A Policy Based Approach to Models at Runtime within Cyber-Physical Systems: A Case Study in Microgrids

Mark Allison and Anil Kuvvarapu
Department of Computer Science, Engineering and Physics, University of Michigan-Flint, Michigan, USA
Email: {markalli, sjleeum, akuvvara}@umflint.edu

Seung-Jin Lee
Department of Earth and Resource Science, University of Michigan-Flint, Michigan, USA
Email: sjleeum@umflint.edu

Abstract—As physical systems which are managed using software become more intricate, the use of models as high level abstractions have been increasingly applied to tame the complexity required of the controlling software. Model based software control systems typically execute models at runtime to produce or inhibit behavior within the underlying system. One category, interpreted domain-specific modeling languages (i-DSMLs) derive their semantics using changes to a causally connected runtime model. These languages have demonstrated effectiveness in the control of cyber-physical systems, however the dynamic nature of their runtime models constrains the range and depth of behaviors that may be specified. This article presents an approach to model-based control of cyber-physical systems which utilize prioritized Event-Condition-Action policies as an augmentation. We address policy conflict management and utilize demand side energy management as our application domain to support proof of principle and utility.

Index Terms—model-driven software development, microgrid, smart grid, policy management

I. INTRODUCTION AND MOTIVATION

An approach to taming complexity inherent within software systems is to place models in lieu of code during software development. Model-Driven Engineering (MDE) utilizes software models as first class artifacts [1]. The common application of models includes the automated transformation to a high level language which is subsequently compiled and executed to realize some behavior. Alternatively, models may be directly executed via a specialized interpreter. The focus of this discourse is the taxonomy of interpreted languages referred to as i-DSMLs. Languages which belong to the class derive their semantics through changes to models at runtime. These i-DSMLs have demonstrated utility in the control of cyber-physical systems [2], [3].

Heralded as the next generation of engineered systems, Cyber-Physical Systems (CPSs) are systems of systems that involve a symbiotic kinship between software and some physical system under its control. Towards ease of readability, the authors will borrow the term Plant from control theory to describe such physical systems. The economic and social impact of CPSs is responsible for the sudden rise in attention from academia, industry and government. As the complexity rises in the plant due to ever increasing functionality demands, novel approaches based on control-computing codesign are warranted. Our approach seeks to equip i-DSML models with a necessary framework to accommodate behavior definitions and higher level autonomic constructs.

The dynamic semantics of an i-DSML may be informally described as follows: The submission of a new user preference model triggers a comparison with the systems perception of the state of the plant. This is accomplished via a causally connected runtime model. Changes between the user’s preference and the current state of the plant are then interpreted based on domain-specific state transition systems to derive some relevant behavior in the plant.

A major limitation of this approach is the dynamic or lazy nature of this runtime model. The interpreter as a system will simply act to maintain stasis between the runtime model and the plant unless acted upon by a new user preference model. In the case of our domain under study, the microgrid, a runtime model which holds that an electric lamp should be in the on state will ensure that the lamp remains in that state unless a new model arrives which prefers the lamp off. The runtime model lacks that capability to specify under what circumstances or condition the lamp may change state; maybe we only want the lamp to be on during daytime. Having our user submitting models for each state change is counter to the autonomic goal of minimizing human interaction and the errors in consort.

In this paper we present an approach to augment adynamic i-DSML runtime models with low level policy...
constructs to derive reactive model behavior at runtime. By making these models reactive to the CPS environment, we allow for users to specify more granular behavior and support context aware decision making at runtime. Allison, et al. [2] detailed an i-DSML, MGridML which, through its interpreter, was capable of rudimentary microgrid control. This work progresses this previous work by addressing the aforementioned limitation and advances the research towards our objective of near autonomic plant control.

More specifically our contributions are:

- A policy based augmentation to an i-DSML metamodel.
- The outline of the execution semantics of the policy augmentation interpretation via semi-formal methods.
- A rudimentary methodology for policy conflict resolution within the context of i-DSMLs

The remainder of this paper is organized as follows: In the next section we explore background concepts critical to this work. Section 3 presents an indicative microgrid scenario with which we thread this article. Section 4 outlines the policy framework after which we present our related work and conclude in Sections 5 and 6 respectively.

II. BACKGROUND

Before we delve into specifics our approach it is prudent to first introduce the domain of discourse and the base technologies that form the platform for our research.

A. Demand-Side Energy Management: The Microgrid

The Microgrid is a semi-autonomous aggregation of microsources and loads capable of sustaining internal energy requirements. As an essential part of the Smart-Grid concept, the microgrid decentralizes power generation of the larger macrogrid by employing distributed energy resources (DERs) to increase reliability. DERs are smaller scale power sources that contribute to meet regular electricity demand. They may either be renewable technologies such as wind turbines, or storage elements such as batteries or plug-in electric vehicles (PEVs). Our models specifically segregate storage elements from other energy sources as they are capable of both load (charging) and source (discharging) states. A grid-tied microgrid connects to its utility source via a Point of Common Coupling (PCC). A Microgrid Central Controller (MGCC) is responsible for the balance of power sources to power consumption elements or loads. Plant elements are connected via physical controllers and sensors which provide the interface by which we address control theory bases of Measure, Compare, Compute, and Correct. This work investigates policy augmentation using the Microgrid Modeling Language (MGridML) [2].

B. Interpreted Domain-Specific Modeling Languages

One prominent aspect of MDE is the utilization of domain specific languages (DSLs) to produce its solution suite [4]. DSLs have focused expressiveness, and are typically designed to address a narrow scope as opposed to general purpose languages such as Java and C++. DSLs encapsulate domain knowledge and allows for users to conceptualize solutions using high level abstractions of concepts and entities from the problem domain [5]. Predominately DSLs are used to generate solutions by transforming models to a high level language which is subsequently executed. Another approach - one which concerns our framework - directly interprets models.

I-DSMLs are interpreted at runtime and derive their semantics from changes between subsequent models. We have demonstrated the use of this approach to effectively manage cyber-physical systems. The execution semantics of i-DSMLs are typically defined within their interpreters also known as domain-specific virtual machine (DSVMs). Fig. 1 illustrates the layered architecture of the MGridML interpreter. The topmost layer is the Microgrid User Interface which provides the model generation environment for users to specify system behavior. These models are then sent to the next layer called the Microgrid Synthesis Engine. The synthesis engine transforms the user’s models to executable control scripts utilizing as its key technology, an adaptable and causally connected runtime model of the plant. The Microgrid Control Middleware exists to execute these control scripts and ensure the delivery of services. The Microgrid Hardware Broker provides for an API to the plant control mechanisms. Our policy based approach primarily concerns the synthesis engine layer of the interpreter.

III. MICROGRID SCENARIO

We will thread the presentation of this approach with a simple representative scenario from the microgrid energy management domain. Our scenario begins with our actor, Dana, a homeowner of an alternative energy enhanced home located in windy Midwest USA. Dana’s home leverages renewable energy using an array of wind
powered turbines. As wind power is intermittent, this source is stabilized via a bank of batteries. This microgrid is controlled via a MGridVM, an i-DSML model based interpreter presented in section 2.2.

Unfortunately our scenario takes a turn for the worst when the utility to which Dana’s microgrid is connected suffers an outage. Since the current runtime model contains a static specification that all internal loads being supplied solely by the utility, everything goes dark; Dana’s microgrid is incapable of islanding itself without receiving an islanding model which tells the PCC to disconnect and the storage controller to put the batteries on-line. If only Dana’s system had the language capability of specifying models capable of responding to environmental change in real time.

IV. POLICY FRAMEWORK

The inclusion of policies within an existing language requires the augmentation of policy syntax and semantic essentials. The syntax of MGridML is defined using its metamodel (abstract syntax, concrete syntax and static semantics). The dynamic semantics of the language is embedded within the interpreter and describes the implication of models and model changes. This section will address how both aspects of the language affected by our policy framework.

A. Metamodel

Fig. 2 presents the ECA policy abstract syntax and how it integrates within the larger MGridML. A MGridML base model MGridSchema comprises a control structure MGridControlSchema and a data structure MGridDataSchema. This separates the description of a microgrid in terms of its configuration and devices properties. This is akin to the separation of control and data structures in procedural languages. By specifying the language in this manner we are able to increase our model change response rate as the data portion tends to change with more frequency [2]. Note that a MGrid policy is attached to the MGridController. This allows for additional semantics to be derived by policy placement within a model. ECA policies are well suited for systems of a reactionary nature [6]. Each policy is a finite set of Event-Condition-Action rules of the form:

\[
\text{if } \text{COND}_i \text{ : } \text{EVT}_i \rightarrow \langle \text{Act}_1, \text{Act}_2, \ldots , \text{Act}_n \rangle
\]

The above rule denotes that upon the occurrence of some event \( \text{EVT}_i \), the system shall perform an ordered sequence of actions \( \langle \text{act}_1, \ldots, \text{act}_n \rangle \) given a specified condition \( \text{COND}_i \). In the lower portion of Fig. 2 are the shaded metaclasses for our policy. We extend the basic notion of an ECA policy with application specific properties which enable us to contain the behaviors they are capable of describing and add prioritization to enable the resolution of conflicts. In its application to cyber-physical systems, events \( \text{EVT}_i \) denotes the \( i \)-th event to occur. Events are primitive and may be parameterized. An example event within our domain is \( \text{PCC:Util:State\_Change} \) which signifies that the utility feed has experienced some change.

Conditions in our context may either be atomic or composite. Composition operations are enumerated AND, OR and XOR. We revisited our Feature Oriented Domain Analysis (FODA) and we are sufficiently satisfied that these operations are powerful enough to describe typical scenarios of the domain. Per our abstract syntax we have also curtailed the maximum number of atomic conditions which comprises a composite condition to two. Since the possible number of states to address when addressing \( n \) elements is \( 2^n \), then by limiting the number of elements that may be addressed within each policy then each policy to a maximum of four rules. At this stage of the research, users will be creating these policies. Cursory user surveys have seen that beyond four rules (\( 2^3 = 8 \)) users become overwhelmed with the task.

SCENARIO: Fig. 3 shows Dana’s islanding scenario reified as a policy. Note that this policy is attached to a StorageController STC001; the target of Rules 00, 01, 10 actions. Policy POL001 is based on state changes in the utility and battery charge. More specifically, the policy considers whether or not the utility is experiencing a
failure and if the storage, STC001, is above some lower threshold. Discharging some storage devices beyond their limit may result in permanent damage. The actions attributed to the four rules are in the form of control scripts which are sent to the Microgrid Hardware Broker layer for realization.

One critical element of the static semantics of the policy metamodel requires that no rule may have the same triggering event and contain an action which directly counters another. We will expand on the notion of counter actions in our discussion of static conflict resolution.

B. Policy Execution Semantics

Fig. 4 presents an overview of the policy interpretation. Prior to a new model being executed, the interpreter performs a static check to determine if any rule is in conflict. A conflict occurs whenever events trigger policy actions which should not occur simultaneously [7]. Since we address the limited scope of a domain with a finite set of control scripts, we are able to classify them according to their counterparts. For example setting a battery to charge and to discharge are classified as control scripts which are counter to each other.

An event is checked against the three tables and applicable rules are fired and placed into active status if no conflicts are detected. Should a conflict arise then that with a higher priority supersedes. Conditions are checked using the causal run-time model of the SE model and not the physical system (latency issues). On one early prototype there was an enormous amount of events generated from the plant. Event changes which do not apply to any rule are via a check in table simply ignored by the policy manager; they are not rechecked. An event however may trigger an update to the current run-time model which will persist its effect.

V. RELATED WORK

Several works have proposed the use of models to build cyber-physical systems [8]-[10]. Our approach relies on an adaptable runtime model and applies policies to enable environment reactive behavior.

To the extent of our systematic literature review we have only been able to identify one approach to integrate policies within i-DSMLs at the language metamodel level. Hernandez and Clarke [11] proposed a signature based approach to compose policy constructs within DSMLs. While this approach utilizes the ECA paradigm, it differs as it proposes to use a weave model to integrate policies in a generic manner. Our approach policies as a composition of rules and furthermore extends to a resolution methodology.

Shankar, et al. [12] proposes a similar approach to policy-based management by extending the Event-Condition-Action rules to incorporate post-condition as additional semantic information. This extension is used to detect conflicts. Conflicts detected are resolved using meta-rules which specify desired system states. Our approach differs by its application to models and our rudimentary policy ratification. Since it is our intent to overlay these primitive policies constructs with higher level autonomic principles our design intent is for a simple robust distinction of policies at this point in our research.

VI. CONCLUSION

This paper outlines our model for a policy framework to augment adaptive runtime models used within cyber-physical systems. We have used energy management of smart microgrids as our application domain to support our proof of principle. This work furthers our overarching goal of self-adaptive control of cyber-physical systems utilizing models. We presented a standalone framework of low level policy constructs capable of describing reactionary behavior for i-DSMLs.

Our future research will concentrate on overlaying this base structure with a self-adaptive framework capable of responding to high level objectives as goals.

ACKNOWLEDGMENT

This work was funded in part by the Research and Creative Activity Grant 2455 from the Office of Research and Sponsored Programs.
REFERENCES


Mark Allison received a BS and a MS degree from the City College of New York in Computer Science and Information Systems respectively. He received his PhD from Florida International University in Computer Science. He has with research interests in Software Engineering emphasizing Model Driven Software Development, energy management systems and merging Artificial Intelligence methods. He has worked in plant engineering and facilities management within the telecommunications industry for close to a decade and is currently an Assistant Professor of Computer Science with the University of Michigan at Flint. In the United States of America. Prof. Allison is a Co-Vice President of South-East Michigan MENSA I and an active member of the Association for Computing Machinery (ACM).

Seung-Jin Lee received a Ph.D. in mechanical engineering from the University of Washington, Seattle, WA, in 2011, M.S. in civil and environmental engineering from Carnegie Mellon University, Pittsburgh, PA, in 2005 and a B.S. in mechanical engineering from Boston University, Boston, MA, in 2003. He is an Assistant Professor of Sustainability and Mechanical Engineering in the Department of Computer Science, Engineering and Physics and Department of Earth and Resource Science at the University of Michigan-Flint, Flint, MI. He was formerly an Oak Ridge Institute of Science and Technology (ORISE) Postdoctoral Research Fellow at the U.S. Environmental Protection Agency’s National Risk Management Research Laboratory. His research interests are focused on incorporating sustainability and design for environment into emerging technologies and to improve technology development, design, and infrastructure through the advancement of life cycle assessment and industrial ecology. Prof. Lee is a member of the International Society for Industrial Ecology and the Korean-American Scientists and Engineers Association.

Anil Kuvvarapu received a B.Tech in Computer Science and M.Tech in Software Engineering from JNT University. He is pursuing his MS in Computer Science from University of Michigan-Flint. He has research interests in Cloud Computing and Data Mining. He has worked as an Assistant Professor of Information Technologies with the JNT University, India and is currently Research Assistant in Computer Science.