Challenges in Physical Modeling for Adaptation of Cyber-Physical Systems

Ivan Ruchkin*, Selva Samuel*, Bradley Schmerl*, Amanda Rico†, and David Garlan*
Institute for Software Research, Carnegie Mellon University, Pittbsurgh, PA, USA
{iruchkin,ssamuel,schmerl,garlan}@cs.cmu.edu †arico@carthage.edu

***Abstract*—Cyber-physical systems (CPSs) mix software, hardware, and physical aspects with equal importance. Typically, the use of models of such systems during run time has concentrated only on managing and controlling the cyber (software) aspects. However, to fully realize the goals of a CPS, physical models too have to be treated as first-class models. This approach gives rise to three main challenges: (a) identifying and integrating physical and software models with different characteristics and semantics; (b) obtaining instances of physical models at a suitable level of abstraction for adaptation; and (c) using and adapting physical models to control CPSs. In this position paper, we elaborate on these three challenges and describe our vision of making physical models first-class entities in adaptation. We illustrate this vision in the context of power adaptation for a service robotic system.**

I. Introduction

State-of-the-art self-adaptation relies on multiple layers of models at run time to enable sophisticated decision-making. It is common to separate the system under adaptation from the adaptation layers that govern control of increasingly higher-level concerns, from specific adaptive strategies like adding a server, to managing long-term goals of the system, like user satisfaction or mission achievement. In particular, models of the software being adapted are used to reason about and manage software-intensive systems. Common models used include architecture models [1]–[5], goal and requirements models [6], [7], and models of tasks [8].

Existing self-adaptive approaches predominantly use sophisticated models of the software, but consider physical characteristics of the system more simply, and mostly only insofar as modeling resources are available to the software. In architecture models, for instance, physical elements may be represented as components (e.g., a motor or a gripper component [9]), making adaptation reasoning discrete and symbolic in most cases.

Cyber-physical systems (CPSs) are characterized by the inclusion of physical phenomena and interactions, such as power and timing in addition to complex software models, which makes adaptation difficult. Compared to traditional control systems, CPSs act in more open and uncertain contexts, and are managed at run time by increasingly distributed, autonomous, and large-scale software parts. If physical dynamics are not accounted for properly in such complex systems, adaptive models may fail to accurately predict the outcome of adaptation, making them less effective than non-adaptive systems. For example, if the software of a robot is reconfigured to get better accuracy in localization without considering physical characteristics like increased power consumption of the hardware, then the robot may not have enough power to achieve its task.

Fully managing CPSs requires run-time reasoning about models of all parts of the system: the cyber (software) *and* the physical models, and combinations of both. When incorporating physical models into adaptation, we face several challenges. First, we need to pick appropriate formal representations for the physical models. The models need to be sufficiently expressive to represent the target behavior of the system and the environment. Different formalism choices may enable or disable certain kinds of analyses and affect possible adaptations. Another challenge is obtaining actual instances of physical models. They cannot be obtained simply by abstraction of software or by design (as is the case for cyber models) — often they require experimentation and data collection to build since we may have no direct control over physical processes, such as urban mobility [10]. The final challenge is incorporating all kinds of models into the adaptation process. Many physical qualities (e.g., power) have a systemic (i.e., cross-cutting) nature and can apply to many adaptation decisions about both cyber and physical aspects. Therefore, an important choice in the design of self-adaptive cyber-physical systems is which adaptation components (e.g., analysis algorithms and execution actors) can manipulate which physical models and how the different control loops for these models can be integrated.

In this paper, we survey existing work on model-based self-adaptation for CPS, focusing on work in the self-adaptive robotics domain, in Section II. In Section III we illustrate several of the challenges we will discuss with an example that is based on our work of building a power model for a service robot in order to add power-aware self-adaptation. Finally, Section IV contains a detailed discussion of the challenges associated with using physical models in combination with software models in CPS.

II. Related Work

Models at run time have been mostly used to guide self-adaptation or verification. The self-adaptive community has focused on various models of the *software* of the system that is being adapted. Such models include requirements, goals, architecture, and tests.

978-1-5090-4130-5/16/$31.00

There has been an abundance of work on applying self-adaptive techniques to robotics systems, which have physical characteristics such as batteries, physical movement, interaction with humans, etc. The works of Sykes et al. [3], [8], [9] include examples from the domain of robotics, but the models that are used at run time focus on the architecture or structure (both behavioral and extra-functional) of the system and its tasks. In most cases physical models are limited to simple models of resources and environment. Many of the approaches involve the use of multiple models. They represent and manipulate models of tasks or policies (e.g., [11]) of a robot, in addition to the software architecture. However, none of these models are purely physical models.

The most sophisticated model of the physical aspects of the robots are discussed in [9], which describes an approach for updating environment models of a robot that has software organized according to Gat's three-layer architecture [12]. The reasons for which an environment model might have to be updated include unmodeled conditions, probabilistic factors, unmodeled transitions, and unmodeled historical dependencies, which manifest themselves as failure of the robot to reach its goal. The model is encoded as a logic program that describes the actions that can be taken when certain conditions hold. To achieve a goal, this model is used to generate a plan. As the robot executes different plans, it collects execution traces for each of them. However, the environment model, while representing some physical aspects, serves the analysis of the software models.

Physical models are used in [13] to reason about optimal configurations of software in an unmanned underwater vehicle (UUV). In this work, the software configuration has to be sensitive to both the speed of the UUV and the amount of power that sensors can use, while fulfilling requirements on the number of measurements that should be taken during a mission. They use run-time quantitative verification [14] to verify that quality attributes are being met and to choose appropriate reconfigurations if not. The quality attributes include physical characteristics (such as power consumption).

None of the approaches describe a methodology or framework for treating physical models as first-class alongside software models in the adaptation process.

III. Motivating Example

We will illustrate physical modeling challenges on the example of TurtleBot[1] – a hardware platform for a service robot whose software can be customized to make the robot perform various tasks such as indoor delivery, notification and escorting people. TurtleBot is equipped with a Kinect depth camera to visually sense its surroundings, navigate, and avoid collisions with obstacles in its path. Its control software is based on the Robot Operating System (ROS) [2], which in turn executes on an on-board computer – the Intel Next Unit of Computing (NUC) [3] – that runs a Linux OS. TurtleBot moves using a mobile base with a two-wheeled differential drive system that is capable of driving in straight lines and arcs, and turning in place. The base also houses a battery that powers all devices.

Since it runs on a battery, TurtleBot has a limited amount of power and thus cannot perform tasks of indefinite duration. Therefore, it can run out of power and shut down in the middle of a task, which is undesirable. To prevent this, offline and online power adaptation is necessary. The former would determine, before starting a task, the system configuration and an action plan that would satisfy energy constraints (e.g., not running out of power in the middle of the task). The latter would react to unanticipated changes in-progress (e.g., if driving drains more energy than expected) and would change the configuration or the plan.

To enable power-aware adaptation for TurtleBot, two power models are necessary:

- A *power consumption model* that predicts how much energy it takes to accomplish a task.
- A *power state estimation model* that determines for a given battery voltage reading how much energy remains in the battery.

Building a power consumption model requires knowledge of how much power each physical part of TurtleBot uses for a given task. From the perspective of power usage, TurtleBot has three components: a base, a Kinect, and a NUC. The power consumption of the base depends on the motion type (driving straight and rotating drain different amounts of power), velocity (translational and rotational), the floor material (e.g., carpet or tile), and the slope. NUC power consumption depends on the amount of processing that the NUC does, which is determined by the parameters of vision and navigation algorithms (e.g, number of processing points in a sensed set of points provided by Kinect) as well as other software-based computations, such as interactions with users. Power consumption of the Kinect sensor is mostly fixed since it delivers fixed data.

Consider the following scenario that we will use to illustrate the challenges: TurtleBot needs to navigate to a target location inside a building. Two paths are available: a shorter path through a crowded hallway, and a longer path through a mostly empty hallway. Using the shorter path would lead to the robot driving with slower speed and potentially stopping to let humans clear the path, whereas the longer path can be driven with a faster speed and fewer, if any, interruptions. Depending on the specific parameters of this scenario, either of the two paths may turn out to be

[1] http://turtlebot.com

[2] http://www.ros.org/

[3] http://www.intel.com/content/www/us/en/nuc/overview.html

energy-optimal and (independently) time-optimal, and these physical factors need to be considered when choosing a path.

In this motivating example, there are both cyber and physical aspects of adaptation. In particular, power balance depends on the mechanical characteristics of the robot and its environment, the chemical characteristics of the battery, the electrical characteristics of computing hardware, and computing complexity of software. These dependencies make TurtleBot a convenient illustration for challenges in physical modeling.

IV. Challenges

Our position is that all models (physical and cyber) should be treated equally in the running system for verification and adaptation. By *physical models*, we mean abstractions of physical objects and interactions, such as those based on mass, electricity, and other physical phenomena. Physical models may include relevant physical devices. For instance, a kinematic model of the robot from Section III would include attributes of its wheels. By *cyber (software) models*, we mean representations of software and hardware structures, data, and computations, such as nodes and event buses in ROS.

Existing techniques that use models during run time tend to favor software models with limited representation of physical phenomena. However, physical models also need to be defined, monitored, and manipulated in ways similar to software models in order to make CPSs adapt better. Also, physical models and their analysis need to be integrated with software models to provide a holistic view of the run-time state of the cyber-physical system.

A number of challenges need to be addressed to achieve this vision. First, different modeling characteristics of software and physical domains need to be considered for identifying and integrating the required models. Second, uncertainty and evolution of physical properties over time should be considered for deriving and defining the physical models. Third, physical and software models need to be adapted and considered in any run-time adaptation decisions that need to be made on the system.

A. Selecting Modeling Formalism

Modern engineering methods for cyber-physical systems offer a multitude of possible formalisms for physical modeling, with varying mathematical foundations. Some are based on differential equations (e.g., Modelica[4]). These are often used to describe continuous-time dynamical systems (for instance, a physical movement). Signal-flow models (e.g., Simulink[5]) can be used to approximate and simulate differential equations. For systems with multiple modes, one can use variations of automata, such as timed, probabilistic, and hybrid automata [15].

Not all formalisms, however, are equally useful to model every physical aspect of a system. The chosen formalism needs to fit the purpose and context in which it is used.

One of the factors that affects the choice of formalism is its expressiveness, which determines what objects and dynamics can be described. For example, if power predictions for a wheeled robot should output real numbers, formalisms that only use integers (e.g., Promela[6]) would not fit this purpose. In such situations using an inappropriate formalism would make models overly coarse, leading to shallow analysis of the system. For physical models, important formalism properties include linearity, treatment of continuity, and the underlying mathematical theories (which constrain permitted functions and quantities). For instance, if a formalism is restricted to polynomial functions (e.g., the input language of KeYmaera [16]), it may be challenging or even impossible to accurately describe dynamics that are traditionally modeled with transcendental functions (such as movement along a curve modeled using trigonometric functions).

Merely modeling a physical system is rarely the final goal; rather, one usually aims to analyze a model to provide some value in the engineering process. For the power model of TurtleBot, the goal of analysis is to predict whether the robot has enough energy to complete a given task. That is an example of an online analysis, but many analyses are done offline. For instance, determining the center of mass of a physical structure is a common offline analysis. Formalism choice is often dominated by the tradeoff between expressiveness and feasible analysis: one aims to use the least expressive formalism possible, provided that it captures the relevant phenomena. Otherwise, if the formalism's expressiveness is too high, analysis may be infeasible or the complexity will make the analysis prohibitive to perform.

In addition to the requirement of different formalisms for different physical aspects, we also face the challenge that many physical aspects require special expertise to model. For example, modeling the mechanical aspects of the TurtleBot requires understanding the mechanics of a rotating wheel, whereas modeling power storage and consumption requires electrical engineering expertise. As cyber-physical systems often incorporate different aspects of physicality, the likelihood that there is a domain expert who understands all the modeling characteristics of the system is low.

The challenges of expressiveness, analytic power/cost, and domain diversity lead us to the following position:

Position 1. *Multiple formalisms are needed to cover all aspects of a CPS. Developing a single unified one is a fruitless exercise.*

Unified modeling formalisms range from strict prescriptive semantics (which may end up being too restrictive and not scale for the whole system, like state automata) to

[4]https://www.modelica.org

[5]https://www.mathworks.com/products/simulink

[6]http://spinroot.com

informal and disjoint semantics (which often do not permit comprehensive automated reasoning). One example of the latter case is the Unified Modeling Language (UML)[7]. It provides several disconnected languages (statecharts, use case diagrams, class diagrams, activity diagrams) that, even for purely software systems, have proven challenging to formally unify and verify. For CPS, we have little hope to find a single such formalism.

Instead, our latest research has been exploring an approach that allows engineers to work within the formalism required for their domain and within their area of expertise, and to integrate those models through abstraction and relation. Abstraction factors out crucial commonalities of models into a simplified space (in a metamodel language, a universal format for all models traces, etc.). Once that space is constructed, some elements of models are related through it. That relation is checked for consistency properties on the structure [17] and semantics [18] of the constituent models. Our vision is that language designers create these checking mechanisms to assist engineers in putting models together.

Furthermore, beyond integrating the models through abstraction and relation, different methods for analyzing models needs to be coordinated. We need to ensure that an analysis is never applied to a model that violates the assumptions of the analysis. Since such violation can originate from updating a model by another analysis, analyses must be executed in the correct order. In [19] we developed an analysis integration approach that uses contracts (assumptions required for the analysis to be valid, and guarantees provided by the analysis) to specify dependencies between analyses, determine their correct order of application, and specify and verify applicability conditions in multiple domains. In [20] we describe some of the ways in which we can deal with the inevitable dependency loops that occur in complex cyber-physical systems with many models and analyses.

Because of the complex dynamics of CPS and uncertainty of operating environments, these models and relationships need to be maintained at run time, as we discuss in the coming sections. But first we discuss how models are built with adaptation in mind.

B. Obtaining Physical Models

Once formalisms for modeling various aspects of a system have been chosen, they are used to build models for the specific system at hand. To use a physical model successfully at run time, the model has to rank well enough on several *model value factors* – characteristics of a model that determine how much value it provides in run-time adaptation. The concrete factors will vary by system, but we can distinguish three general categories of model value factors:

1) *Analytical power* – the capacity of the model to provide conclusions (estimates, predictions) with required precision, granularity, and horizon. Analytical power is in part determined by the formalism, and partially by the model itself. For instance, a power model learned from a single run of TurtleBot would not accurately predict another run's power consumption.
2) *Fragility* – dependency of the model's analytical power on the model's assumptions, and difficulty of carrying over the model to a new context from its original context. For example, if a power model does not easily carry over from one surface material (e.g., carpet) to another (e.g., tiles), it is said to be fragile with respect to surface material. If a model is to be used across multiple contexts, its fragility needs to be addressed and managed.
3) *Computational cost* – the amount of computation needed to perform the required analyses. Again, although a formalism sets the overall expectations of computational costs, the model itself may contribute to them. For example, a state model with redundant states may slow down its exploration by a model checker. Computational costs often correlate with time needed for analysis, which can be critical during a self-adaptive system's execution if it is important to react quickly.

Ideally, requirements on these value factors would determine the approach to building a physical model. Unlike cyber models, which are often created through abstraction and (re-)engineering the modeled software, physical models may be built based on a theory or raw data:

- *Theory-driven* models: the phenomena are modeled using an appropriate physical theory. For example, a robot wheel may be modeled mechanically to derive the power it takes to move [21]. Theory-driven models may be calibrated to suit the target context, but the primary emphasis is on the theoretical abstractions.
- *Data-driven* models: the phenomena are modeled indirectly, by collecting data and creating (e.g., statistical) models of this data. Data-driven models do not necessarily use or reflect the physical theories about reality. To build a data-driven model for TurtleBot power consumption, one could run multiple experiments with repetitive movement tasks (e.g., spinning on the floor for 15 minutes) and use regression to extract the model from the experimental data [22].

It is challenging to make model building decisions to align model value factors with the requirements. Before a theoretical model is built or data is collected, it is difficult to know the margins of error and prediction horizons that a model can tolerate. It is also hard to set an expectation of how a model would respond to context switches.

Position 2. *For physical models, approaches to building models significantly affect the resulting value of the model.*

[7] http://www.uml.org

Therefore, model-building decisions need to be carefully considered in the light of model value factors. There is little formal guidance on making model-building decisions in this way, and more of such guidance is needed.

In the case of robot power modeling, we examined the implications of the two model-building approaches on the power model value factors. We concluded that predictions over a long future horizon (up to an hour of operation) with relatively low precision (acceptable error up to 10%) can be accomplished by a data-driven model. A theory-driven model, on the other hand, could have more precise predictions (on the order of minutes), but its error would accumulate for long horizons. Thus, a data-driven model was preferable from the perspective of analytical power.

The fragility of our data-driven power model, however, left much to be desired: we cannot predict power consumption in new contexts (e.g., a different surface material) without additional data collection. However, we can reuse the method and equipment we used to collect data originally to rapidly extend models as needed.

We chose to pursue a data-driven power model for TurtleBot, influenced by the fact that we had no expertise in electrical, mechanical, and chemical engineering required to build theory-driven models. We collected power consumption data from multiple motor tasks from executions of each motion task up to 30 minutes. Then, we removed the outliers in the data and built the model using linear regression. Although the result was satisfactory, more guidance on how model building decisions affect model value would have helped streamline the decisions and perhaps build a higher-precision model. Having systematic processes for guiding model choice will be crucial going forward, especially as the size and complexity of these systems increases.

C. Using Physical Models in Adaptation

The constructed physical models may be used in three ways to make adaptation decisions in the target system. First, and most commonly, they might be used to *estimate* the current state of the system. For example, there are no on-board devices on the TurtleBot to measure the remaining battery capacity directly. But, the output voltage from the battery can be measured, and giving an estimate of the remaining battery capacity. Second, physical models can be used to *predict* impacts of physical actions. Third, the models might be used to *search* for an adaptation strategy that satisfies the given constraints. In this case, the search needs to be sensitive to the relationships between the different models.

Aside from adapting the system, physical models can *themselves* be updated because of changing physical conditions. For instance, if the power model is consistently making overly optimistic predictions, it may be possible that wear and tear on the robot has reduced the efficiency of movement. There are multiple ways to respond to this situation: (re-)learning, calibration, updating ontology, and so on. All of these adapt the model itself without changing the system unlike classical software-based adaptation.

Thus, using physical models in adaptation is a two-part challenge:

- How to organize decision-making to adapt the system based on cooperative use of its models?
- How to organize decision-making to adapt models based on their model value (as described in Section IV-B)?

This challenge leads us to the following position:

Position 3. *Physical models should be treated as first-class entities in adaptation, and thus given equal importance as cyber models. The value of each physical model needs to be tracked throughout the system execution, and these models should be adapted as needed.*

The way we address this position is by giving the system self-awareness of some aspects of model-building, particularly model value factors. Since the system needs to manipulate its models, it needs to operate proactively under the same concerns as do the engineers that built the original version of the model. In TurtleBot's case, the robot should continuously monitor how good its model-based predictions are and evaluate the precision of the models, adapting when the precision becomes unacceptable — before it finds itself with too little power to proceed. When possible, this evaluation should take known causal connections into account (e.g., wear and tear that reduces motion efficiency). It is, however, not advisable to improve the models without sufficient evidence due to the risk of overfitting.

Our vision of making physical models first-class can be implemented in various ways. On the one hand, physical aspects can be explicitly embedded into existing software models; for instance, annotating architectural views with physical quantities [17]. This approach, however, is subject to expressiveness limitations described in Section IV-A. On the other hand, it is often convenient to construct separate physical models to have more freedom in model-building choices described in Section IV-B. But then more effort needs to be spent on relating these separate physical models to cyber models. Either way, physical modeling needs to be as involved in adaptive processes as cyber modeling.

V. Conclusion

Physical models are as important as cyber models when it comes to model-based adaptation of cyber-physical systems. However, we face several significant challenges in using physical models in adaptation. One needs to carefully select a formalism. The method of obtaining the model influences the scope of its applicability and the analyses that can be run on the model, as well as their cost. As our experience with power modeling suggests, sophisticated experimental protocols can help collect and abstract data properly. Guide-

lines for selecting formalisms and picking experiments are needed to help develop custom physical models for CPS.

Another set of challenges is associated with *using* physical models in adaptation. Contextual sensitivity, information hiding, and maintaining models at run time are among the prominent barriers in adaptation using physical models. Depending on the type of model, different model qualities need to be prioritized for successful adaptation.

We thus come to a conclusion that modern model-based self-adaptive systems lack the ability to adequately treat physical phenomena. To improve the state-of-the-art of self-adaptive CPS, we need new approaches that focus on explicitly bringing together cyber and physical models, by taking into account their characteristics and features. We also envision tools and environments that help construct, relate, and maintain heterogeneous CPS models more systematically, instead of having tools for each kind of model.

ACKNOWLEDGMENTS

The authors would like to thank Rick Goldstein, Joydeep Biswas, and Manuela Veloso for their assistance with setting up and operating TurtleBot technology. This material is based on research sponsored by NSF under Grant No. 1560137 and by AFRL and DARPA under agreement number FA8750-16-2-0042. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation therein. The views, opinions, findings, conclusions, and recommendations contained herein are those of authors and should not be interpreted as necessarily representing or reflecting the official policies or endorsements, either expressed or implied, of NSF, AFRL, DARPA, of the U.S. Government.

REFERENCES

[1] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Architecture-based Runtime Software Evolution," in *Proc. of the 28th Int. Conf. on Software Engineering*, Kyoto, Japan, 1998.

[2] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based Self-adaptation in the Presence of Multiple Objectives," in *Proc. of the 2006 Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Shanghai, China, 2006.

[3] D. Sykes, W. Heaven, J. Magee, and J. Kramer, "Plan-directed Architectural Change for Autonomous Systems," in *Proc. of the 2007 Conf. on Specification and Verification of Component-based Systems (SAVCBS '07)*, Dubrovnik, Croatia, 2007.

[4] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models@Run.Time to Support Dynamic Adaptation," *Computer*, vol. 42, no. 10, Oct. 2009.

[5] N. Huber, A. Hoorn, A. Koziolek, F. Brosig, and S. Kounev, "Modeling Run-time Adaptation at the System Architecture Level in Dynamic Service-oriented Environments," *Serv. Oriented Comput. Appl.*, vol. 8, no. 1, Mar. 2014.

[6] V. E. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos, "Requirements-driven Software Evolution," *Comput. Sci.*, vol. 28, no. 4, Nov. 2013.

[7] N. Bencomo, A. Belaggoun, and V. Issarny, "Dynamic Decision Networks for Decision-making in Self-adaptive Systems: A Case Study," in *Proc. of the 8th Symp. on Software Engineering for Adaptive and Self-Managing Systems*, San Francisco, CA, USA, 2013.

[8] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, and K. Inoue, "Learning Revised Models for Planning in Adaptive Systems," in *Proc. of the 35th Int. Conf. on Software Engineering*, San Francisco, CA, USA, 2013.

[9] V. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, and S. Uchitel, "MORPH: A Reference Architecture for Configuration and Behaviour Self-adaptation," in *Proc. of the 1st Int. Workshop on Control Theory for Software Engineering (CTSE 2015)*, Bergamo, Italy, 2015.

[10] D. Zhang, J. Zhao, F. Zhang, and T. He, "UrbanCPS: a cyber-physical system based on multi-source big infrastructure data for heterogeneous model integration," in *Proc. of the ACM/IEEE 6th Int. Conf. on Cyber-Physical Systems*, 2015.

[11] J. C. Georgas and R. N. Taylor, "Policy-based Self-adaptive Architectures: A Feasibility Study in the Robotics Domain," in *Proc. of the 2006 Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Leipzig, Germany, 2008.

[12] E. Gat, "Three-layer Architectures," in *Artificial Intelligence and Mobile Robots*. Cambridge, MA, USA: MIT Press, 1998.

[13] S. Gerasimou, R. Calinescu, and A. Banks, "Efficient Runtime Quantitative Verification Using Caching, Lookahead, and Nearly-optimal Reconfiguration," in *Proc. of the 9th Symp. on Software Engineering for Adaptive and Self-Managing Systems*, Hyderabad, India, 2014.

[14] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive Software Needs Quantitative Verification at Runtime," *Commun. ACM*, vol. 55, no. 9, Sep. 2012.

[15] R. Alur, *Principles of Cyber-Physical Systems*. Cambridge, Massachusetts: The MIT Press, Apr. 2015.

[16] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems," in *Proc. of the Int. Conf. on Automated Deduction*, Berlin, Germany, 2015.

[17] A. Y. Bhave, B. Krogh, D. Garlan, and B. Schmerl, "View Consistency in Architectures for Cyber-Physical Systems," in *Proc. of the 2nd ACM/IEEE Int. Conf. on Cyber-Physical Systems*, Apr. 2011.

[18] A. Rajhans, A. Y. Bhave, I. Ruchkin, B. Krogh, D. Garlan, A. Platzer, and B. Schmerl, "Supporting Heterogeneity in Cyber-Physical Systems Architectures," *IEEE Tran. on Automatic Control*, vol. 59, no. 12, Dec. 2014.

[19] I. Ruchkin, D. de Niz, S. Chaki, and D. Garlan, "Contract-Based Integration of Cyber-Physical Analyses," in *Proc. of the 14th Int. Conf. on Embedded Software (EMSOFT '14)*, New Delhi, India, Oct. 2014.

[20] I. Ruchkin, B. Schmerl, and D. Garlan, "Analytic Dependency Loops in Architectural Models of Cyber-Physical Systems," in *Proc. of the 8th Int. Workshop on Model-based Architecting of Cyber-Physical and Embedded Systems*, Ottawa, Canada, Sep. 2015.

[21] J. Morales, J. L. Martnez, A. Mandow, A. Pequeo-Boter, and A. Garca-Cerezo, "Simplified power consumption modeling and identification for wheeled skid-steer robotic vehicles on hard horizontal ground," in *2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2010.

[22] A. Rico, I. Ruchkin, B. Schmerl, and D. Garlan, "Hardware Power Modeling for Turtlebot," *ResearchGate*, Jul. 2016. [Online]. Available: https://www.researchgate.net/publication/306274755_Hardware_Power_Modeling_for_Turtlebot