Towards Interpreting Models to Orchestrate IaaS Multi-Cloud Infrastructures

Mark Allison *, Stephen Turner* and Andrew A. Allen†
*The University of Michigan-Flint
Flint, MI
{markalli, swturner}@umflint.edu
†Georgia Southern University
Statesboro, GA
andrewallen@georgiasouthern.edu

***Abstract*—One challenge to the cloud computing paradigm is the task complexity associated with designing and managing multi-cloud solutions based on operational objectives. Heterogeneous vendor interfaces and a lack of standardization compounds this complexity and may eventually lead to vendor lock-in.**

In this article we present a model driven approach to allowing network administrators to intuitively describe and rapidly realize non-trivial IaaS behavior in realtime. We have developed iCloudML, an interpreted domain-specific modeling language and its interpreter as tooling support for the domain.

I. Introduction

Cloud computing is an enabling technology for on-demand, networked accessed computing and storage resources[1]. Heralded as the next natural step in information technology provisioning [2], this paradigm represents the delivery of computing and storage resources as a utility. This usage based payment model leverages the economy of scale to allow consumers to employ a vast amount of on-demand resources without the associated risks and overhead of infrastructure provisioning and maintenance. Shared pools of resources are able to be procured or released on an unprecedented scale. The promises of cloud computing is however not without its challenges.

Application programming interfaces (API's) by which the consumer may specify their requirements from cloud vendors remain essentially proprietary with little standardization efforts [1]. This phenomena makes it difficult for the consumer to migrate all or part of the infrastructure to a more desirable cloud, leading to *vendor lock-in*. This lock-in exposes the consumer to the mercy of providers in terms of reliability issues, security/privacy concerns and rate increases. One well referenced example is Linkup's loss of 45% of their customers data [3]. Consumers have seen these concerns as barriers to acceptanceamazon cite. Other concerns which are cause for apprehension and challenges the model include data auditability and performance unpredictability. Consumers with data and performance critical infrastructures require more control of services outside of the CSP's fault tolerance mechanisms.

One method of addressing these challenges is a multi-cloud solution whereby businesses incorporate redundant systems. This requires the ability to safely migrate from one public cloud to the next or even to incorporate a local or private cloud; this latter blend is termed *hybrid* clouds. The business concern of alleviating lock-in by operating a hybrid cloud is non-trivial in its execution and requires constant monitoring and a thorough understanding of the proprietary service models and their best practice application[4]. Furthermore the human expertise required is not easily transferable and becomes itself a single point of failure that was the concern at the onset.

Currently the approaches at providing a homogeneous view of hybrid clouds remains largely ad-hoc. There exists the need for logically centralized decision making for the IaaS hybrid cloud. In this paper we present a model based approach that seeks to encapsulate the differences in disparate systems to provide a simple and intuitive graphical interface to the user. To this end we present, iCloudML, an interpreted domain specific modeling language and its interpreter. Our position is that our model based approach captures domain and vendor specific knowledge to reduce design faults and configuration complexity. In the development of this new language we applied a generic model of execution developed specifically to assist language authors to define the dynamic semantics in a standardized manner[5].

Specifically the contributions of this work are:

- A metamodel for iCloudML an interpreted domain-specific modeling language targeted at allowing network administrators to rapidly and intuitively manage multi-cloud IaaS infrastructures
- An interpreter framework to contain iCloudML semantics based on a Generic Model of Execution.

The remainder of this paper is organized as follows: Section 2 provides an overview of the application domain and the core technologies applied in this approach. Section 3 discusses the language's syntactic and semantic elements and introduces an indicative scenario as guide. Section 4 addresses works which are closely related for context, and we conclude with an overview and future directions for this research path.

II. Background

In this section we discuss the underlying concepts and technologies which form the bases for this research. We first introduce our domain of discourse, IaaS and subsequently overview I-DSMLs, our enabling technology.

978-1-4799-6600-4/15/$31.00 ©2015 IEEE

A. Infrastructure as a Service

Driven by the economy of scale and rapid advancement in virtualization technology, Infrastructure as a Service (IaaS) represents a base service model of cloud computing. This model allows for the provisioning of storage, processing and internetworking resources. Consumers of this service are able to rapidly provision abstracted hardware and pay on an as needed basis. Administrators are required to manage both hardware and software (including operating systems). The attraction to this model is that ability of an organization to reduce significantly expensive hardware and staffing needs of a data intensive infrastructure.

B. Interpreted DSMLs

Model- Driven Software Development (MDSD), represents a paradigm shift in creating complex software systems[6]. Instead of objects as first class artifacts the methodology utilizes models as a higher level of abstraction. While relatively new, MDSD has demonstrated significant promises towards productivity gains over object oriented development[7]. Key to realizing the goals of MDSD is the use of domain specific modeling languages (DSMLs) with a more focused expressiveness within a particular problem domain than their general purpose counterparts such as Java and C++.

Their are two primary ways to realize behavior using models. The first and more predominant approach is to translate a model to a High Level Language as an intermediate representation, then compile and execute to realize behavior. The alternate approach is to utilize a specialized interpreter to realize behavior directly. This interpreted methodology allows for changes to be made at runtime without explicitly rebuilding, retesting and redeploying the model as with the code generation approach [8]. The approach to interpretation being applied in this research relies on execution semantics determined by changes to models at runtime. We refer to this taxonomy as i-DSML.

The language's interpreter which is referred to as a domain specific virtual machine, is based on a 4 layered architecture representing a separation of concerns (See Figure 4 for an overview of our specific approach). The uppermost layer, *User Interface*, provides the means by which the user may specify models in an intuitive manner using elements and concepts from the problem domain. The second layer is a *Synthesis Engine* which produces directives (Control Scripts) based on differences between the user preference model and the state of the system under control or *Plant*. The current state of the plant is represented by a causally connected adaptable runtime model. The third, or *Middleware* layer's primary concern is to transform the Control Scripts to platform specific directives and guarantee an execution outcome. The final layer is the *Hardware Brokerage* layer which provides an API to the plant itself. In the next section we will discuss how the I-DSML concept is applied to our domain of discourse, IaaS management.

III. iCloudML

The intent of ICloudML is to provide abstractions of features and concepts to support infrastructure as a service (IaaS) cloud management. As such our DSML requires that knowledge of that domain be gathered and the key concepts captured and subsequently represented within the language's syntactical and semantic constituents.

A. Feature Oriented Domain Analysis

Prior to developing and Domain-Specific Language, it is imperative that there is a systematic and thorough analysis of the application domain [9]. The result of our study of typical usage scenarios and business level objectives of the domain is a taxonomy model depicted by the feature diagram in Figure 1. The diagram captures top level features of the domain, including `resources`, `privacy`, `security`, `resilience`, a service level agreement(`SLA`) and `policies`. The mandatory `resource` feature contains IaaS `virtual machines` (VMs), `storage` and `networking`. `SLAs` will dictate the agreed upon quality of service (`QoS`) which the system will utilize in its benchmarking and performance monitoring. The set of features represented is by no means exhaustive and represents a minimal set of features required to satisfy our proof of principle.

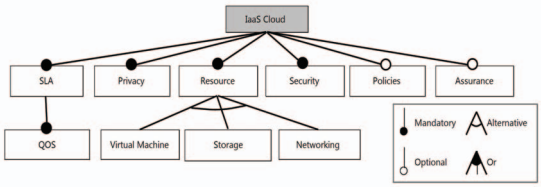

Fig. 1: Feature diagram of the IaaS cloud domain

B. Metamodel

From our feature model we derive the language's syntax via a meta-model, which is a model of models comprising an abstract syntax, static semantics, and concrete syntax. Not all features are able to be addressed by the language's syntax, which is weaved as a concern within the execution semantics. For example `security` is a concern that may not be addressed adequately using the language's syntactic elements, however we may use the syntax to specify the configuration of domain `resources`.

Figure 2 reflects the iCloudML abstract syntax. An IaaS hybrid cloud may contain one or more `CloudSchemas` which may represent a local or remote cloud; the syntax restricts the model to a singleton local cloud. Each cloud schema comprises a QoS container, a `CloudDataSchema` and a `CloudControlSchema`. The latter two elements separate the control (configuration) from data, akin to the separation you may find within a procedural language. The `CloudControlSchema` has logical container types *DataStor* and `ServerClusterArch`. The language utilities `policy` constructs allow for the user to specify more granular behavior[10]. Policies are comprised of rules that are $< Event, Condition, Action >$ triples. An example of a cloud policy would be that at 2am (event), while NAS7 is operational (condition), perform an incremental backup on servers 2 and 7 (Action). A `CloudDataSchema` addresses the actual cloud elements of the `VMs` and `Storage` along with their configuration.

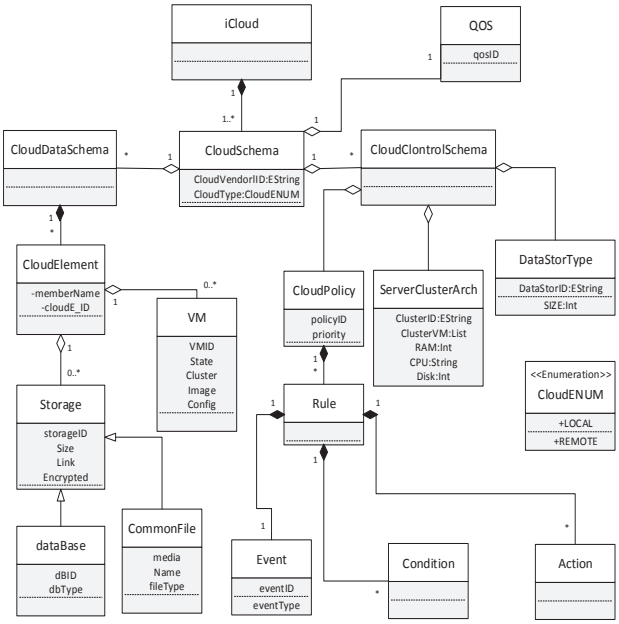

Fig. 2: ICloudML Abstract Syntax

C. Illustrative Scenario

We weave our presentation of the approach by offering an indicative usage scenario from the domain.

□ Dana is a network administrator who desires to create a network infrastructure with messaging and active directory services. Her task specification is as such that there is a high demand for service assurance which implies redundancy. She sets out to develop a multi-cloud solution with a local cloud for her companies more sensitive data. The planned infrastructure will require scalability for future growth. Utilizing current techniques, she would be required to manually create and configure each remote cloud by accessing their interfaces in a stovepipe manner. Intra-cloud coordination and addressing the infrastructure as a sole entity is accomplished in an ad-hoc manner. As the demands on the network grows this solution will slowly become untenable for our poor administrator.

Alternatively, to accomplish orchestration utilizing iCloudML she begins by creating a control instance as in Figure 3A to describe the desired network configuration and logical groupings of domain elements. Her configuration specifies two IaaS clouds each having their own policies. She will now need to construct a related data instance as in Figure 3B to describe the essential properties of the model elements. Here we see two servers, `Server172` and `Server231`, provisioned for their respective roles. There is a `DataStor` which is linked to both clouds signifying its redundant property. Upon submission the graphical model instances generate text based control and data schemas, which are submitted for realization via the Model Interpreter. We next present the interpreter framework.

D. Model Interpretation

Our view of the interpreter is that of a real-time reactive system steeped in control theory[11]. As such, its role is to maintain its interaction with the environment (plant) it seeks to control. Figure 4 shows an overview of the architecture; we will address each layer in turn within this section, but first, let us present the system more formally.

We begin by defining our environment E as a finite set of discrete states.

$$E = \{s, s', s'', ...\} \tag{1}$$

The semantics of the language are based on changes to models at runtime. These changes are interpreted via change mapping to produce directives to the *plant*. Each model is a tuple $< CCS, CDS >$ whereby CCS is a `CloudControlSchema` and CDS is a `CloudDataSchema`. The i^{th} user preference model $U_i =< CCS_{u_i}, CDS_{u_i} >$ is compared to the current state to the system s_c causally represented by Runtime Model $R_c =< CCS_{r_c}, CDS_{r_c} >$ to produce a set of changes δ. So our model comparison may be defined as:

$$MC : U_i \setminus R_c \rightarrow \delta \tag{2}$$

Change Interpretation CI maps the set δ to a set of control scripts ρ , producing:

$$CI : \delta \mapsto \rho \tag{3}$$

Since δ respects the system state we can then describe our interpretation as:

$$E \times U \rightarrow E \times \rho \tag{4}$$

We next discuss the role of each layer of the interpreter.

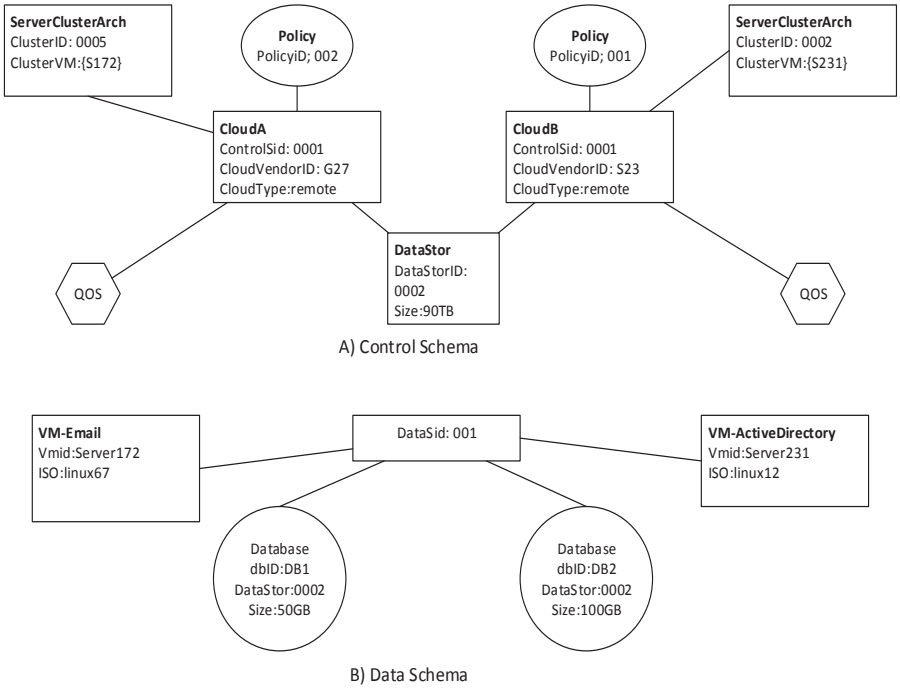

Fig. 3: ICloudML Scenario Model Instances

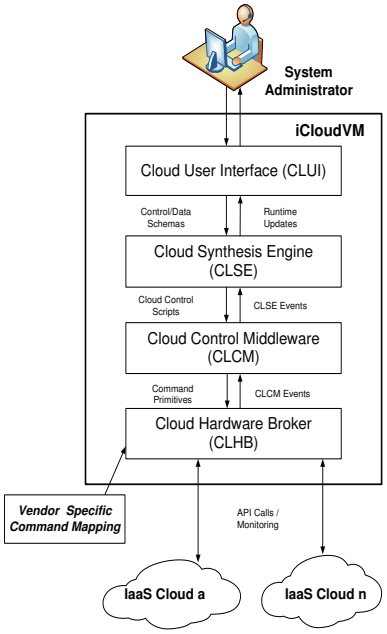

Fig. 4: iCloudML Interpreter's Layered Architecture

1) Cloud User Interface: This layer affords the user to design and validate preference models in an intuitive manner. There are three subsystems involved: (1) An *iCloud Modeling Environment* (iCME) which allows expert users to create and validate graphical schema instances (G-iCloudMl). Revisting our scenario, it is the iCME that facilitates the development of our administrator's control and data instances as in Figures 3(A) and (B); (2) An IaaS user interface allows for novice users to monitor and make changes to models at runtime in a drag and drop manner (Figure 5 shows a sample interface); and (3) A schema transformation environment, which transforms the graphical instances in (1) and (2) to X-iClouldML, the text based representation that serves as input to the Synthesis Engine.

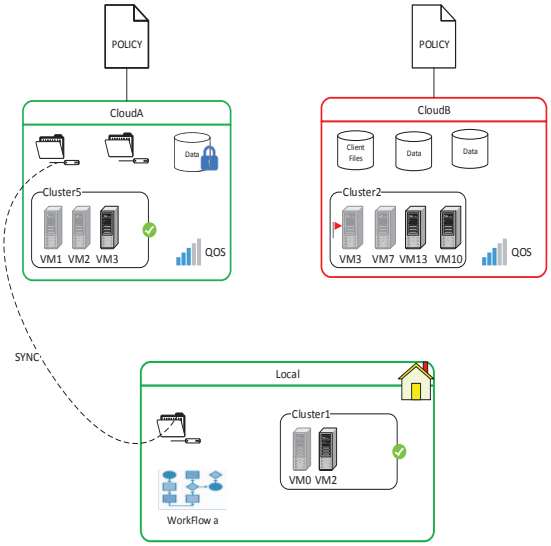

Fig. 5: IaaS User Interface

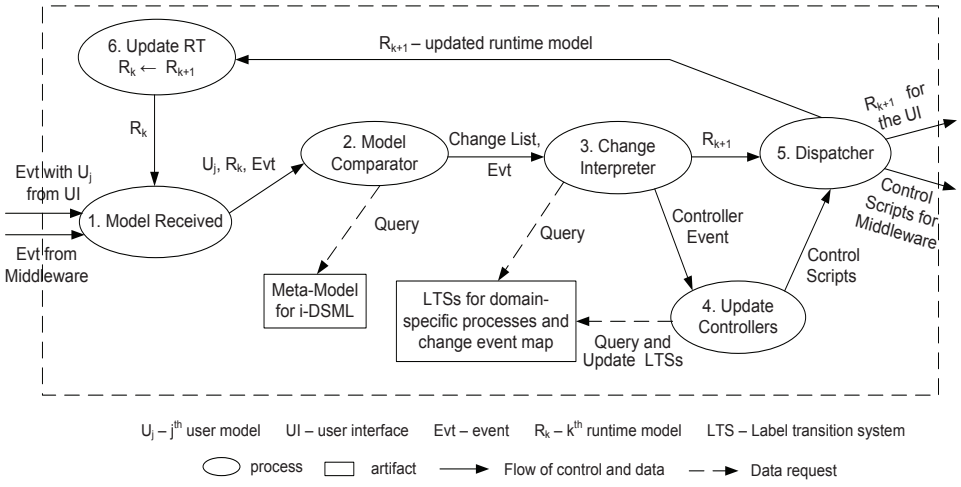

Fig. 6: Generic Model Synthesis Approach

2) Cloud Synthesis Engine: This layer hosts most of the execution semantics of the language. It takes as input events occuring within the plant and X-iCloudML model instances, and produces iCloud Control Scripts that are sent to the lower Middleware layer for execution. Figure 6 is a portrayal of a generic model of execution from Allison et.al.[5], upon which the SE is based. There are two primary subsystems at this layer: A *Model Comparator* and a *Change Interpreter*.

The submission of a new user model U_i triggers a comparison with the current *runtime model* R_k. Our *Model Comparator* uses a differencing algorithm (typically most XML type differentiation algorithms may be applied with minimal augmentation) to produce a *ChangeList*. An individual change within the aggregate *ChangeList* is of the form `ADD, DEL` or `CHG`. The *Change Interpreter* queries a change event map and the set of active Labeled Transition Systems (LTS) to determine the state of the particular model element. An event raised by the mapping signals a transition within the LTS and generates the control script; the LTS may be replaced by any structure based on output automatas. Figure 7 shows a minimalistic state transition diagram for a iCloudMl Virtual Machine, which may be extrapolated to a LTS. Note that this abstraction is minimalistic as we focus on a proof of principle. Once the lower layers have confirmed execution of a control script, *UpdateRT* updates the runtime model to R_{k+1} and the system awaits another signal.

Following our earlier scenario, the submission of a control instance in Figure 3A would be compared to the current runtime model. Since the runtime model is initialized to *null* at startup, the comparison would generate a changeList {ADD CONTROLSCHEMA (ControlSID=0001 CloudVendorid=G27..). ADD POLICY (PolicyID=002, Parent=ControlSID0001) ....}. In turn the change mapping would raise an ADDCS event creating a new state machine for our new Control Schema. This would be accomplished for each element in turn. Each change enacted is reflected in the Cloud User Interface.

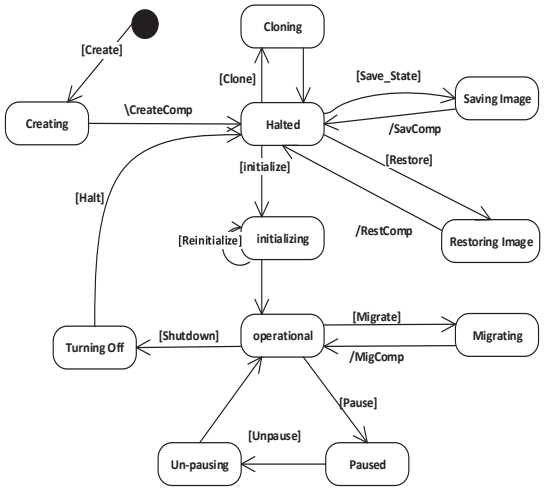

Fig. 7: State Transition Diagram for Virtual Machine Element

3) Cloud Middleware: This layer supports the execution of the control scripts from the synthesis engine. Control scripts that are common platform independent primitives are made cloud vendor specific by dynamically loading and interpreting corresponding macros. This layer is charged with handling plant and internal events that may not warrant a alteration in the runtime model.

4) Cloud Brokerage Layer: As the lowest layer, the CBL exposes an API from the different cloud vendors to the middleware. Ensuring the connection and monitoring of performance to satisfy QoS is a critical concern.

IV. Related Work

There are several works in the literature that addresses a domain specific approach to cloud computing. Ferry et.al.[12]

argues the need for multi-cloud system mamangement and proposes the Cloud Modeling Language CloudML, a domain-specific modeling language and interpreter concept to support administration for the IaaS and PaaS cloud stacks. While the spirit and motivation for the approaches are similar, our approach differs in that we have curtailed our approach to IaaS and based our interpreter semantics on model changes at runtime. We place emphasis on the use of models with policy augmentation to support behavior descriptions within our runtime model. Bunch et. al. [13] proposes Neptune, a domain-specific language geared towards the configuration and deployment of high performance computing software to cloud platforms. Our approach differs as we utilize a visual based language that is directly interpreted. Our approach looks at a more low level provisioning of resources. Brandtz et.al.[14] presented a similar text based approach to the deployment of cloud-aware applications. Pim4Cloud DSL, as it is called, is a generative language while ICloudML is directly interpreted.

Our approach is based upon lessons learned within the development of two prior i-DSMLs. In the first, the communications modeling language (CML)[15] was developed to provide a user-centric environment to amalgamate communication requirements. The second, the Microgrid Modeling Language [16], addresses the issue of power management within the smart microgrid domain. Stemming from these early works we subsequently developed a generic model of execution [5], which formed the basis of iCloudML. This work futhers the validation of the utility and robustness of the i-DSML methodology.

V. Conclusion

In this paper we presented an approach to managing multi-cloud infrastructures utilizing an interpreted domain-specific modeling language. This approach addresses vendor lock-in and allows for seemless manipulation of multi-cloud IaaS infrastructures. In developing this concept we have expanded the application of our generic model of execution.

Our future work will involve extending the functionality of the language and its interpreter to address a more expansive primitive set.

Acknowledgment

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

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] M. A Vouk, "Cloud computing–issues, research and implementations," *CIT. Journal of Computing and Information Technology*, vol. 16, no. 4, pp. 235–246, 2008.

[3] J. Brodkin, "Loss of customer data spurs closure of online storage servicethe linkup," *Network World (August 2008)*, 2008.

[4] B. Satzger, W. Hummer, C. Inzinger, P. Leitner, and S. Dustdar, "Winds of change: From vendor lock-in to the meta cloud," *IEEE internet computing*, vol. 17, no. 1, pp. 69–73, 2013.

[5] M. Allison, P. J. Clarke, and X. He, "A generic model of execution for synthesizing interpreted domain-specific models," *International Conference on Soft Computing and Software Engineering*, 2015.

[6] B. Hailpern and P. Tarr, "Model-driven development: the good, the bad, and the ugly," *IBM Syst. J.*, vol. 45, no. 3, pp. 451–461, 2006.

[7] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering*. IEEE Computer Society, 2007, pp. 37–54.

[8] J. den Haan, "Model driven development: Code generation or model interpretation?" [Online]. Available: http://www.theenterprisearchitect.eu/archive/2010/06/28/model-driven-development-code-generation-or-model-interpretation

[9] A. Van Deursen and P. Klint, "Domain-specific language design requires feature descriptions," *Journal of Computing and Information Technology*, vol. 10, no. 1, pp. 1–18, 2002.

[10] M. Allison, S.-J. Lee, and A. Kuvvarapu, "A policy based approach to models at runtime within cyber-physical systems: A case study in microgrids," *The International Conference on Software and Information Systems*, 2015.

[11] J. C. Doyle, B. A. Francis, and A. Tannenbaum, *Feedback control theory*. Macmillan Publishing Company New York, 1992, vol. 1.

[12] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Proceedings of the IEEE Sixth International Conference on Cloud Computing, CLOUD*, vol. 13, 2013, pp. 887–894.

[13] C. Bunch, N. Chohan, C. Krintz, and K. Shams, "Neptune: a domain specific language for deploying hpc software on cloud platforms," in *Proceedings of the 2nd international workshop on Scientific cloud computing*. ACM, 2011, pp. 59–68.

[14] E. Brandtzæg, P. Mohagheghi, and S. Mosser, "Towards a domain-specific language to deploy applications in the clouds," in *Cloud Computing 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, 2012, pp. 213–218.

[15] Y. Deng, S. Masoud Sadjadi, P. Clarke, V. Hristidis, R. Rangaswami, and Y. Wang, "Cvm-a communication virtual machine," *Journal of Systems and Software*, vol. 81, no. 10, pp. 1640–1662, 2008.

[16] M. Allison, K. Morris, Z. Yang, F. Costa, and P. J. Clarke, "Towards reliable smart microgrid behavior using runtime model synthesis," in *IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE), 2012*, 2012, pp. 185–192.