A Model-driven Approach to Develop and Manage Cyber-Physical Systems *

Adalberto R. Sampaio Junior[1], Fábio M. Costa[1], and Peter Clarke[2]

[1] Instituto de Informática, Universidade Federal de Goiás, Brazil
{adalbertojunior|fmc}@inf.ufg.br
[2] School of Computing and Information Sciences, Florida International University
clarkep@cis.fiu.edu

Abstract. Cyber-Physical Systems (CPS) integrate computing, networking, and physical processes to digitally execute tasks on or using the physical elements of a system. Power microgrids are a particular kind of CPS that enables management and autonomic control of local smart grids, aiming at reliability, fault tolerance and energy efficiency, among other goals. This paper explores a new approach based on MDE that uses models at runtime techniques to manage and control microgrids. The approach employs a model execution engine that manages a causally connected runtime model of the microgrid and interprets user-defined models in order to generate controls for the microgrid elements. We demonstrate the approach by building the lower layer of the model execution engine. Furthermore, we explore a model-driven technique to build the execution engine and use the resulting experience to argue that the approach can be extended to other kinds of cyber-physical systems.

1 Introduction

Cyber-physical systems (CPS) are an integration of computational and physical processes that can interact with humans and the environment [1, 2]. These systems have operations that are monitored, coordinated, controlled and integrated, typically using with feedback loops where physical processes affect computation and vice versa [3, 4].

CPS research is structured into a number of sub-disciplines, with abstraction and architecture, modeling, and control as some of the main study areas [5, 2]. The architecture of CPS must provide a common way to handle both the cyber and physical elements of systems, but as CPS is a new research area, many architectural details remain open. Furthermore, modeling CPS is still a challenge when it comes to the integration of system and model [5, 6]. Clearly, model driven engineering (MDE) is one of the key techniques to solve these architectural and integration challenges [5, 2, 6, 4].

The goal of MDE is to raise the abstraction level and reduce the effort and complexity of development [7]. In CPS, models can be used at different architectural levels [6] and, with models@run.time (M@RT) [8], they can be used not

* This work was supported by FAPEG and CNPq (Grant # 473939/2012-6).

only to build and configure the system, but also to dynamically monitor and control its functionality and behavior while running.

Using M@RT, a system may have a high-level, causally connected (i.e., reflective), representation of itself in the form of one or more related models that are kept in sync with the system as it executes. This is so that changes in the system are immediately reflected in the model, and vice-versa.

A promising approach to M@RT consists in using a model execution engine that maintains a causally connected runtime model of the system and interprets high-level user-defined models that may be submitted at any time as the means for users to dynamically reconfigure the system [9]. This approach to configure systems can decrease the burden of designing and maintaining complex systems, such as CPS, as it eliminates the dependence on highly skilled personnel [10].

This paper explores a model-driven approach to build model execution engines, and propose its use for the management of CPS at different levels, from definition (topology) to the control of their physical components. We present a realization of this approach in the domain of microgrid energy management, a kind of CPS that is now common in many parts of the world as a way to optimize the use of distributed energy resources (DERs) [5]. The model execution engine consists of different layers, each with a different abstract view (a model at runtime) of the CPS, thereby making it easier for these models to be verified and analyzed, based on the concerns of that layer. In this work we focus on the layer that interfaces with the physical components of the microgrid.

Section 2 describes how microgrids and CPS are conceptually linked with models at runtime in our work, and the implications of this for microgrid management. Section 3 presents the architecture for a model execution engine to manage and execute microgrid models. In Section 4 we describe how the physical elements of the microgrid are accessed via the model execution engine, and how the internal mechanisms of the execution engine were developed using MDE. In Section 5 we discuss related work, and in Section 6 we conclude with a discussion on the impact of M@RT for the development of CPS.

2 The role of M@RT for CPS and Microgrids

Microgrids are small, typically local, power grids that integrate small scale and distributed energy sources into electrical distribution systems, improving energy efficiency with the use of smart control techniques. This approach has a positive impact on power consumption, as it lowers the emission of greenhouse gases and reduces costs related to power transmission. Besides these advantages, microgrids also help increase the overall reliability of the power infrastructure. For instance, by using control policies it is possible to configure the behavior of the microgrid in order to maintain vital components working seamlessly, even in the presence of grid faults, as must happen in critical facilities, such as hospitals.

The smart behavior of a microgrid is enabled by a set of controllers that manage the loads, sources and storage elements of the plant. A microgrid has two types of controllers: local controllers that manage the individual microgrid

elements, and a central controller that manages all local controllers and is responsible for coordinating the entire system, optimizing its behavior according to policies set by the microgrid owner. The controllers are connected forming a complex distributed system that manages all physical devices in the microgrid.

Controllers can process events from devices, enabling the management mechanisms to react to changes in the plant. In addition, they receive and process commands from the user (in our case, from the execution engine), which are transformed into commands to control the physical devices. Therefore, controllers are at the frontier between the cyber and physical elements in this domain.

In CPSs in general, and microgrids in particular, problems such as the heterogeneity of devices pose a great challenge to the design of large-scale systems. In addition, security, real-time assurance and network connectivity are also challenges that must be addressed when designing such systems. Several techniques have been considered to improve the design of CPSs, models being among them. Generally speaking, model-driven mechanisms can be used to capture events from the physical elements and, based on such events and on what is prescribed in the models, generate commands to configure and/or control those elements.

Furthermore, the models used to design and control CPSs can be specified in user-friendly domain-specific modeling languages (DSML), which abstract away many of the system's low-level concerns and focus on modeling constructs that are familiar to users. In addition, automatic processing of such models by an execution engine enables them to be directly used to exert the user's intent in the form of the corresponding changes in the system.

In the context of microgrids, [11] proposes a domain-specific modeling language, MGridML, which captures the structural and semantic features of microgrids. This language is interpreted by a layered execution engine, MGridVM, which reads user input models and configures the microgrid accordingly.

Each layer of MGridVM maintains part of the runtime model of the microgrid, thus keeping track of the state and configuration of the plant. As each layer has its own part of the runtime model, different aspects of the configuration are maintained during system execution and can be dynamically changed as a result of events from the plant elements or new input models submited by the user.

Aiming to generalize the applicability of such model execution engines, and also to decrease their complexity of development, [12] proposes an approach that uses metamodels to design and implement execution engines such as MGridVM for different application domains. In the following we show how this approach can be used to build execution engines to manage CPSs. Specifically, we demonstrate the use of the approach to build the bottom layer of the execution engine (the layer that accesses the physical resources) for microgrids, providing insights on how the approach could be exploited in other CPS application domains.

3 Architecture of the execution engine

MGridVM is a model execution engine that interprets models built using the MGridML language [11]. MGridML models are composed of two schemas that

together define the configuration of a microgrid: the control schema, which represents the logic configuration and the policies that govern the structure and behavior of a microgrid; and the data schema, which represents the actual physical elements in the plant.

As shown in Figure 1, MGridVM is built using a layered architecture inspired by the CVM platform, which defines a model execution engine for the communications domain [9]. Each layer deals with a distinct stage of model creation and execution, besides maintaining state in the form of a runtime model.

When MGridVM executes a microgrid model, the model provided by the user is compared with the current runtime model, and any differences between the two are used to generate commands (or calls) to reconfigure the microgrid accordingly. These model differences are also used to update the runtime model, which always reflects the current microgrid configuration. In addition, any relevant changes in the underlying microgrid plant cause the generation of events, which are processed by MGridVM in order to update the runtime model accordingly, thus completing the causal connection link. In addition, an event may also trigger an autonomic management mechanism, which uses the runtime model to reason about the system and react by performing any necessary reconfiguration in response.

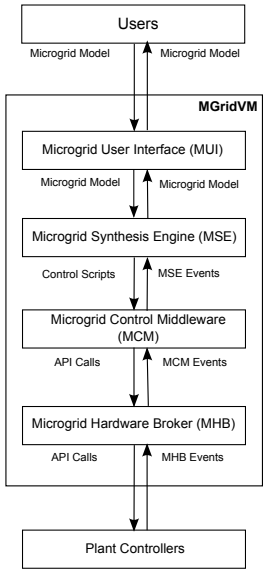

Fig. 1. Architecture of MGridVM and the flow of the calls and events during the microgrid model processing [11].

Thus, communication between adjacent layers, or between the MHB layer and physical devices, happens through calls to (or events from) a layer or physical device.

Calls are processed as transactions, avoiding inconsistencies in the runtime model maintained by the execution machine. When a call is processed, other calls may be generated from it; when this occurs, calls are chained with their parent call, creating an execution tree. A call may only apply its changes at the runtime model of a layer if, and only if, all its child calls are successfully executed in their respective contexts (i.e., layer); if this does not happen, a rollback of all calls in the execution tree is carried out, and upper layers are notified in the form of events.

Events may be raised at any layer of the architecture. If a event is received (from below) by a layer and causes its runtime model to change, such change needs to be reflected at the runtime model of all layers in order to avoid inconsistencies between the layers. Therefore, similarly to calls, events also need to be processed as transactions in order to ensure that their effects will only be made permanent if event processing at all layers is successful.

In the following, we describe the layers of MGridVM.

Microgrid User Interface - MUI: provides the user (specialist or not) with the ability to specify the configuration and requirements of a microgrid in the form of a graphical model. MUI validates this model and transforms it into an XML-based model, which is then passed to the next layer for processing.

Microgrid Synthesis Engine - MSE: receives a text-based model from MUI and transforms it into control scripts to be executed by the lower layers. It performs a comparison between the input model (from MUI) and the current runtime model and, according to the differences between them, generates control scripts. These scripts are used by the lower layers to effect the corresponding changes in the underlying microgrid plant. Control scripts are generated by MSE using a series of state machines that constitute a labeled transition system (LTS) at the heart of MSE.

Microgrid Control Middleware - MCM: executes the commands of the control scripts in order to manage the controllers and their managed device types, mapping the types (logic representation of devices) to specific physical devices in the system. MCM may also apply non-functional properties to the commands before they are sent to the MHB layer.

Microgrid Hardware Broker - MHB: executes the commands received from MCM and applies them to the physical devices. MHB has a generic API to handle physical devices in a vendor-independentsway. It also has a number of adaptors that translate the vendor-independent commands into vendor-specific commands for each device, thus dealing with the heterogeneity problem. In addition, MHB has an autonomic manager (see Section 4.1) that receives and processes events generated by the underlying devices as described above.

4 Metamodel-based implementation

MDE is a suitable approach for the development of complex systems such as CPSs [6], but the use of models to develop such systems brings another kind of complexity: model interpretation is a non-trivial process, and the related mechanisms are hard to develop. In this work, we employ the approach proposed in [12] to reduce this complexity of developing such mechanisms. In what follows, we present the metamodel proposed in [12] and describe its use to develop the MHB layer of MGridVM.

A high-level view of the metamodel is shown in Figure 2. It defines a set of managers, each one with a specific function in the broker. Managers are responsible for processing the events received from the physical devices and the calls received from the upper layers, as well as for applying the appropriate policies and context information in this process.

The main class of the metamodel is **Manager**, and an instance of this class represents the broker from the point of view of the layer above (MCM). It defines the scope for resource management and groups other components (lower level managers) that define the specific attributes and functionality of the broker layer. Figure 2 shows how the following elements are associated: (a) **Interface**

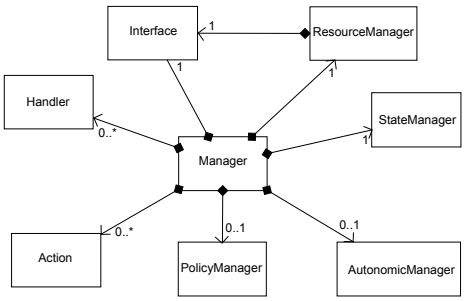

Fig. 2. Metamodel used to develop the broker layer of the execution engine.

– defines the operations supported by a manager and the events it may generate; (b) **Action/Handler** – defines how events and calls are processed; (c) **ResourceManager** – defines the resources managed by a specific Manager, including their interfaces and how they are obtained; (d) **StateManager** – handles the data structures that represent the runtime model maintained by the layer; (d) **AutonomicManager** – defines the elements that provide the autonomic behavior of the layer; and (e) **PolicyManager** – handles and evaluates policies for resource selection.

Once an instance of **Manager** is created, it can be used as a resource for another (higher-level) instance of **Manager**. This enables the creation of a hierarchy of managers as part of the architecture of the MHB layer. In this work, this hierarchy is key to the management of hierarchical microgrids.

4.1 Microgrid Hardware Broker Implementation

A microgrid system has several controllers that manage its physical devices, applying policies that govern their behavior in the presence of calls (from the upper layer) and events (from the physical devices). Controllers group physical devices according to their characteristics and the topology of the microgrid.

All configuration actions upon physical devices are performed in terms of procedures defined in the controller interface. These procedures are responsible for controls such as turning on and off a device or controller, or getting/setting device or controller properties. Thus, when designing a model-based microgrid management system, we can abstract the low-level details the microgrid plant and focus on the interface of the controllers.

As an aside, controllers can manage both legacy and smart devices. In the former case the controller has to process all signals raised by the devices, and send only elementary controls to the devices. In the latter, the devices have mechanisms to process signals and are able to execute more complex commands sent by the controller.

In a hierarchical microgrid [13, 14], as shown in Figure 3, there are two kinds of controllers: the central controller, which coordinates the microgrid and provides its non-functional properties; and several local controllers, which manage

the elementary physical devices, such as distributed energy resources (DER) and loads. Each kind of local controller manages a specific kind of resources. The central controller in turn manages a group of local controllers as resources. As a result, the MHB layer manages resources at different levels, working as an abstraction of the microgrid plant, meaning that the calls supported at its interface are a superset of the calls that are available to control the plant.

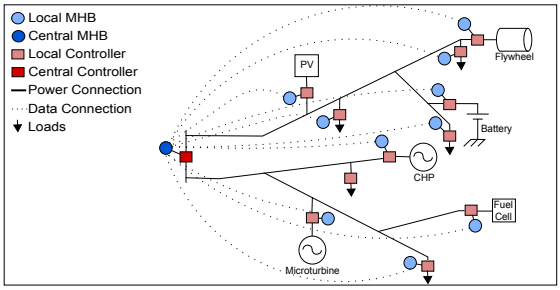

Fig. 3. Hierarchical microgrid topology (based in [13]) and its relationship with the MHB.

As seen in Figure 4, when modeling the MHB layer using the proposed MDE approach, the metamodel is used to build a two-level broker, composed by a centralized top level, called Central MHB, which abstracts the central controller interface, and a distributed bottom level, formed by several sub-brokers, called Distributed Local MHBs, each one abstracting a local controller in the microgrid. Both the central and local MHBs are instances of class Manager.

At the broker's top level, the *Central MHB* has a **ResourceManager** that groups the several local controllers of the microgrid and provides an interface to access them. The **StateManager** at this level maintains the state of the microgrid's central controller as well as references and properties for each of the local controllers. This state is part of the runtime model maintained by MHB and thus is defined according to the metamodel.

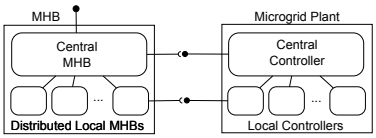

Fig. 4. Interfaces between the MHB and the plant of the microgrid.

At the lower level, the *Distributed Local MHB* works in a similar way and its **ResourceManager** keeps track of the elementary devices of the microgrid, exposing the interfaces used to change their properties in the case of smart devices. Moreover, the **StateManager** at the bottom level maintains its share of the runtime model, namely the state of a local controller and the properties and references to the physical devices it controls.

The resources managed by the MHB layer are used through an interface that is a generic facade to physical devices. For each different kind of device there is a specific resource class, which abstracts the vendor-specific interfaces.

When a call arrives at MHB from the upper layer, it is handled by both levels before being executed on a physical element. The call is first analyzed by the *Central MHB*, which selects the *Distributed Local MHB* representing the local controller that manages the resource targeted by the call. The selection is made by querying the runtime-model and its policies.

Events that arrive from the physical resources are handled in a similar way. An event is first analyzed by the manager controlling the resource that generated it, and then by the central manager. If the model has a handler for the event, the event is analyzed by the corresponding manager, which selects the resource or the runtime property to be reconfigured. On the other hand, if no handler exists, the event is forwarded to the upper layer (MCM) where it may be processed.

At both levels, the **AutonomicManager** and the **PolicyManager** are responsible for the autonomic behavior of MHB.

4.2 Model execution by the MHB layer

A microgrid configuration model is created and interpreted in a series of steps carried out by the four layers of MGridVM. Note that each layer, including MHB, can only process what can be modeled using MGridML, the DSML of MGridVM. Thus, if MGridML has some limitation to model a microgrid, this reflects as a limitation on MHB's management capabilities. The model execution process is described below.

First a graphical model is created at the MUI layer, using MGridML, and analyzed by the MSE layer. At MSE, a control script is generated and sent to the lower layers for execution. An example script generated by MSE is shown in Table 1. The process carried out by MUI and MSE is presented in detail in [11].

Execution Order	Command	Comment
1	initializeMGrid("MCG0001")	start microgrid
2	addLoadController("LC001")	add load controller LC001
3	addPCCController("PCC001")	add point common coupling PCC001
4	addStorageController("SC001")	add storage controller SC001
5	addLoadDeviceType("LDT001",..."LC001")*	add type LDT001 to controller LC001
6	addMeterType("SMT001", ..., "PCC001")*	add type SMT001 to controller PCC001
7	addStorageDeviceType("SDT001", ... "SC001")*	add type SDT001 to controller SCC001
8	addLoadDevice("LD001", "LDT001", ...)	add device LD001 related to type LDT001
9	addSmartMeter("SM001", "SMT001" ...)	add device SM001 related to type SMT001
10	addStorageDevice("SC001", "SDT001" ...)	add device SC001 related to type SDT001

Table 1. Sample control script – commands marked with a "*" are executed only at the MCM layer, and do not need be transformed and sent to MHB.

Based on the script generated by MSE, MCM executes and configures its runtime model. After this step, commands may need to be executed on the physical devices to actually configure the microgrid. These commands (namely `addXxxDevice`, `remXxxDevice`, `addXxxController`, `remXxxController`,

`setProperty`, `requestProperty` and `initializeMGrid`) need to be sent to MHB for execution. However, they must first be transformed into the lower-level commands supported by the MHB interface, as shown in Table 2.

When MHB receives calls (commands) from MCM (except for calls 8 and 9 in Table 2), each call usually follows two distinct flows of execution: one to configure the physical devices, and another to configure the layer's runtime model.

Commands 1, 4, 5, 7 and 9 are related to controllers, and generally need to be sent only to the top level of MHB. At this layer, calls are handled to the Central Controller (flow 1) and, if their execution is successful, the runtime model of this layer is updated (flow 2); otherwise an exception is thrown and is handled as an event by the *Manager*, as described in Section 4.

	MCM Command	MHB Call
1	initializeMGrid	start
2	addXxxDevice	addDevice
3	remXxxDevice	remDevice
4	addXxxController	addCtrlr
5	remXxxController	remCtrlr
6	setProperty	setDevProperty
7	setProperty	setCtrlrProperty
8	requestProperty	getDevProperty
9	requestProperty	getCtrlrProperty

Table 2. Mapping between the MCM commands and the MHB interface

Note that commands 4 and 5, although not related to DERs and loads, need to be sent to the bottom level, after transforming them into `startCtrlr` and `stopCtrlr` commands, respectively, in order to match the bottom level API. This is required because controllers are not necessarily physically added or removed from the system, but started or stopped in the microgrid topology.

On the other hand, commands 2, 3, 6, 8 are related to devices, and thus need to be sent to the bottom level of MHB, where the layer has access to the microgrid's Local Controllers, which can directly access the DERs and loads.

5 Related Work

A similar approach to microgrid management using a multilayer architecture is employed in [15], which proposes an architecture based on economic and technical criteria. The microgrid model is structured in two layers, a bottom layer that inspects DERs and loads, and a top layer that receives signals from the bottom and applies configurations to optimize execution. In contrast to our approach, signals are sent intermittently by the system to the top layer. In our approach, the bottom level of the broker captures events and, only when necessary, throws them upwards for analysis, avoiding bottlenecks in large systems due to a large amount of events that need to be dealt with by the centralized part of the broker.

In a more general sense, the use of models do develop CPSs has also been proposed. In [16] a model-driven approach is used to control fault tolerance in CPSs. The model represents how events propagate through the components and how the system analyzes them to prevent faults. The approach to event handling is similar to ours. It defines the events that need to be treated, as well as the actions that must be applied. The main difference is that we treat events as soon they arrive at the broker, while in [16] events must propagate to different components in order to be handled.

In [17] an approach is presented to build CPSs using UML-based models. As in our work, it can, in principle, be used to manage any kind of CPS. However, their approach is limited to code generation, not allowing runtime management.

Finally, within our group work has been carried out that proposes a domain-independent definition of model execution based on metamodeling [12]. The approach enables construction of model execution engines for different domains as instances of the proposed metamodel. We build on this work by adapting it and demonstrating its applicability to the domain of CPS.

6 Conclusion

This paper presented a model-driven approach to manage microgrids in the form of an engine (MGridVM) that executes microgrid models. Despite being work in progress, the way microgrid concepts were linked to the execution engine constructs, and the way the microgrid components were embodied by the broker's metamodel indicates that this is a promising approach for other kinds of CPSs. The metamodel for the broker layer captures the main control features of a CPS: event manipulation, actuation on physical devices and coordination. These features are hard to manage using more traditional CPS development techniques [2], and the use of a model-driven approach is a way to overcome the challenges involved. The model used to build the broker defines, in the handlers, the way events and calls related to physical devices must be treated, without implementation-specific concerns such as the kind and location of the physical device, which are common concerns in traditional CPS development.

Coordination of the system is also managed in a natural way through policy and autonomic managers. Using the policies in the model, the broker can autonomously respond to events from the physical devices, as well as to calls from above to configure the microgrid, thus autonomically adapting system behavior.

Furthermore, our work provides additional validation for the use of models as a generic way to build broker layers for model execution engines, as proposed in [12]. The approach was first validated in the communication domain, and now we demonstrated its applicability for microgrids, with further indication of its applicability in the more general domain of CPS.

Implementation of MHB is currently being concluded and will enable a preliminary validation, using a microgrid simulator, of the results of call/event processing and runtime model maintenance. Future work involves analyzing how the approach impacts microgrid management, and what are the tradeoffs between the cost of model processing and the flexibility provided by the approach for configuration and autonomic control of microgrids, and whether these costs are acceptable in real microgrids and in other kinds of CPSs.

Another important issue to be tackled is the autonomic management of microgrids, using policies to capture the user's preferences (user's policies) and the general behavior of any microgrid (system's policies). Furthermore, we also want to investigate the use of the approach to improve the reliability CPSs, considering issues such as electric power safety and stability.

References

[1] Baheti, R., Gill, H.: Cyber-physical Systems. The Impact of Control Technology (March 2011) 161–166

[2] Sanislav, T., Miclea, L.: Cyber-Physical Systems-Concept, Challenges and Research Areas. Journal of Control Engineering and Applied Informatics **14**(2) (2012) 28–33

[3] Rajkumar, R.R., Lee, I., Sha, L., Stankovic, J.: Cyber-physical systems. In: Proceedings of the 47th Design Automation Conference on - DAC '10, New York, New York, USA, ACM Press (2010) 731

[4] Lee, E.a.: Cyber Physical Systems: Design Challenges. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), IEEE (May 2008) 363–369

[5] Derler, P., Lee, E., Vincentelli, A.S.: Modeling Cyber–Physical Systems. Proceedings of the IEEE **100**(1) (January 2012) 13–28

[6] Karsai, G., Sztipanovits, J.: Model-integrated development of cyber-physical systems. Software Technologies for Embedded and Ubiquitous Systems (2008) 46–54

[7] Hailpern, B., Tarr, P.: Model-driven development: The good, the bad, and the ugly. IBM Systems Journal **45**(3) (2006) 451–461

[8] Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer **42**(10) (October 2009) 22–27

[9] Deng, Y., Masoud Sadjadi, S., Clarke, P.J., Hristidis, V., Rangaswami, R., Wang, Y.: CVM – A communication virtual machine. Journal of Systems and Software **81**(10) (October 2008) 1640–1662

[10] Wu, Y., Allen, A.A., Hernandez, F., France, R., Clarke, P.J.: A domain-specific modeling approach to realizing user-centric communication. Software: Practice and Experience **42**(3) (March 2012) 357–390

[11] Allison, M., Morris, K.A., Yang, Z., Clarke, P.J., Costa, F.M.: Towards Reliable Smart Microgrid Behavior Using Runtime Model Synthesis. In: 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering. Number Cm, IEEE (October 2012) 185–192

[12] Sousa, G.C.M., Costa, F.M., Clarke, P.J., Allen, A.A.: Model-driven development of DSML execution engines. In: Proceedings of the 7th Workshop on Models@run.time - MRT '12, New York, New York, USA, ACM Press (2012) 10–15

[13] Zamora, R., Srivastava, A.K.: Controls for microgrids with storage: Review, challenges, and research needs. Renewable and Sustainable Energy Reviews **14**(7) (2010) 2009 – 2018

[14] Jiang, Z., Dougal, R.A.: Hierarchical microgrid paradigm for integration of distributed energy resources. In: 2008 IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, IEEE (July 2008) 1–8

[15] Enrich, R., Skovron, P., Tolos, M., Torrent-Moreno, M.: Microgrid management based on economic and technical criteria. In: 2012 IEEE International Energy Conference and Exhibition (ENERGYCON), IEEE (September 2012) 551–556

[16] Dubey, A., Karsai, G., Mahadevan, N.: Model-based software health management for real-time systems. 2011 Aerospace Conference (March 2011) 1–18

[17] Magureanu, G., Gavrilescu, M., Pescaru, D., Doboli, A.: Towards UML modeling of cyber-physical systems: A case study for gas distribution. IEEE 8th International Symposium on Intelligent Systems and Informatics (September 2010) 471–476