Towards Integrated IoT Languages

Muzaffar Artikov
Software Engineering Department
Urgench branch of Tashkent
University of Information Technologies
Urgench, Uzbekistan
muzaffar.artikov@ubtuit.uz

Johannes Meier
Software Engineering Group
University of Oldenburg
Oldenburg, Germany
meier@se.uni-oldenburg.de

Andreas Winter
Software Engineering Group
University of Oldenburg
Oldenburg, Germany
winter@se.uni-oldenburg.de

***Abstract*—Internet of things (IoT) is a nascent technology that envisages connecting everything to the Internet and intends to improve people's lives by providing intelligent services ubiquitously. As IoT comprises heterogeneous technology stack it is difficult for developers to describe IoT components for further exploiting on IoT development activities. Model-driven engineering posses required capabilities to describe heterogeneous IoT components, separate stakeholders' concerns and one of the best candidates to handle the IoT development challenges.**

Therefore, there is a need to develop a set of languages, which models IoT systems from different perspectives and keep those models consistent. This paper intends to describe an integrated multi-viewpoint approach to develop IoT systems.

***Index Terms*—Internet of Things, Languages, Integration**

I. Motivation

Nowadays Internet is heavily loading with various types of smart devices to provide more intelligent services. These smart devices (or things) are made by exploiting sensors and actuators and equipped with networking capabilities enabling to exchange messages between each other and to form the Internet of things (IoT).

Smart technologies employ sensors and actors to increase the efficiency of services and processes, including environmental sustainability, energy efficiency, mobility, health care, safety, and security. ICT helps to optimize the processes, while IoT provides the platform for managing a multitude of small sensor, actor devices, home servers, etc.

In 1980's, students of Carnegie Mellon University published availability and temperature of soda of coke vending machine on a network [1]. This example is seen as the first IoT system. However, the term "Internet of Things" was coined by Kevin Ashton in 1999 in the context of supply chain management [2]. He defined it as follows: *"The Internet of Things is about installing sensors (RFID, IR, GPS, laser scanners, etc.) for everything, and connecting them to the internet through specific protocols for information exchange and communications, in order to achieve intelligent recognition, location, tracking, monitoring and management"* [3]. This definition points to the different parts of IoT systems. IoT systems combine various components of the following technology categories:

- **Hardware**. Various sensors/actuators, networking devices, embedded systems, gateways, sensor nodes, smart devices (e.g. Smart Home appliances) belong to this category. Hardware is required to sense data from the environment and to influence the environment by actions.
- **Software**. This category includes different software components, such as IoT operation systems, management software, drivers, libraries, third party services, virtual things, etc. Programming languages can be related this category as well. Software is required to control the hardware and to realize IoT use cases, which analyze sensed data and derive actions.
- **Network**. Various network topologies, network protocols, network connection types (e.g., wired or wireless) can be concerned in the scope of this category. Networks are required to allow communication between sensors sensing data, servers managing IoT use cases and actuators executing environmental actions.

Developing and maintaining a system that encompasses all of the above-mentioned components is quite complex. Therefore, developers often resort to ad-hoc solutions solving only the current problem and ignoring overall design decisions or architecture. These ad-hoc solutions might be unstable and their evolution would require more and more effort each time. For instance, if any device in the system needs to be replaced by another device, developers will have to reconstruct the entire system by rewriting some parts of the code, changing data exchange protocols, and so on.

During the development of IoT projects, different kinds of developers take part. Surely, these developers should own common knowledge pertaining the system in order to collaborate efficiently. For instance, if the hardware developer develops a new kind of sensor, then software developers should know about the capabilities of that sensor. Subsequently, the software developer should develop software fulfilling the assumption of the sensor's capabilities.

While developing IoT systems, most IoT development teams, firstly, install hardware infrastructure (sensors, actuators, embedded systems), and afterwards they interconnect hardware and software components including operation systems (OS), drivers, libraries. To interconnect components of IoT system they employ various network protocols depending on the type of IoT systems and taking into account the geographical width of the system. To control the whole system they still need software. While developing, they have to face various challenges that are the result of heterogeneous

technologies, which include various software, hardware, connectivity technologies and so on.

Model-Driven Engineering (MDE) methodology possess required concepts to face the IoT development and maintenance challenges. As first-class citizens in MDE, models employed to represent the IoT domain knowledge and allow separation of the domain concerns from technical implementation. One of the main benefits of MDE is through abstraction it handles heterogeneity challenge of IoT systems.

II. Related Work

Model-Driven Engineering provides reliable foundations and is considered as an enabling technology for advanced IoT applications. MDE is the modern day approach of software system development, which supports well-suited abstraction concepts for development activities. Though, MDE is mainly used in software development activities, it is possible to apply its techniques to other development domains such as IoT development as well.

There are several MDE approaches for developing IoT applications, e.g. the Sirius-based ThingML language [4]. The approach envisages expressive modeling of the IoT-based smart architectures, possibly with code generation. The motivation for model-driven development is to describe a system on a higher level of abstraction. This is usually done in UML and other languages by diagrams modeling specific aspects or views of model-driven architectures for smart systems. In ThingML, the state machine diagrams are used in several embedded domains to model the behavior of specific objects e.g. the discrete behavior of components. In the MDE paradigm of ThingML, the states of hardware components are managed by defining finite state machines.

MDE techniques are proposed to ease the development of IoT applications. In approach, applications are specified using high-level abstractions using models. These models are then used to produce deployable source code. For instance, PervML [5] enables developers to specify their software architectures at abstraction levels through a set of models.

Ciccozzi and Spalazzese introduced MDE4IoT [6], a MDE Framework supporting the modeling of Things and self-adaptation of Emergent Configurations of connected systems in IoT-based smart systems. As IoT systems consist of several connected software services and hardware components, there might be failures in performance of the overall system because of some non-responding devices. According to the article, in such cases the system should adapt to work and sustain without these inactive devices, and re-install and maintain its activities. In order to avoid such failures, MDE4IoT is meant to exploit the combination of a set of domain-specific modeling languages to achieve separation of concerns.

The research presented in [7] uses the MDE principles to build a holistic development methodology involving a common, semantically expressive abstraction model, to specify a smart space with its specific services. It proposes the Resource-Oriented and Ontology-Driven Development (ROOD) methodology, which improves traditional MDE-based tools through semantic technologies for rapid prototyping of smart spaces according to the IoT paradigm. In the framework of ROOD, the Smart Space Modeling Language (SsML) was developed based on UML, that defines a Domain Specific Model (DSL). It can be used for describing high-level behaviors, interactions and context information of the entire smart space. It further defines the processing aspects related to the sensing and actuating capabilities of the smart objects, as well as the context model they manage; moreover, encapsulate these concepts into RESTful resources.

Patel et. al. [8] presents a multi-stage model-driven approach for IoT application development, based on identification of the skills and responsibilities of the various stakeholders involved in the process. The approach uses configurable modeling languages that are customized for a particular stakeholder task and application area, where abstractions available to a specific stakeholder are generated from information provided by other stakeholders at previous stages. The approach is complemented by methods for generating code and mapping tasks that lead to the deployment of node-level code on composite devices.

III. Internet of Things-Case Study

In this paper simple Smart Conference Room scenarios (Figure 1) are considered. Smart Conference Room is equipped with two ”window blinds thing”s (brown colored) and ”smart bulb thing” (blue colored). Smart blinds can open and close by the request from the IoT gateway (red colored). Smart bulb switches on and off by the request as well. Thus, things communicate with cloud via IoT gateway and execute commands from the cloud. Gateway manages communications between blind things and smart bulb and cloud. A mobile phone communicates with cloud in order to send commands to the system. In the cloud there is a server application, which serves mobile application requests consequently by sending commands to gateway application.

In our example, things communicate with gateway via MQTT and gateway communicate with cloud using Websocket protocol. Mobile application communicates with the cloud via HTTP protocol.

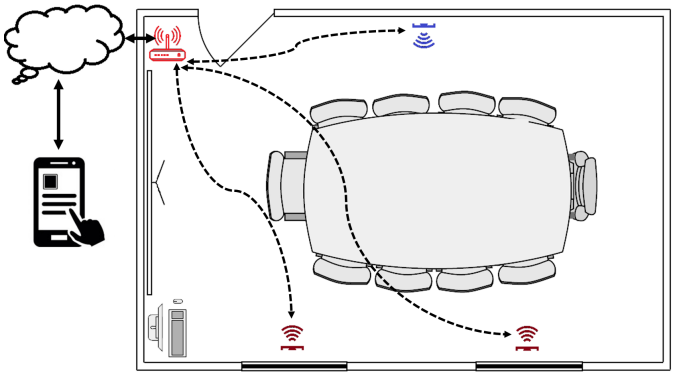

Fig. 1: Smart Conference Room

We will consider two scenarios within the room. First, in *Presentation* scenario blinds should be closed and bulb should be switched off. Second, in the *Discussion* scenario blinds should be opened and the light on if there is not sufficient light in the room (e.g. the sun is set). Execution of each scenario is conducted via the mobile application. In simple words, if user clicks on a "Presentation" button in the mobile application the *Presentation* scenario is set, and if user clicks on "Discussion" button the *Discussion* scenario is executed.

IV. IoT Languages

The central artifacts in MDE are models. A **model** is an abstraction of the system, which may already exists or is intended to exist in the future [9]. In software engineering *software models* are the documentation and implementation of software systems [10]. Models can be exploited to describe the system itself and as well as the individual elements of the system. Models focus on relevant aspects of the system's domain. Models serve to represent systems on different levels of abstraction. One of the benefits of model abstraction is that it helps to understand the system faster. Therefore, they can be used as a documentation of the system as well. Besides, models can be used for various purposes in software engineering, e.g. for model transformation, code generation, and testing. In model-driven software development models designed via Unified Modeling Language in many cases [11]. The structure and all elements which are allowed to be part of a model are defined by one *metamodel*. Each model conforms to one metamodel.

If a model is used to represent a system from a certain point of view, defined by the concerns of stakeholder, it is called a *view*. Than the corresponding metamodel of the model is called *viewpoint*.

The following primary model-driven abstraction languages (i.e., domain-specific languages) can be described to define their various elements in order to attain the greater abstraction levels of IoT systems:

Thing Description. Figure 2 contains the metamodel for *Things*. Things as the main components in IoT are uniquely identifiable, should have unique address in Internet and should have physical location. Things provide sensing and acting services, they can be in certain state. Within things occurs events which are relevant to the context of the whole system. Things may be physical or virtual (mock). Things can be named physical if they exist physically among IoT system's components. Virtual things are the things, which are non physically located within IoT system. They can be third party services like maps, weather services, etc. or mocks, which can play the role of things virtually without existing in reality.

Rule Description. "Things" in IoT may operate based on a simple control logic consisting of event triggers and actions. For developing the control systems in IoT, their rule systems can be defined using a rule description language (Figure 3). Simple rules can be described in "**if** `Event` **then** `Action`" manner. Actions may be executed sequentially (SequentialAction) or simultaneously (ParallelAction). These

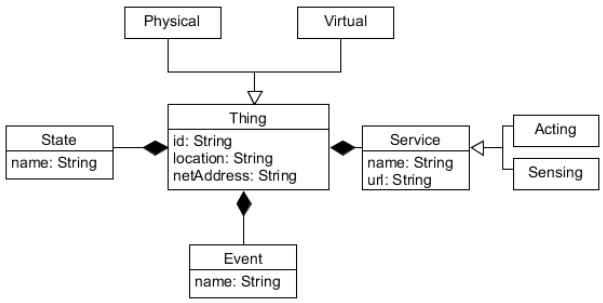

Fig. 2: Viewpoint for Things

actions incorporate other action instances. Sequential actions has property named "order" that defines the execution order of each sub-action.

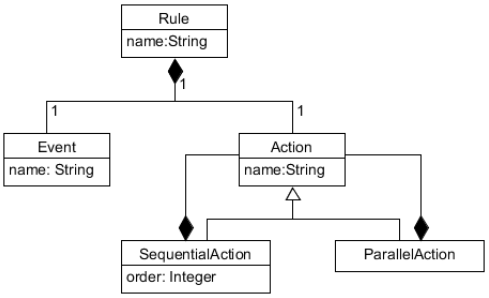

Fig. 3: Viewpoint for Rules

Communication Description. As IoT systems incorporate various individual components, they have to communicate with each other to fulfill certain tasks. Thereby, they perform specific tasks by the orchestrations of various soft- and hardware components based on control and data flow. Communication among various IoT components can be abstracted by a communication description language (metamodel described in Figure 4). An IoT network can incorporate another networks within itself. A network consists of communications between nodes. Herewith, it should be able to describe information such as the connection link (e.g., wired, wireless), communication protocol, data exchange formats and etc. Worth to mention that there is a relation between connection protocols and connection links, as some protocols work with a wireless link, some work with wired linked connections.

These are the minimal set of description languages for abstracting IoT systems in order to achieve a higher level of efficiency in IoT development and maintenance. Certainly, there can be more languages to describe other aspects of IoT systems. Each language represents an IoT system from a certain point of view. Models designed in these views should serve as an existing asset in other views. For instance, if in *Things* view there were described *N* things, in *Communication* view there may also be described communications between these *N* things. In *Rules* view *Acting services* from *Things* view

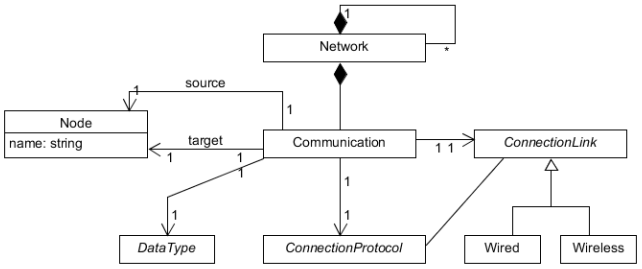

Fig. 4: Viewpoint for Communication

can be used as an *Action* instance. Therefore, the integration of views is required and sketched in the next section.

V. Integration

The final goal is to develop *one* working IoT system. To manage their complexity and heterogeneity, *several* different languages are required to describe different aspects of the IoT system regarding the different concerns of different stakeholder. Therefore, Section IV presented viewpoints for Thing, Rule, Communication. These viewpoints model IoT systems from different perspectives, so that different IoT experts can model independently from each other their views. In order to form the final IoT system as a whole, the three viewpoints have to be integrated.

As detected in Section IV, the three viewpoints describe overlapping and depending information. Therefore, the views managed by different stakeholder contain overlapping and depending information. For the three IoT languages, they are depicted in Figure 5: In the Rules, only events and actions are

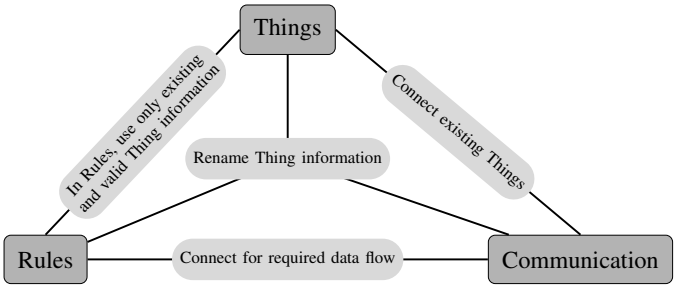

Fig. 5: Consistency issues between the three IoT languages

usable which are defined by the Things. The Communication must connect only existing Things. Since the Rules require data flow between sensors and actuators, this data flows has to be enabled by the Communication. Renaming of Things and their properties has to be renamed in all views.

These consistency rules hold for the three IoT views and are crucial to eliminate misapprehensions between developers and experts of IoT systems. It is error-prone and time-consuming to ensure such consistency rules manually for huge IoT systems with lots of different stakeholder. To enable different stakeholder to work together effectively, the consistency between the different views have to be ensured *automatically*.

To ensure consistency in multi-view environments, several approaches are currently under development [12] and can be reused to integrate the introduced IoT languages. Since the three IoT languages already exist in form of their viewpoints (Section IV) and are used to specify the Smart Conference Room example in form of views (Section VI), the approach for integration has to support already existing viewpoints and views. Since cross-viewpoint analyses and code generation for the whole system are goals of the integration, the approach for integration has to provide one explicit metamodel and conforming model representing the whole IoT system. Following these two main requirements, MoConseMI [13] is selected for the integration of IoT languages.

The use of MoConseMI to integrate the three IoT languages in form of the viewpoints is sketched in Figure 6: The

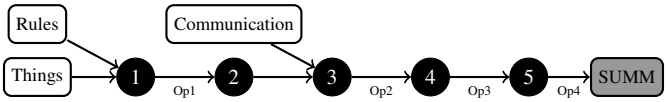

Fig. 6: Operator-based Integration of IoT languages

existing viewpoints for Things, Rules and Communication are combined technically into one whole viewpoint at ❶ and ❸. Afterwards, operators are selected and configured to integrate the viewpoints contentwise regarding their overlapping information, sketched here at ❷, ❹ and ❺, by improving the viewpoints step by step. At the end, the SUMM represents all concepts for IoT systems as "Single Underlying MetaModel" in an integrated and optimized way. This chain of operators is executed at runtime to propagate changes made by one developer inside one view into all other views. Therefore, the consistency between all views is ensured automatically by the operators improving viewpoints and views at the same time. The final SUMM represents the whole integrated IoT system.

VI. Application

In Section IV defined three languages (viewpoints) to describe IoT systems. The running example, which is sketched at Section III can be modeled with the corresponding views of these viewpoints. Thus, an example would be considered from Things, Communication and Rules viewpoints.

If consider our example from the *Things* perspective, there are four things exist in our system: two window blind things, a smart bulb thing and a smartphone. All four things are *Physical* things. Window blind things provide two services: `OpenBlindActing` and `CloseBlindActing` and they have four states: `OPEN`, `OPENING`, `CLOSING` and `CLOSED`. In window blinds there may occur `OnOpen` and `OnClose` events. Smart bulb thing provides `SwitchOnActing` and `SwitchOffActing` services and it has two states: `ON` and `OFF`. Within a smart bulb may occur `OnSwitchOn` and `OnSwitchOff` events. A smartphone tend to select one of two scenarios: Presentation and Discussion. It provides `SelectPresentationActing` and `SelectDiscussionActing` services and it has

three states: DEFAULT, PRESENTATION, DISCUSSION. Smartphone has OnPresentationSelect and OnDiscussionSelect events, which occur when one of the two scenarios selected. In Things view each thing is named and the network addresses are defined.

After describing things the communication network can be described. The network description can be considered as a description of the configuration between nodes. In our example nodes can be things, cloud and gateways. The defined names and network addresses from Things view would be exploited as Node instances to describe communication.

In Rules view for each scenario a model can be described. *Presentation* and *Discussion* scenarios can be described as Rule instances as following in a textual notation.

```
RULE Presentation:
IF smartPhone.onPresentationSelect
THEN windowdBlind1.CloseBlindActing AND
windowBlind2.CloseBlindActing AND
smartBulb.switchOffActing;
END RULE;

RULE Discussion:
IF smartPhone.onDiscussionSelect
THEN windowdBlind1.OpenBlindActing AND
windowBlind2.OpenBlindActing AND
smartBulb.switchOnActing;
END RULE;
```

In a rule description each rule can be named. In our example there are "Presentation" and "Discussion", initiated by the keyword RULE. Each rule ended with END RULE keyword. If describe first rule instance, there is an event named onPresentationSelect, which occurs when user chooses Presentation scenario. This event is generated by *smartPhone* thing instance and is presented in IF section of the rule. In THEN section defined actions that should be executed, when an event occurs, which are OpenBlindActing of blinds and SwitchOffActing of the smart bulb thing.

After using the developed IoT languages from Section IV to describe the Smart Conference Room, there is no more effort for the integration: Since the integration was specified once as indicated in Section V to integrate the three IoT languages in general, this integration can be directly used for each IoT system modeled with these three IoT languages, including the Smart Conference Room.

VII. Conclusion

This vision paper presents a model-driven approach for the development of Internet of Things systems. As IoT systems comprise various aspects it is proposed to develop different aspects separately and integrate them to provide one whole IoT system. Thing, Rule and Communication were identified as important aspects to describe for IoT systems and appropriate viewpoints were developed in form of metamodels. Each viewpoint describes a single aspect of IoT systems, which allows separation of concerns and helps to manage complexity. To ensure consistency between these aspects, the IoT viewpoints are integrated into a Single Underlying MetaModel representing all concepts of IoT systems.

The identified viewpoints cover not all aspects of IoT systems: Therefore, future work is to identify viewpoints to model missing properties of IoT systems, e.g. used data of IoT systems, as well as to extend the existing viewpoints.

Future work is also to develop a concrete syntax for each language: Since Section IV developed only metamodels to describe the possible elements of the IoT languages, notations how to represent these elements in usable way for stakeholders is still missing. Metamodels and models for the domain-specific IoT languages can be developed using the Eclipse Modeling Framework (EMF) [14]. On top of EMF, Xtext [15] allows to develop a concrete textual syntax, as it is sketched for the Rule language in Section VI. The Sirius framework [16] enables to develop a concrete graphical syntax.

References

[1] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of things (iot): A literature review," *Journal of Computer and Communications*, vol. 3, no. 05, p. 164, 2015.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[4] F. Fleurey, B. Morin, and A. Solberg, "A model-driven approach to develop adaptive firmwares," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2011, pp. 168–177.

[5] E. Serral, P. Valderas, and V. Pelechano, "Towards the model driven development of context-aware pervasive systems," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 254–280, 2010.

[6] F. Ciccozzi and R. Spalazzese, "Mde4iot: supporting the internet of things with model-driven engineering," in *International Symposium on Intelligent and Distributed Computing*. Springer, 2016, pp. 67–76.

[7] I. Corredor, A. M. Bernardos, J. Iglesias, and J. R. Casar, "Model-driven methodology for rapid deployment of smart spaces based on resource-oriented architectures," *Sensors*, vol. 12, no. 7, pp. 9286–9335, 2012.

[8] 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, 2015.

[9] A. R. Da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Computer Languages, Systems & Structures*, vol. 43, pp. 139–155, 2015.

[10] A. G. Kleppe, J. Warmer, J. B. Warmer, and W. Bast, *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[11] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[12] J. Meier, H. Klare, C. Tunjic, C. Atkinson, E. Burger, R. Reussner, and A. Winter, "Single underlying models for projectional, multi-view environments," in *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019)*, 2019.

[13] J. Meier and A. Winter, "Model Consistency ensured by Metamodel Integration," *6th International Workshop on The Globalization of Modeling Languages, co-located with MODELS 2018*, 2018.

[14] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, "Emf: Eclipse modeling framework 2.0," 2009.

[15] M. Eysholdt and H. Behrens, "Xtext: implement your language faster than the quick and dirty way," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM, 2010, pp. 307–309.

[16] V. Viyović, M. Maksimović, and B. Perisić, "Sirius: A rapid development of dsm graphical editor," in *Intelligent Engineering Systems (INES), 2014 18th International Conference on*. IEEE, 2014, pp. 233–238.