Modeling SOA-Based IoT Applications with SoaML4IoT

Bruno Costa
Instituto de Matemática – IM
Programa de Pós-Graduação em Informática – PPGI
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
brunocosta.dsn@gmail.com

Paulo F. Pires, Flávia C. Delicato
Instituto de Matemática – IM
Programa de Pós-Graduação em Informática – PPGI
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
{paulo.f.pires, fdelicato}@gmail.com

***Abstract*—** A common feature of Internet of Things (IoT) applications is heterogeneity, regarding network protocols, data formats, hardware and software platforms. Aiming to deal with this heterogeneity, several frameworks have applied the Model-Driven Development (MDD) to build IoT applications. On the software architecture point of view, the literature has shown that the Service-Oriented Architecture (SOA) is one of the most suitable style to address the interoperability of heterogeneous entities composing these solutions. However, to the best of our knowledge, there is no modeling language tailored for the design of SOA-based IoT applications. This paper aims at bridging this gap by introducing SoaML4IoT. The modeling language extends the SoaML, the standard graphical modeling language proposed by the OMG for designing systems that follow the services approach.

Keywords— internet of things, services-based, system

I. INTRODUCTION

Heterogeneity is common in Internet of Things (IoT) systems [1]–[3], regarding network protocols, data formats, hardware and software platforms. This heterogeneity poses new challenges to develop IoT applications by following traditional approaches, such as the node-centric programming [4], which has been widely used in the context of many IoT systems, notably the ones based on Wireless Sensor Networks (WSN). Instead, aiming to deal with the heterogeneity, several frameworks (for example, [5]) applied the Model-Driven Development (MDD) [6] to build IoT applications. MDD is an approach that uses models as the primary artifact of the development process. The models are created by using Domain Specific Modeling Languages (DSML), or DSL for short, which provide high-level domain abstractions expressed through inter-connected symbols.

From the software architecture point of view, the heterogeneous nature of IoT requires structuring the applications by considering the interoperability of different software components and hardware devices. This critical issue has led the researchers to investigate architectural patterns and styles that most favor the communication of the inter-connected entities in an IoT system. In this sense, many studies (e.g., [7]) have shown that the Service-Oriented Architecture (SOA) [8] is the most suitable style to address the interoperability of IoT applications. In the SOA approach, the devices' capabilities (e.g., sensing, actuating) are provided as software services through well-defined interfaces.

However, although the literature has identified that the SOA style is one of the most suitable approaches for addressing the interoperability in the IoT domain, concepts related to the services approach are not figured as first-class citizens in the existing MDD frameworks for IoT, such as UML4IoT [9], ArchWiSeN [10], COMFIT [5], Patel & Cassou [11], FRASAD [12], IoT Link [13], Srijan [4], ThingML [14], ELIoT [15], and IMPReSS. This is a critical problem since even crucial but straightforward design questions related to SOA style cannot be answered by the model, such as, "What are the services, and by which interface they are exposed?", "Who are the provider/consumer participants that are part of the application?", "How the participants interact with each other?", "What is the choreography of the services?", or "What is the protocol used by the interfaces." A DSL that does not provide concepts to allow answering these design questions possibly hampers the specification of SOA-based IoT applications.

Aiming to tackle such a problem, this paper proposes a DSL, named as *SoaML4IoT*, tailored for designing SOA-based IoT systems. The DSL was built on the Service oriented architecture Modeling Language (SoaML) [17], the Object Management Group (OMG) standard for designing SOA solutions. SoaML consists of an extension of UML (i.e., a UML profile), which was conceived through an extensive collaboration of experts from academia and industry. As stated by the SoaML specification, the modeling language focuses on the basic service modeling concepts, and the intention is to use it as a foundation for further domain extensions. Therefore, based on proven domain models for IoT (e.g., IoT-ARM [7], WSO2[18]) we extend such a language, allowing the precise specification of the entities composing IoT solutions that follow the services approach. A comparison with existing MDD frameworks has shown that SoaML4IoT is more expressive for specifying SOA-based IoT applications than the languages proposed by such frameworks, allowing to conceive models that answer the design questions elicited above.

The rest of this paper is organized as follows. Section II presents the conceptual foundations of this work. Section III introduces the SoaML4IoT. In Section IV we evaluate our proposed language. Section V provides final remarks and future directions.

II. PRELIMINARIES

In this Section, we present an overview of the basics of DSL engineering, with focus on the elements on which our approach is built on, that is, UML Profiling and SoaML.

A. UML Profiling

The Unified Modeling Language (UML) [20] is a general-purpose modeling language proposed by OMG for visualizing, specifying, constructing and documenting artefacts of software-intensive systems. The UML provides a set of mechanisms (e.g., stereotypes and tagged values) to extend its elements to a domain. These mechanisms are grouped into the so-called UML Profiles. A profile can be seen as a graphical notation for a given DSL created on the top of UML [21]. When creating such a notation, the concepts from the metamodel are conceived as stereotypes, each one associated with the UML element they extend, and their

properties become tagged values. Stereotypes are UML classes with a name, representing the domain concept, and a set of tagged values, which represents the domain attributes.

B. SoaML

SoaML [17] is the standard graphical modeling language proposed by the OMG for designing systems that follow the services approach. It consists of a metamodel and a UML profile for the specification of services within SOA style. An extensive example of modeling with SoaML is given by Elvesæter and colleagues [22]. In the following, we present key concepts of the SoaML metamodel. In parenthesis, we show the UML element that SoaML profile extends.

Participants (Class) are entities that provide or use services. *Capability* (Class), refer to functions required by stakeholders and provided by a participant through a service. A *Service* (Port) denotes value delivered to another through a well-defined interface. *Provider* and *Consumer* (Interface) are roles of participants that provide or consumes services, respectively. The description of how the participants interact to provide or use a service can be specified as *Service Contracts* (Collaboration). It specifies the roles each participant plays in the service – provider or consumer – and the choreography of the service, that is, what information is sent between the provider and consumer and in what order. When specifying the choreography of a given service contract, any UML behavior specification can be used, such as interaction and activity diagrams. Finally, the *Services Architecture* (Collaboration) describes how participants work together for a purpose by providing and using services expressed in the service contracts.

III. SoaML4IoT

In this Section, we introduce our proposed framework. The modeling language was designed by following the methodology presented by Brambilla and colleagues [6].

A. Abstract Syntax

The SoaML4IoT metamodel is depicted in UML Class diagram of Figure 1.

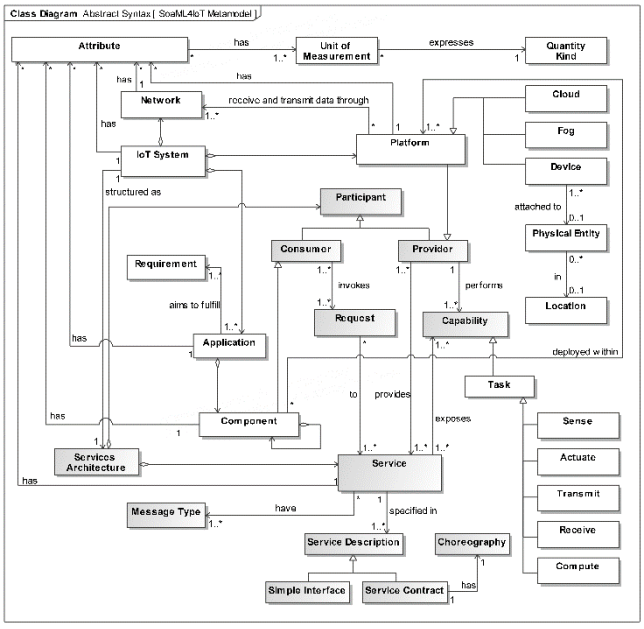

Figure 1. The SoaML4IoT Metamodel

Based on the analysis of the IoT literature (specially, [7], [18], and [19]), it extends the SoaML metamodel (gray classes) with fundamental concepts of the IoT domain (white classes). In the following, we explain each concept. The elements of SoaML was already introduced in subsection II.A.

An IoT System is a cyber-physical system composed of Platforms, Applications, and Networks. A Platform refers to computer nodes, which can be of type Cloud, Fog, or Device. Nodes located in the cloud provide a set of resources (i.e., processing, storage), which can be provisioned and paid on demand. Fog nodes are computers that act as near- devices computing/storage resources or as bridges to connect the devices to the cloud. A Device is a computer attached to Physical Entities, which can be anything of the real world from objects and cars to animals and human beings. A physical entity lies in a specific geographic location.

The devices enhance physical entities with sensing, actuating, communication, and computing capabilities. Sense tasks aim to collect data from physical entities (e.g., car speed, human body's temperature) or the environment it is inserted into (e.g., room's temperature or lighting level); Actuate tasks can affect the physical realm (e.g., turn on/off a heater, sound alarm); Transmit and Receive tasks regard the communication of the device; finally, the Compute task refers to the processing capability of the device. The platforms communicate with each other and eventually with the Internet by using wired or, more often, wireless networks. A Network refers to the set of nodes and links providing the communication path by with the platforms receive and transmit data. An Application refers to a set of specialized algorithms that request services and process data aiming at fulfilling user-defined requirements. An Application is composed of one or more Components, which are software units that can be deployed within platforms. Note that the metamodel represents the IoT applications as a set of components instead of a monolithic structure.

In SoaML4IoT metamodel, the Capabilities performed by the providers are specialized as Tasks (sense, actuate, transmit, receive, and compute). In turn, the Providers are specialized as Platforms. Components act as Consumers, requesting the required tasks exposed through the devices' services. In other words, devices attached to physical entities expose their provided tasks through services, which are consumed by application's components aiming at fulfilling the stakeholders' requirements.

An essential concept of the SoaML4IoT metamodel is the Attribute. In SoaML4IoT metamodel, we specialize such a concept. Besides representing properties of classes, an attribute is refined as a property that may be useful for the architectural decision-making process. Therefore, an attribute is a property that may be predicted or estimated at design-time to help to answer a design question. The reasoning behind this objective is to avoid creating too-detailed models, which are hard to manage [94]. Thus, the modelers would specify only attributes that are strictly necessary to answer design questions. The elements of the metamodel that have such specialized attribute are IoT System, Platform, Network, Application, Component, and Service, which are the main structural elements of a SOA-based IoT solution. The specialized attribute is specified in the metamodel as a first-class citizen. Thus, the concrete representation of these elements is realized in the UML profile as tagged values. An

attribute may have a Unit of Measurement (e.g., watt) expressing a Quantity Kind (e.g., electric power).

B. Concrete Syntax

The concrete syntax of SoaML4IoT consists of an UML profile extending the UML Port, Class, Dependency, Interface, Association, and DataType (Figure 2). Such extensions realize the concepts with the same name from the SoaML4IoT metamodel.

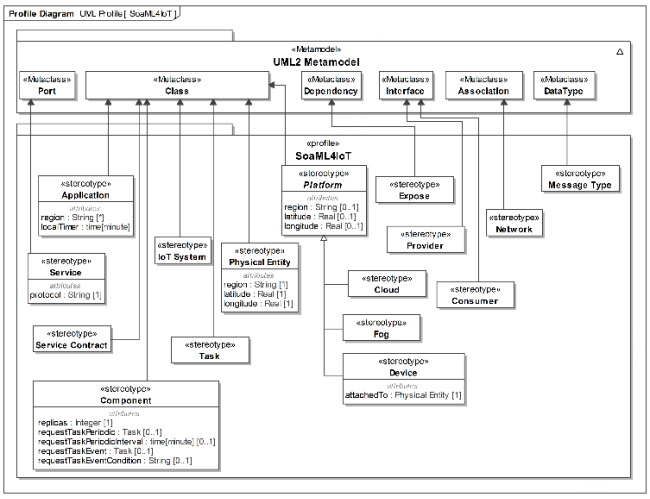

Figure 2. The SoaML4IoT Concrete Syntax

As introduced in the SoaML4IoT metamodel, the *Attribute* concept is realized in the concrete syntax as tagged values by providing information that is necessary for the architectural decision-making process. In the presented version of SoaML4IoT, we provide essential properties to allow representing periodic and event-based IoT applications.

The *Service* has the attribute *protocol*, aiming to specify its application-level protocol (e.g., HTTP, CoAP). The *Application* also has one attribute, *region*, aims to specify the geographic location of the target environment (as requested by the application), from which the sensing data will be collected and/or actuation tasks will be performed upon.

The *Component* stereotype has the attributes for specifying periodic and/or event-based applications. Recall that an application is composed of one or more components. Thus, although the periodic or event-based model refers to the application, the components are the responsible for actually performing that required requests. The attribute *requestTaskPeriodic* models the task to be requested at regular time intervals, which is specified in the attribute *requestTaskPeriodicInterval*. On the other hand, the attribute *requestTaskEvent* defines the required task that aims to be requested when an event occurs. Such an event is specified in SoaML4IoT textually, in the attribute *requestTaskEventCondition*.

The *Physical Entity* stereotype allows specifying its geographic *region* through the *latitude*, *longitude* attributes. The *Platform* (abstract) also allows specifying its geographic *region* through the *latitude*, *longitude* attributes. The Device stereotype has only *attachedTo* attribute, which models the physical entity equipped with the device.

IV. Evaluation

The evaluation aims at analyzing the expressivity of the abstract syntax (metamodel) and qualitative aspects related to the concrete syntax of SysML4IoT. In the following, we present such analysis.

A. Abstract Syntax Analysis

The expressiveness requirement is about the degree to which a language allows its users to capture phenomena in the domain. [23]. Thus, the more of the domain of interest can be represented by the language, the greater expressive such language has. In light of the framework for measuring expressiveness in conceptual modeling proposed by Patig [24], we assess the expressiveness of IoTDraw metamodel comparing it with the related work, that is UML4IoT [9], ArchWiseN [10], COMFIT [5], Patel et al. [11], FRASAD [12], IoT Link [13], , SensorML [25], and Srijan [4].

In the Patig's framework, D_1 and D_2 denote separate metamodels, whose expressiveness is to be compared. The models of these descriptions are represented by $M(D_1)$ and $M(D_2)$, respectively. Two descriptions D_1 and D_2 are *equally expressive* if every model of D_1 is also a model of D_2 and vice-versa, i.e., $M(D_1) = M(D_2)$. On the other hand, a description D_1 is *more expressive* than a description D_2 if all models of D_2 are proper subsets of the models of D_1, that is, $M(D_1) \supset M(D_2)$. In other words, D_1 provides all elements to create $M(D_2)$, including the elements to create $M(D_1)$. In our evaluation, we compare the expressiveness of $M(\text{IoTDraw})$, $M(\text{UML4IoT})$, $M(\text{ArchWiseN})$, $M(\text{COMFIT})$, $M(\text{Patel et al.})$, $M(\text{FRASAD})$, $M(\text{IoT Link})$, $M(\text{SensorML})$ and, $M(\text{Srijan})$.

The expressiveness of the metamodels are verified according to a set of statements that the models can represent. By statement, we mean a syntactic expression and its meaning. As introduced by the Patig's framework, first-order predicate logic [26] can be used as a common description language to represent the statements, because of the comprehensible definition of its models. Thus, each statement is modeled as an atomic formula, representing a description of the domain of interest.

To conceive the list of statements to compare the expressiveness of SoaML4IoT with other DSLs, we analyze proven reference models of IoT and SOA. Reference models represent a set of essential concepts and relationships between them concerning a specific domain abstractly [27]. It is based on a small number of unifying concepts and can be used as a basis for explaining a given domain. The objective was to identify key concepts covered by such reference models and represent them as first-order logic statements. For the IoT domain, we analyze the IoT Architectural Reference Model (IoT-ARM) [7], the WSO2 IoT Reference Architecture [18], and the IEEE Standard for an Architectural Framework for Internet of Things (P2413) [19]. Considering the SOA style, we analyze the OASIS [28] and SOA Ontology [29].

The set of statements representing essential concepts and relationships in the IoT domain is listed in Table 2. Besides identifying key elements of the IoT domain, these statements represent the expressivity of the metamodel to allow answering design questions related to the SOA style

Table 1. Statements for SOA-Based Modeling Frameworks

#	Statement	Description
S1	$iotSystem(i)$	The IoT system i
S2	$platform(p)$	The platform p
S3	$cloud(p, c)$	The platform p is of type cloud c
S4	$gateway(p, g)$	The platform p is of type gateway g
S5	$device(p, d)$	The platform p is of type device d
S6	$physicalEntity(e)$	The physical entity e
S7	$location(e, l)$	The location l of physical entity e
S8	$isAttachedTo(d, e)$	The device d is attached to the physical entity e
S9	$task(d, t)$	The task t performed by the device d
S10	$network(n)$	The network n
S11	$attribute(n, a)$	The attribute a of the network n
S12	$application(o)$	The application o
S13	$attribute(o, a)$	The attribute a of the application o
S14	$consumer(o)$	The role of the application o
S15	$provider(d)$	The role of the device d
S16	$service(t, s)$	The task t is exposed by the service s
S17	$attribute(s, a)$	The attribute a of the service s
S18	$contract(s, c)$	The contract c of the service s
S19	$choreography(s, h)$	The choreography h of the service s
S20	$messageType(s, m)$	The message type m of the service s

After introducing the statements, in Table 2 we compare the expressivity of the existing modeling frameworks for IoT with focus on their capability to represent SOA-based aspects. In our analysis, we search for concepts that are firs-class elements in the modeling languages.

Table 2. Expressiveness Comparison

#	IoTDraw	[9]	[10]	[5]	[11]	[12]	[13]	[25]	[4]
S1	✓	✓	✓	✓	✓	✓	✓	✓	✓
S2	✓	✓	✓	✓	✓	✓	✓	✓	✓
S3	✓	-	-	✓	-	-	✓	-	-
S4	✓	✓	✓	✓	✓	✓	✓	✓	✓
S5	✓	✓	✓	✓	✓	✓	✓	✓	✓
S6	✓	✓	-	-	-	-	✓	✓	-
S7	✓	-	-	-	✓	✓	-	✓	-
S8	✓	-	-	-	-	✓	-	✓	-
S9	✓	✓	✓	✓	✓	✓	✓	✓	✓
S10	✓	✓	✓	✓	✓	✓	✓	✓	✓
S11	✓	✓	✓	✓	✓	✓	✓	✓	✓
S12	✓	✓	✓	✓	✓	✓	✓	✓	✓
S13	✓	✓	✓	✓	✓	✓	✓	✓	✓
S14	✓	-	-	-	-	-	-	-	-
S15	✓	-	-	-	-	-	-	-	-
S16	✓	-	-	-	-	-	-	-	-
S17	✓	-	-	-	-	-	-	-	-
S18	✓	-	-	-	-	-	-	-	-
S19	✓	-	-	-	-	-	-	-	-
S20	✓	-	-	-	-	-	-	-	-

Table 2 shows that most of the modeling frameworks provide concepts to represent devices, tasks, networks, and applications. However, concepts related to physical entity, location and the attachment of devices to physical entities are addressed by only part of the frameworks. Considering the concepts regarding SOA, none of the analyzed approaches provides first-class elements to represent services, participants or contracts. It can be noted that IoTDraw is more expressive than the other approaches regarding concepts related to SOA style.

B. Concrete Syntax Analysis

The Cognitive Dimensions [30] is the framework adopted to evaluate the concrete syntax of SoaML4IoT. Our analysis is based on the study of Souza and colleagues [31], which applies the Cognitive Dimensions framework to evaluate the graphical notation of a metamodel for software development methodologies standardized by ISO/IEC 24744 [32]. The authors adopt six dimensions in the evaluation, that is, *viscosity*, *hidden dependencies*, *visibility*, *premature commitment*, *abstraction* and, *secondary notation*. For each dimension, it is defined the level (low, high) that can be attained by the elements and diagrams. Table 3 lists each dimension and its levels. In the following, we analyze each dimension considering the SoaML4IoT.

Table 3. Cognitive dimensions and levels

Dimension	Description	Levels
Viscosity	Resistance to make changes.	**Low**: easy to make changes (few strokes to draw the elements and few components for each element) **High**: difficult to make changes (high interdependence between composing elements and diagrams)
Hidden dependencies	Degree to which important links between entities are implicit.	**Low**: links between elements are explicit (there are pre-defined types of relationships between pair of elements). **High**: links between elements are implicit (the types of relationships between elements are unknown)
Visibility	Ability to view components easily.	**Low**: elements are hard to be distinguished (different kinds of element with similar color and shape, hardly distinguished when placed side by side). **High**: elements are clearly distinguished (each kind of element has a different color and shape, easily distinguished when placed side by side).
Premature commitment	The order of notation manipulation actions is constrained.	**Low**: the other of drawing the elements is free (the sequence suggested for creating the diagrams admits overlaps and iteration). **High**: the order of drawing the elements is constrained (the notation enforces a pre-defined order in which to create the diagrams).
Abstraction	Possibility to create new elements in the notation.	**Low**: difficult to create new elements (the notation does not envision changes on the elements made by users, but it does provide abstract

Dimension	Description	Levels
		symbols when specific types are unknown). **High**: possible to create new elements (the notation allows the creation of new elements, symbols, relationship types, etc.).
Secondary notation	Ability to add information on the notation outside its formal concrete syntax.	**Low**: lack of informal syntax (the notation is designed to depict information using only formalized elements). **High**: presence of informal syntax (the notation allows using informal notation to add extra information).

1) Viscosity

The viscosity dimension of a notation refers to its resistance to changes. One relevant aspect of such dimension is that the viscosity is related to the exploratory design, in which the modelers would need a certain degree of freedom when drawing the elements of a given system with the notation. High viscosity is harmful to exploratory design because it makes difficult to introduce any changes in the models specified with the notation. In this sense, low viscosity is beneficial for graphical notations. On the other hand, high viscosity enhances the precision of the model, since it does not allow any representations that are not aligned with the syntactic and semantic foundations. It may be beneficial for creating unambiguous models. Otherwise, whether a model has representations that are not explicitly aligned with the foundations, the models may harmful the communication between stakeholders.

The IoTDraw was conceived as a profile for UML; thus, it inherits the notations of such a modeling language. In this sense, introducing changes in the notation of UML is quite tricky because all elements express concepts from underlying syntactic and semantic. That is, when changing a notation of a given UML element, it is necessary to align it explicitly with the concept that this new notation refers to. In this sense, the viscosity of SoaML4IoT is high, which improve the precision of the models but may hamper the exploratory design.

2) Hidden dependencies

There are two types of dependencies, that is, *topological containment* and *relationships*. The first refers to elements that contain other elements. The second is related to relationships between a pair of elements. Hidden dependencies occur when the relationship between two components is not fully visible in the diagrams. High level of hidden dependencies in a given notation is acceptable when modelers are drawing diagrams in exploratory design since it produces a simpler model. However, when specifying the architecture of a given system, it is expected low hidden dependencies aiming to guarantee the precision of the model.

In SoaML4IoT, the level of hidden dependencies is low. Considering the topological containment, an IoT system is modeled as a UML Class containing explicitly objects typed as applications and devices. The hidden dependencies dimension of SoaML4IoT is especially low regarding the relationships of the elements.

3) Visibility

The visibility dimension is the capability of a modeling language to provide elements that are easy to identify in a diagram. When the visibility is high, it facilitates the identification of distinct components of a system, being crucial in complex systems. On the other hand, low visibility can be harmful to the fast comprehension of the model, especially for modelers that do not participate in the specification.

SoaML4IoT is based on UML profiling, which distinguishes the elements only with the stereotype annotation. It is not easy to distinguish the elements of SoaML4IoT since the only difference between them is the profile signature in the top of the UML elements. Therefore, the visibility of SoaML4IoT is low, that is, elements are hard to be distinguished. However, UML profiles allow associating graphical icons to the stereotypes, which facilitates the facilitates the identification of distinct components of a system.

4) Premature commitment

In the context of modeling languages, premature commitment is related to how free the modeler is when creating the model regarding the order to represent its elements and diagrams. When the level is low, i.e., that is little constraints to manipulate, the notation it may be beneficial for the modeler since the notation does not force he/she to decide prematurely issues related to a particular aspect of the system. On the other hand, the language may support the modeling process by enforcing a pre-defined order in which to create the diagrams. In this case, the level premature commitment of the language is high.

SoaML4IoT follow the contract-based SOA approach, that is, the first step to create a services architecture is the specification of contracts and their participants. Next, it is necessary to model the participants realizing the interfaces, and, in the following, compose the services architecture. Therefore, the level of the premature commitment of SoaML4IoT is high.

5) Abstraction

The abstraction refers to the capability of the notation providing means for creating new elements for the concepts of the metamodel. It is worth to highlight that it does not refer to extending the concepts of the reference model but creating new symbols for the existing ones. Creating new symbols provide an extension mechanism for the modelers. However, low level of abstraction in the notation may ensure the precision of the models since there is only one symbol for each concept. SoaML4IoT does not envision changes on the elements made by users.

6) Secondary notation

The secondary notation refers to extra information added to the model but that does not impact on the model consistence. That is, high level of secondary notation dimension allows to enhance the model with additional information. The SoaML4IoT allows adding extra information on the models only through UML notes.

V. Conclusion and Future Work

In this paper, we propose a modeling language for the specification of SOA-based IoT applications named as SoaML4IoT. The language extends the SoaML OMG

standard, which is the standard graphical modeling language proposed by the OMG for designing systems that follow the services approach. The evaluation shows that our language is more expressive then the related work, which allow to identify elements of both IoT and SOA domain. Due the lack of space, it is not possible to show an example on how to model SOA-based IoT applications with SoaML4IoT. However, such an example can be found at brccosta.github.io/iotdraw, where the profile and its manual are also available for download.

The SoaML4IoT extends the SoaML profile by enhancing its metamodel with IoT-specific concepts. Therefore, we follow the SoaML design, which does not specify any constraint to support model validation. In our approach, the only constraints that support the modelers to conceive models that are compliant with SoaML4IoT metamodel are the relationships and their cardinalities. Thus, and important future work is to analyze which constraints should be created furthering the correctness of the models. Such constraints must reflect rules of the IoT applications domain. Since SoaML4IoT is fully OMG-compliant, the constraints can be written in OCL [33], for example, as invariants and pre- and post-conditions.

ACKNOWLEDGMENT

This work is partially supported by São Paulo Research Foundation – FAPESP, through grant number 2015/24144-7. Flávia C. Delicato and Paulo F. Pires are CNPq Fellows.

REFERENCES

[1] R. Buyya and A. V. Dastjerdi, Eds., *Internet of Things: Principles and Paradigms*. Elsevier, 2016.

[2] A. Ghasempour, "Optimum number of aggregators based on power consumption, cost, and network lifetime in advanced metering infrastructure architecture for Smart Grid Internet of Things," in *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2016, pp. 295–296.

[3] A. Ghasempour and T. K. Moon, "Optimizing the Number of Collectors in Machine-to-Machine Advanced Metering Infrastructure Architecture for Internet of Things-Based Smart Grid," in *2016 IEEE Green Technologies Conference (GreenTech)*, 2016, pp. 51–55.

[4] A. Pathak and V. K. Prasanna, "High-Level Application Development for Sensor Networks: Data-Driven Approach," in *Theoretical Aspects of Distributed Computing in …*, 2011, pp. 865–891.

[5] C. M. de Farias *et al.*, "COMFIT: A development environment for the Internet of Things," *Futur. Gener. Comput. Syst.*, 2016.

[6] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2012.

[7] A. Bassi *et al.*, Eds., *Enabling things to talk: Designing IoT solutions with the IoT Architectural Reference Model*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[8] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.

[9] K. Thramboulidis and F. Christoulakis, "UML4IoT—A UML-based approach to exploit IoT in cyber-physical manufacturing systems," *Comput. Ind.*, vol. 82, pp. 259–272, Oct. 2016.

[10] T. C. Rodrigues, T. Batista, F. C. Delicato, and P. de F. Pires, "Architecture-Driven Development Approach for WSAN Applications," in *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing*, 2015, pp. 68–75.

[11] P. Patel and D. Cassou, "Enabling High-level Application Development for the Internet of Things," *J. Syst. Softw.*, vol. 103, pp. 62–84, Jan. 2015.

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

[13] F. Pramudianto *et al.*, "IoTLink: An Internet of Things Prototyping Toolkit," in *Proceedings - 2014 IEEE International Conference on Ubiquitous Intelligence and Computing, 2014 IEEE International Conference on Autonomic and Trusted Computing, 2014 IEEE International Conference on Scalable Computing and Communications and Associated Sy*, 2014, pp. 1–9.

[14] B. Morin, N. Harrand, and F. Fleurey, "Model-Based Software Engineering to Tame the IoT Jungle," *IEEE Softw.*, vol. 34, no. 1, pp. 30–36, Jan. 2017.

[15] A. Sivieri, L. Mottola, and G. Cugola, "Building Internet of Things software with ELIoT," *Comput. Commun.*, vol. 89–90, pp. 141–153, Sep. 2016.

[16] C. Kamienski *et al.*, "Application development for the Internet of Things: A context-aware mixed criticality systems development platform," *Comput. Commun.*, vol. 104, pp. 1–16, May 2017.

[17] OMG, "Service oriented architecture Modeling Language (SoaML) Specification. Version 1.0.1," 2012.

[18] Microsoft, "Microsoft Azure IoT Reference Architecture." [Online]. Available: https://azure.microsoft.com/en-au/updates/microsoft-azure-iot-reference-architecture-available/. [Accessed: 01-Feb-2018].

[19] IEEE, "IEEE P2413 Working Group," 2015. [Online]. Available: http://grouper.ieee.org/groups/2413/. [Accessed: 03-Aug-2015].

[20] OMG, "Unified Modeling Language - UML, Version 2.5.1," 2017.

[21] L. Fuentes-Fernández and A. Vallecillio-Moreno, "An Introduction to UML Profiles," *Upgrade*, vol. 5, no. 2, 2014.

[22] B. Elvesæter, A.-J. Berre, and A. Sadovykh, "Specifying Services using the Service Oriented Architecture Modeling Language (SoaML) - A Baseline for Specification of Cloud-based Services," in *Proceedings of the 1st International Conference on Cloud Computing and Services Science*, 2011.

[23] J. Horkoff, F. B. Aydemir, F.-L. Li, T. Li, and J. Mylopoulos, "Evaluating Modeling Languages: An Example from the Requirements Domain," *Concept. Model.*, pp. 260–274, Oct. 2014.

[24] S. Patig, "Measuring Expressiveness in Conceptual Modeling," Springer, Berlin, Heidelberg, 2004, pp. 127–141.

[25] OGC, "Sensor Model Language (SensorML)," 2014. [Online]. Available: http://www.opengeospatial.org/standards/sensorml. [Accessed: 28-Mar-2017].

[26] J.-F. Monin and M. G. Hinchey, Eds., *Understanding Formal Methods*. London: Springer London, 2003.

[27] E. Y. Nakagawa, F. Oquendo, and J. C. Maldonado, "Reference Architectures," in *Software Architecture 1*, Chichester, UK: John Wiley & Sons, Ltd, 2014, pp. 55–82.

[28] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz, "Reference Model for Service oriented Architecture," 2006.

[29] T. O. Group, "Service-Oriented Architecture Ontology Version 2.0." [Online]. Available: http://www.opengroup.org/soa/source-book/ontologyv2/index.htm. [Accessed: 11-Jul-2018].

[30] A. Blackwell and T. Green, "Notational Systems—The Cognitive Dimensions of Notations Framework," in *HCI Models, Theories, and Frameworks*, Elsevier, 2003, pp. 103–133.

[31] K. Sousa, J. Vanderdonckt, B. Henderson-Sellers, and C. Gonzalez-Perez, "Evaluating a graphical notation for modelling software development methodologies," *J. Vis. Lang. Comput.*, vol. 23, no. 4, pp. 195–212, Aug. 2012.

[32] ISO/IEC, "ISO/IEC: 24744 Software Engineering— Metamodel for Development Methodologies. Annex A Notation," Geneva, 2010.

[33] OMG, "Object Constraint Language (OCL)," 2016. [Online]. Available: http://www.omg.org/spec/OCL/. [Accessed: 04-Feb-2017].