949.
7. I. Ruchkin, J. Sunshine, G. Iraci, B. Schmerl, and D. Garlan, "IPL: An integrated property language for multi-model cyber-physical systems," in *Proc. Int. Symp. Formal Methods*, 2018, pp. 165–184.
8. C. S. Timperley, A. Afzal, D. S. Katz, J. M. Hernandez, and C. Le Goues, "Crashing simulated planes is cheap: Can simulation detect robotics bugs early?" in *Proc. IEEE Int. Conf. Software Testing, Validation and Verification*, 2018, pp. 331–342.
9. PRISM Model Checker, University of Oxford, 2011. [Online]. Available: http://www.prismmodelchecker.org/