Model-driven Development of Adaptive IoT Systems

Mahmoud Hussein[1,2], Shuai Li, and Ansgar Radermacher
[1]CEA, LIST, Laboratory of Model Driven Engineering for Embedded Systems,
P.C. 174, Gif-sur-Yvette, 91191, France
[2]Faculty of Computers and Information, Menofia University, Egypt
{mahmoud.hussein, shuai.li, and ansgar.radermacher}@cea.fr, mahmoud.hussein@ci.menofia.edu.eg

***Abstract*- There is an increasing demand for software systems that utilize the new Internet of Things (IoT) paradigm to provide users with the best functionalities, through transforming objects from traditional to smart ones. In recent years, a number of approaches have been proposed to enable the development of such IoT systems. However, developing IoT systems that adapt at runtime is still a major challenge. In this paper, we propose a model-driven approach to ease the modeling and realization of adaptive IoT systems. First, to model an IoT system, we adopted SysML4IoT (an extension of the SysML) to specify the system functions and adaptations. We also adopted a publish/subscribe paradigm to model environment information and its relationship with the system. Second, based on the system design model, code is generated which is deployed later on to the hardware platform of the system. To show our approach applicability, we have developed a smart lighting system in the context of the S3P project.**

Keywords- IoT Systems; Adaptive Software; Model-driven; Publish-Subscribe Paradigm

I. INTRODUCTION

Recently, a growing number of objects have been connected to the internet for realizing the idea of Internet of Things (IoT) in a number of domains such as smart homes, transportation, healthcare, industrial automation, etc. [1]. In these domains, the IoT paradigm transforms objects from traditional to smart ones to provide end users with functions and qualities of IoT systems. Such systems need to be adapted at runtime to take the environment changes into account [2]. A major challenge is how to develop such adaptive IoT software systems [3].

In recent years, a number of approaches have been proposed to ease the development of IoT systems (e.g. [1], [4], [5], and [6]). However, these approaches do not enable runtime adaptation of IoT systems while they are in operation. In addition, some of the existing approaches focus only on one development phase (e.g. analysis [4] or deployment [6]). Thus, there is a need for an approach that fully supports the development of adaptive IoT software systems.

In this paper, we propose a model-driven approach to ease the development of the adaptive IoT systems. First, based on the system requirements, a design model is specified. This model captures the system functionality and adaptations. The functionality is modelled using an extension of SysML [7] for capturing IoT concepts (i.e. SysML4IoT) [4], while the environment information needed by the system are modelled following a publish/subscribe pattern [8]. To model the system adaptations, a state machine approach is adopted [9]. The machine states represent the different configurations of the system, while the transitions represent its adaptation triggers (i.e. the pre-conceived changes to which the system can adapt) such as concrete hardware failures. Second, the design model is used to generate the system implementation. To do so, the design model is transformed to an IoT platform specific model, which is then used for generating the code. Finally, this code is deployed to a hardware platform to have a fully functioning software system.

To show the applicability of our approach, we have used it to develop a smart lighting system in the context of the S3P project (Smart, Safe and Secure Platform, see [10]). The system hardware platform has temperature and luminosity sensors, a light strip, an Edison board [11] that has a data distribution service (DDS), i.e., Java implementation of publish/subscribe paradigm [12], and a board with the MicroEJ operating system [13]. At runtime, our approach also supports the synchronization between the lighting system execution and its design, through reflecting the system state to its design model.

The remainder of the paper is organized as follows. A short description of related work is given in Section II. Our approach for designing an adaptive IoT software system and generating its implementations is described in Section III. In Section IV, we present our approach implementation. Finally, we conclude the paper in Section V.

II. RELATED WORK

The work introduced in this paper is related to developing IoT and adaptive systems. In the following, we describe related work from these two angles.

A. Developing IoT Systems

Patel and Cassou have proposed an approach to ease the development of IoT systems [1]. In this approach, the different system concerns are separated by a development methodology which is implemented through a development framework. This framework supports stakeholder actions to develop, deploy, and maintain an IoT system. It provides a set of modelling languages for specifying each development concern and to abstract the complexity related to the system heterogeneity and scale. The approach also provides automation through code generation that allows stakeholders to focus on the system's functional logic while the task mapping and linking techniques produce a device-specific code that results in a distributed system hosted by the individual devices.

The approach proposed by Costa et al. aims to design IoT systems and verify their QoS properties [4]. To model the system, they have introduced SysML4IoT, a SysML profile based on the IoT reference architecture model [14]. They also proposed a model-to-text translator called SysML2NuSMV. It converts an IoT model to a NuSMV (symbolic model verifier) program automatically [15]. Using the SysML4IoT, elements of

the system and their properties can be precisely identified by software engineers. The engineers are also able to verify system properties without modifying the system model (i.e., to keep its precision and alignment with the IoT reference model). The approach also does not require any profound knowledge about NuSMV programs from the engineers.

A model-driven approach for improving the reusability and flexibility of sensor software systems, called FRASAD (FRAmework for Sensor Application Development), has been proposed in [5]. In this approach, the system is described using a rule-based model and a Domain Specific Language (DSL). The FRASAD tool has a graphical user interface, code generation components, and supporting tools for helping the developers in modelling, realizing, and testing an IoT system. The evaluation of the approach showed that it enables fast development of IoT systems, and reduces the cost of dealing with their complexity.

The deployment of IoT systems to heterogeneous sensors, actuators, and business components is a challenging task. Thus, the Open TOSCA (Topology and Orchestration Specification for Cloud Applications) runtime has been proposed to ease this task [6]. This runtime platform supports the automated plan-based deployment and management of the IoT systems defined following the packaging format of TOSCA, which called CSAR (Cloud Service ARchive). This format enables the bundling of all system entities (e.g. management and business operations) in a self-contained manner.

For building interoperable and cross-domain IoT systems, a Machine-to-Machine Measurement (M3) framework has been introduced in [16]. The framework supports the IoT developers in semantically annotating machine-to-machine (M2M) data, and generating IoT software systems by combining M2M data from heterogeneous areas. Therefore, end users can receive the high-level information about their sensor data.

B. Developing Adaptive Systems

The *Rainbow* framework provides mechanisms to monitor a system environment, analyze it for initiating the adaptation process, select the required adaptation strategy, and act needed changes to the running system [17]. To capture the system reactions to context changes, they use a language called *Stitch*.

Sheng et al. proposed a model-driven approach to ease the development of context-aware services [18]. In this approach, they consider the service functionality as a single service, and the environment changes are used by a number of adaptation rules to adapt the output parameters of this service.

Zhang and Cheng inroduced an approach to create formal models of a system behaviour [19]. In this approach, the system adaptive behavior is separated from its non-adaptive behavior. This separation makes the system models easier to specify and verify. They used Petri-nets to capture the system adaptive behaviour, where the environment changes are used as guidance for the transition between system states.

The *SOCAM* project (Service-Oriented Context-Aware Middleware) has introduced an architecture to build adaptive software systems [20]. It uses a central server for gathering context information from distributed context providers. Such information is then processed, so that it can be used by the system functionality.

The *MUSIC* project is a component-based framework that is used to optimize the system overall utility in response to the context changes [21]. They have a quality of service (QoS) model that describes the system composition together with the relevant QoS dimensions, and how they are affected when the system changes from one configuration to another. The quality of service model is used for selecting a new configuration that has the best utility and is able to cope with the context changes.

Heaven et al. have developed an approach for adapting a software system in response to the environment changes while preserving its high level goals [22]. They use Labelled Transition Systems (LTS) to capture the system states and the environment situations.

Andrade et al. have proposed an approach to cope with the unanticipated changes of a system's adaptive behaviour [23]. They separate the system adaptation from its functionality, and represent the adaptation logic as a set of condition-action rules. These rules are constructed as a component-based system that can be changed at runtime.

Morin et al. proposed a technique to handle the exponential growth of the number of configurations that are derived from the system's high variability [24]. They combine model driven and aspect oriented approaches to cope with the complexity of adaptive software systems.

Limitations. The approaches discussed above for developing IoT systems do not enable their runtime adaptation in response to environment changes. However, IoT systems need to be adapted in response to hardware and software failures (e.g. a sensor failure, or unavailability of a function). In addition, the approaches proposed for developing adaptive systems target general software, which makes their adoption for IoT software development difficult and error prone due to the complexity introduced by IoT system characteristics such as their large scale and distributed nature. Furthermore, some of the existing approaches focus only on one development phase (e.g. analysis or deployment). Therefore, there is a need for an approach that fully supports the development of IoT software systems, and enables their runtime adaptation to cope with the environment changes.

III. Development of IoT Adaptive Systems

To ease the development of adaptive IoT systems, we have proposed a two-phase process (see Figure 1). In the first phase, a system model is created to specify the system functionality and its adaptive behavior. In phase two, code corresponding to the design model is generated. In the following, we describe the two phases in detail.

A. Modelling Adaptive IoT Systems

To model an adaptive IoT software system, two aspects of the system need to be captured: the system functionality and its adaptive behavior. The adaptive behavior specifies the system reactions to a number of anticipated context changes such as a sensor failure.

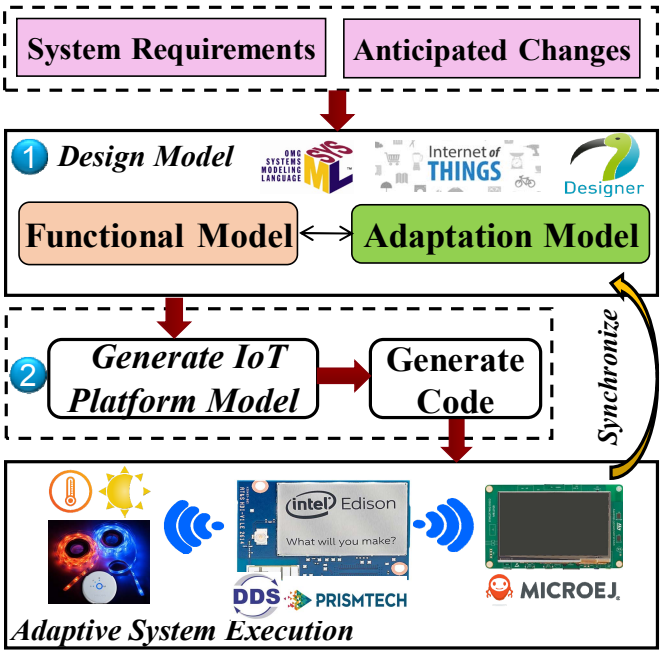

Figure 1. A Process for developing adaptive IoT systems

The System Functionality. The system consists of a set of functions that interact with each other to meet the end user requirements. To model such functionality, we first adopted a SysML extension that is called SysML4IoT [4] for capturing IoT concepts. The SysML4IoT profile is based on the IoT reference architecture model (see Figure 2) [14]. In the SysML4IoT, the IoT system (referred to as an *Augmented Entity*) consists of *Devices* and *Services*. A device is a hardware element which can be a *Tag* (to identify a *Physical Entity* such as a light strip id), a *Sensor* (to monitor a physical entity such as temperature), or an *Actuator* (to act on a physical entity such as turning lights on or off). A service is a software component to enable interactions between the system users (which can be *Human* or *Digital Artifact*) and the physical entities. The service also exposes a *Resource* that can be on a device or on the network.

Following the SysML4IoT profile, we model an IoT system as a composite structure that includes a number of functional services (e.g. light strip controller), a set of sensors to monitor environment information required by the functional services (e.g. current temperature and luminosity level), and a number of actuators to control parts of the system (e.g. switching the light strip to on or off).

Second, we adopted a publish/subscribe pattern to specify which information is provided by the sensors or required by the functional services [8]. In this pattern, the *Topic* concept is used to specify the required or provided information, while the *Data Reader* and *Data Writer* concepts are used for specifying readers and writers of the topics information.

Following the publish/subscribe paradigm, we modeled the environment information as a set of topics (e.g. temperature and luminosity), while the data reader concept is applied to a functional service (e.g. a light controller to read the luminosity level) and the data writer is applied to sensors (e.g. temperature sensor to provide the current temperature).

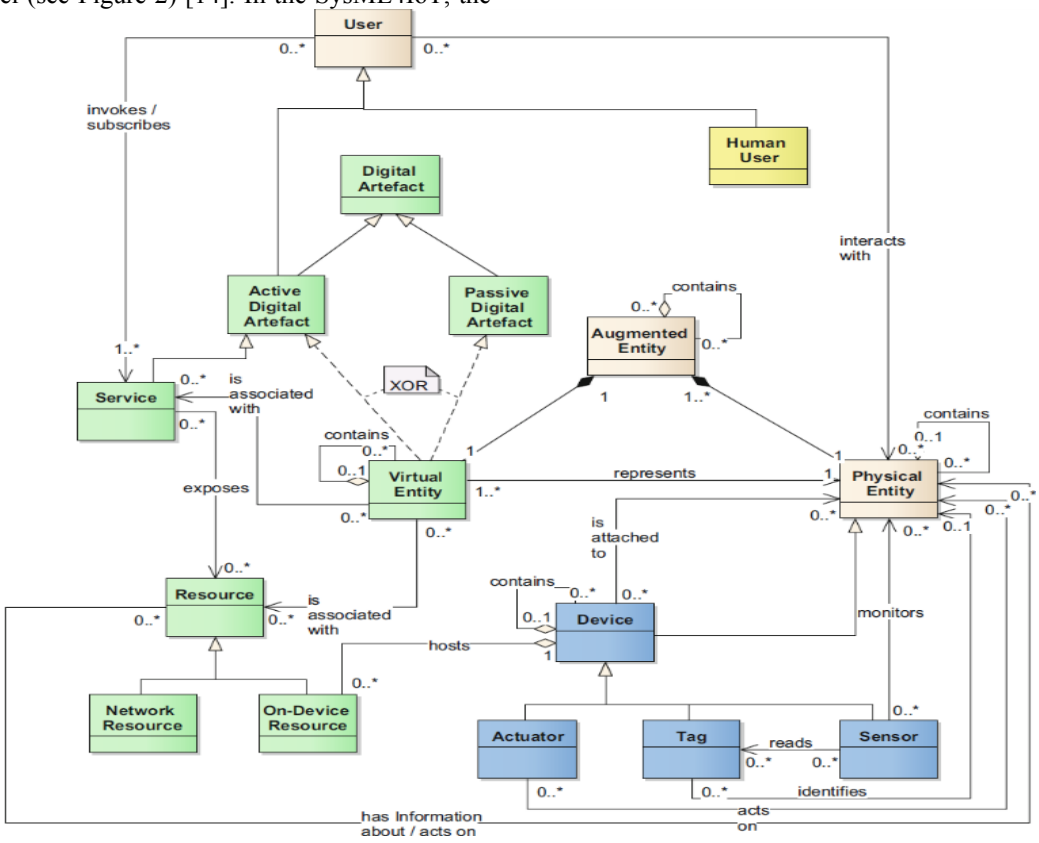

Figure 2. IoT domain model [14]

The System Adaptive Behavior. To adapt an IoT system in response to environment changes, we have introduced a system management component [25]. This component is responsible for the system switching from a configuration to another in response to an adaptation trigger. Therefore, we need to model adaptation triggers and different runtime configurations (states) of the IoT system. Both represent a runtime system state, and then we model both of them as UML instance specifications. These specifications describe component instances along with values for their attributes. Such a set is called a deployment plan, a term inspired from the CORBA component model. In our approach, a system deployment plan represents either an adaptation trigger or a configuration (state) [26].

In order to model the adaptive behavior of an IoT system, we adopted the state machine approach [27]. This technique makes adaptation policies easy to specify/understand, and it is useful for validation and verification purposes. In this machine, states are corresponding to the system configurations, while transitions represent adaptations between the configurations. Each transition is guarded and triggered through an adaptation event. For example, to cope with a sensor failure during a specific state, the system switches from its current state to a configuration that recovers from this failure.

B. Code Generation

The code generation is performed in two steps, where the high level model of the system is transformed to an IoT specific model which is later on transformed to a code.

Generate IoT Specific Model. Code generation is usually tied to the environment (platform) in which the system is going to execute. The IoT platforms are very large, and then it is very difficult for any approach to consider all of them. Thus, in our work, as a proof of concept, we generate an IoT system that can execute on a specific Java implementation of the publish-subscribe paradigm, which called data distribution service (DDS) [12]. The idea behind choosing such a platform is that the generated Java code can run on a very large number of devices, implying that our approach can have a wide applicability.

To generate code for a system modelled using SysML4IoT and publish/subscribe paradigm concepts, we automatically transform the system model to a model that is compatible with the target DDS platform [28]. In this transformation, a functional service (or an actuator) is transformed to a micro-service concept. This service includes initiate, start, and stop methods. The initiate method is responsible for initializing the functional service subscriptions to specific DDS topics. The start method prepares the topics data readers, while the stop method closes the opened data readers. In the same manner, the sensors are transformed to periodic-services that have initiate, start, and stop methods similar to the micro-services. These services also include "schedule" and "get duration" methods. The "get duration" method is used to determine how frequent the environment information are sensed, while the "schedule" method includes the specific strategy to get such information.

Generate Java Code. Based on an IoT model (generated as described above), the Java code is generated. In this process, first, for each functional service, a Java project is generated which includes a code corresponding to the functionality and the service connections with the sensors and actuators. This project is later compiled to a JAR file that can be deployed to the DDS Java implementation. In the same manner, Java projects for the sensors and actuators are generated.

Second, to generate the code corresponding to a DDS topic, an Interface Definition Language (IDL) file is generated which is then used by the idlj tool to generate Java code [29]. Third, to adapt the IoT software system at runtime, a system management service based on the adaptation state machine is created, which has the ability to identify the adaptation triggers and actions. It also contains code that is able to change the system configuration by starting and stopping the functional services at runtime.

IV. IMPLEMENTATION

In this Section, we use the concepts previously explained in Section III to model an adaptive lighting system, and generate its implementation. This case study has been conducted in the context of the S3P project, [10]. We also describe tools that support our approach.

A. Modelling an Adaptive Lighting System

As discussed previously, a model of an adaptive system should specify its functionality and adaptive behavior. Below, we discuss the model of the adaptive lighting system.

Modelling System Functionality. To create a design model of the adaptive lighting system following our approach, we use the Papyrus UML modeler [30]. The architecture model of the system functionality is shown in Figure 3. The model consists of a number of functions that are linked to each other through functional ports. In Figure 3, there are a number of functional services to control lights based on the environment information availability (e.g. luminosity and temperature controller). It also includes two sensors for monitoring the temperature and the luminosity level (i.e. temperature and luminosity sensors). Furthermore, there is a service to control the lights manually using a user interface.

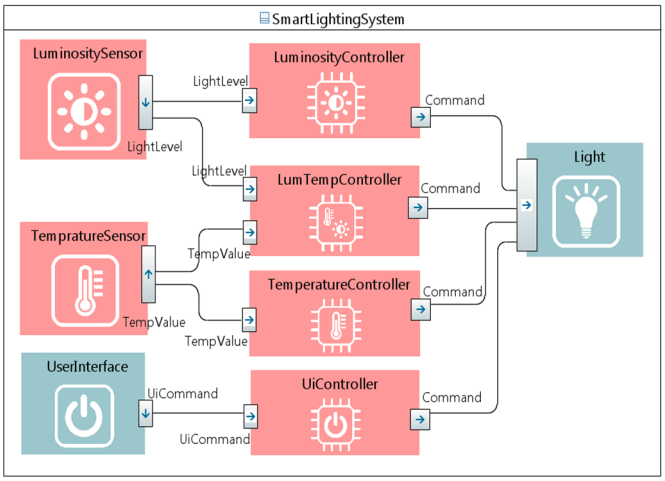

Figure 3. The architecture model of smart lighting system

Modelling the System Adaptive Behavior. To model the adaptive behavior of the system, both the adaptation triggers and the system states need to be specified. An example of an

adaptation trigger modeled as an instance specification is shown in Figure 4 (see the top part). For each component, a runtime system state is defined {e.g. Active or Inactive}. In the example, temperature controller, and temperature and luminosity sensors are in active state, while temperature and luminosity controller, and luminosity controller are inactive. A configuration to cope with the availability of the luminosity sensor is shown in Figure 4 (see the bottom part). The instance specification for each component is defined as <*Component Name, and State*>. Therefore, this configuration is defined as follows:

{< temperature controller, Inactive >, < temperature sensor, Active>, < luminosity sensor, Active>, < luminosity controller, Inactive>, < temperature and luminosity controller, Active>}

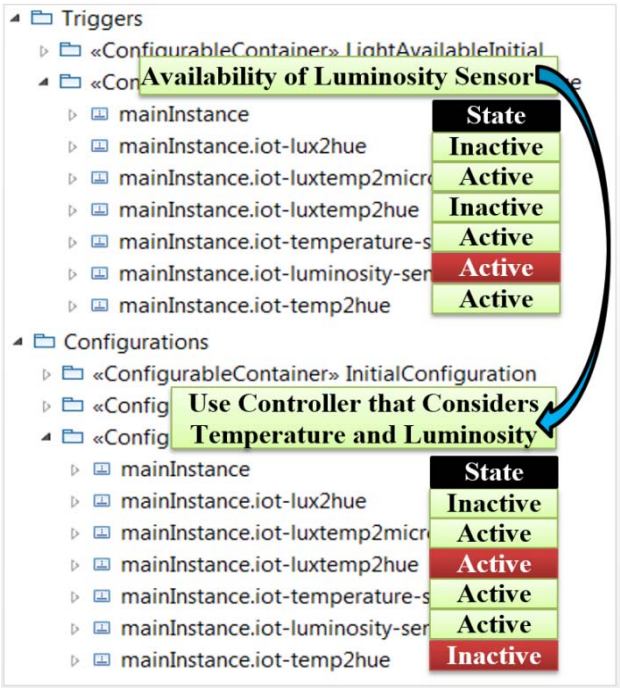

Figure 4. Specifying an adaptation trigger and a system response

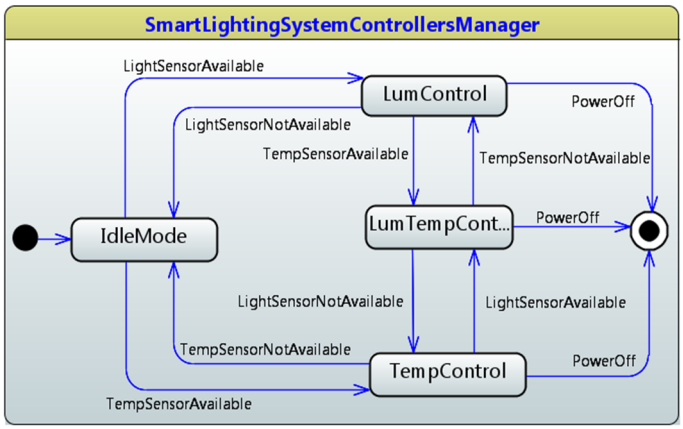

Figure 5. Adaptive behavior of the smart lighting system

To model the switching between the system configurations in response to the adaptation triggers, a state machine is created as shown in Figure 5. For example, in response to the availability of the luminosity sensor (the adaptation trigger specified on top part of Figure 4), the system adapts from its current configuration to another that takes the luminosity level into account (i.e. the system configuration shown at the bottom of Figure 4). This switching is specified by a transition (i.e. light sensor available) shown in Figure 5. In this transition, the state of the temperature controller is changed from "Active" to "Inactive", while the state of the temperature and luminosity controller is changed from "Inactive" to "Active".

B. *Code Generation*

Papyrus software designer [31] is an extension of Papyrus [30]. It provides code generation from UML models for a number of programming languages such C, C++, and Java, and supports the integration of new code generators. We have added support for the generation of adaptive IoT software systems to this tool. This enhancement is based on extensibility mechanism in the tool, notably the use of an extensible model-2-model transformation chain. The resulting tool enables the generation of an IoT specific model from the system's high level model.

Using this extension, first, code corresponding to a functional service such as luminosity controller is generated in two steps (see Figure 6). The top part of the Figure shows the service design model, while the bottom part shows its extended platform specific model (on the left), and the generated Java project (on the right). The generated Maven project can then be complied to create a JAR file (i.e. lux2hue.jar), which is later on deployed to a running instance of the DDS implementation [28]. Therefore, users and other services can interact with this service. Similar to generating the code for the functional services, code corresponding to system sensors and actuators is generated.

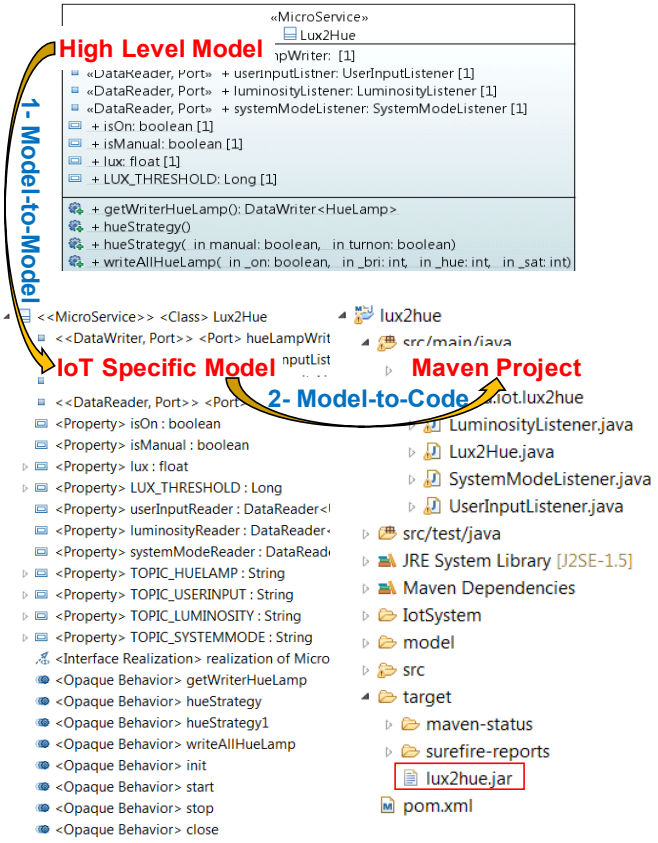

Figure 6. Code generation of an IoT functional service

Second, as discussed above, the information required by the functional services is modelled as a set of topics. Thus, for a functional service to use these topics, Java code corresponding to these topics needs to be generated. The topic data structure is modelled by the TopicStruct concept that includes a number of data items. For example, a temperature topic has id, value, and raw value (rvalue) as data items (see the top part of Figure 7). To generate the Java code corresponding to this topic, an IDL file representing this data structure is generated, which is then used by the idlj tool to generate the Java code as shown in the bottom part of Figure 7.

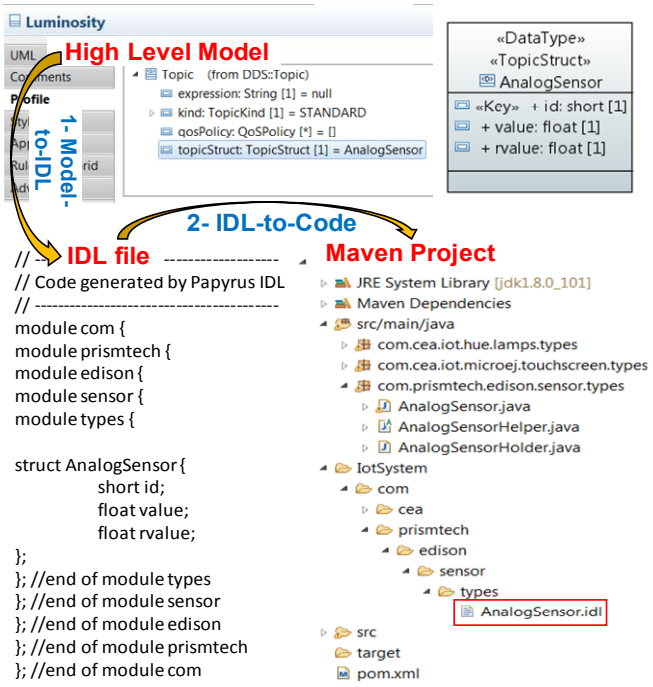

Figure 7. Generating code corresponding to the DDS topics

C. *System Deployment and Runtime Monitoring*

The hardware platform that we have used to deploy our IoT system generated above includes a light strip, temperature and luminosity sensors, an Edison board (a computer-on-module provided by Intel as a development system for IoT devices [11]) that has Java implementation of the data distribution service middleware running on it, and a board with the MicroEJ operating system [13] that is optimized for the IoT architectures (see Figure 8).

To have a fully functioning IoT system, the generated code is compiled and deployed to the Edison board. In addition, an application is developed for the MicroEJ board that enables the manual control of the light strip. Furthermore, to enable runtime monitoring of the system, we used a model animation framework developed in the context of the Papyrus Moka [32]. Using Moka, the runtime state of the deployed system is reflected to its design model by instrumenting the generated code with monitors. We also support the control of the running system via the model: functional services can be started and stopped from the design model.

In Figure 8, we show a snapshot of the case study during its operation. In this snapshot, the light controller, the luminosity sensor, and the luminosity controller are active. Therefore, in response to changes in luminosity level, the light strip is turned on or off. In Figure 8, the active components are also indicated by model elements that have the green colour (changed using the Moka animation framework).

Figure 8. Snapshot from the case study execution

V. CONCLUSION

In recent years, a growing number of objects have been connected to the internet to realize the idea of Internet of Things (IoT). A challenge is how to develop adaptive systems that can benefit from the IoT. In this paper, we have proposed a model-driven approach to develop such IoT systems. First, based on system requirements, a design model is created that captures the system functionality and its adaptation. The functionality is modelled by the SysML4IoT profile, while data needed by the system functionity is modelled following the publish/subscribe paradigm. To model the runtime adaptation, the state machine approach is adopted. Second, the model is used to generate system implementations. To do so, the high level design model is transformed to an IoT platform specific model, which is then used to generate the Java code. Finally, the generated code is deployed to the hardware platform of the system to make it fully functioning.

As a future work, firstly, we plan to extend our approach to enable the deployment of IoT systems to other IoT platforms, and to enable automatic system deployment to such platforms. Secondly, further evaluations will also be carried out to assess the approach robustness by applying it to a set of case studies.

REFERENCES

[1] P. Patel and D. Cassou, "Enabling High-level Application Development for the Internet of Things," *Journal of Systems and Software*, vol. 103, pp. 62-84, Januray 2015.

[2] A. P. Athreya, B. DeBruhl and P. Tague, "Designing for self-configuration and self-adaptation in the Internet of Things," in *9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, Austin, TX, 2013, pp. 585-592.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, 2015.

[4] B. Costa, P. F. Pires, F. C. Delicato, W. Li and A. Y. Zomaya, "Design and Analysis of IoT Applications: A Model-Driven Approach," in *IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing*, Auckland, 2016, pp. 392-399.

[5] X. T. Nguyen, H. T. Tran, H. Baraki and K. Geihs, "FRASAD: A framework for model-driven IoT Application Development," in *IEEE 2nd World Forum on Internet of Things* , Milan, 2015, pp. 387-392.

[6] A. Franco da Silva, U. Breitenbücher, K. Képes, O. Kopp, and F. Leymann, "OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker," in *Proceedings of the 6th International Conference on the Internet of Things (IoT'16)*, New York, NY, USA, 2016, pp. 181-182.

[7] (2017, June) SysML. [Online]. http://www.omgsysml.org/

[8] W. Kang, K. Kapitanova and S. H. Son, "RDDS: A Real-Time Data Distribution Service for Cyber-Physical Systems," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 2, pp. 393-405, May 2012.

[9] M. Hussein, R. Nouacer, A. Radermacher, "A Model-driven Approach for Validating Safe Adaptive Behaviors," in *19th Euromicro Conference on Digital Systems Design (DSD 2016)*, Limassol, Cyprus, August 31 – September 2, 2016.

[10] (2017, June) Smart, Safe and Secure Platform (S3P). [Online]. http://www.esterel-technologies.com/S3P-en.html

[11] (2017, June) Intel Edison Board. [Online]. https://software.intel.com/en-us/iot/hardware/edison

[12] (2017, June) Open Source Data Distribution Service (DDS). [Online]. http://www.prismtech.com/dds-community

[13] (2017, June) MicroEJ Platforms. [Online]. http://developer.microej.com/index.php?resource=JPF

[14] A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. van Kranenburg, and S. Langeand S. Meissner, *Enabling things to Talk*. Berlin, Heidelberg: Springer, 2013.

[15] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," in *Proceedings of the 14th International Conference on Computer Aided Verification (CAV '02)*, 2002, pp. 359-364.

[16] A. Gyrard, S. K. Datta, C. Bonnet and K. Boudaoud, "Cross-Domain Internet of Things Application Development: M3 Framework and Evaluation," in *3rd International Conference on Future Internet of Things and Cloud*, Rome, 2015, pp. 9-16.

[17] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure," *Computer* , vol. 37, no. 10, October 2004.

[18] Q. Z. Sheng, J. Yu, A. Segev, K. Liao, "Techniques on developing context-aware web services," *Int. Journal of Web Information Systems*, vol. 6, no. 3, pp. 185-202, 2010.

[19] J. Zhang and B. H. C. Cheng, "Model-based development of dynamically adaptive software," in *the 28th international conference on Software engineering*, Shanghai, China, 2006.

[20] T. Gu, H. K. Pung, and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *J. Netw. Comput. Appl.*, vol. 28, pp. 1-18, 2005.

[21] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz, "MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments," *Software Engineering for Self-Adaptive Systems*, vol. LNCS 5525, pp. 164-182, 2009.

[22] W. Heaven, D. Sykes, J. Magee, and J. Kramer, "A Case Study in Goal-Driven Architectural Adaptation," *Software Engineering for Self-Adaptive Systems*, vol. LNCS 5525, pp. 109-127, 2009.

[23] S. S. Andrade and R. J. de Araujo Macedo, "A non-intrusive component-based approach for deploying unanticipated self-management behaviour," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS '09)*, 2009.

[24] B. Morin, O. Barais, G. Nain, and J. Jezequel , "Taming Dynamically Adaptive Systems using models and aspects," in *the 31st International Conference on Software Engineering*, 2009.

[25] M. Salehie, and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst*, vol. 4, no. 2, pp. 1-42, 2009.

[26] M. Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language (3 ed.)," , Boston, MA, USA, 2003.

[27] B. Morin, O. Barais, J.M. Jezequel, F. Fleurey, and A. Solberg, "Models@ Run.time to Support Dynamic Adaptation," *Computer*, vol. 42, pp. 44-51, 2009.

[28] (2017, June) AgentV. [Online]. https://github.com/PrismTech/agentv

[29] (2017, June) idlj. [Online]. http://docs.oracle.com/javase/7/docs/technotes/tools/share/idlj.html

[30] S. Gérard, C. Dumoulin, P. Tessier, and B. Selic, "Papyrus: a UML2 tool for domain-specific language modeling," in *International Dagstuhl conference on Model-based engineering of embedded real-time systems (MBEERTS'07)*, Berlin, Heidelberg, 2007, pp. 361-368.

[31] W. Chehade, A. Radermacher, F. Terrier, B. Selic, and S. Gerard, "A model-driven framework for the development of portable real-time embedded systems. ," in *International Conference on Engineering of Complex Computer Systems*, 2011, pp. 45-54.

[32] J. Tatibouet, A. Cuccuru, S. Gérard, F. Terrier, "Towards a Systematic, Tool-Independent Methodology for Defining the Execution Semantics of UML Profiles with fUML ," in *MODELSWARD*, 2014.