Model Driven Development for Internet of Things Application Prototyping

Ferry Pramudianto
Fraunhofer FIT
Schloss Birlinghoven
Sankt Augustin, Germany

Indra Rusmita
Bonn-Aachen International Center
for Information Technology
Dahlmannstraße, Bonn-Germany

Mathias Jarke
I5 RWTH Aachen University
Templergraben 55
Aachen, Germany

***Abstract*—We present an architectural view for the Internet of Things prototype development that emphasizes the separation of domain modeling from technological implementations. Using the provided model driven tool, domain experts are able to construct domain models easily by composing virtual objects and linking them to the implementation technologies. Having them linked, a prototype code in Java can be generated by the tool. The generated code allows developers to extend it into full applications simply by interfacing the virtual objects without dealing with the complexity of specific sensors and actuators technologies. Subsequently, participants involved in the European research projects evaluated the architecture and the tool using a software walk-through technique whose results are discussed in this paper.**

Keywords-component; internet of things, architecture, domain model, code generation, model driven development, service oriented architecture.

I. INTRODUCTION

The Internet of Things (IoT) refers to an emerging paradigm which envisions seamless integration among smart physical objects, applications, and services that interact and communicate among themselves by exchanging data and information[1]. The growth of IoT community has been encouraged by the rapid development of wireless sensor and actuator networks, identification tags such as barcode and RFID, and electronic prototyping platforms such as Arduino[1]. Nonetheless, IoT is still very young research field where researchers and industry are still trying to find a common ground to establish standardized approaches. This has made IoT prototype development challenging.

According to our interviews to the developers involved in several European research projects dealing with IoT applications, they often face problems during IoT developments caused by the lack of technology and architecture standardization. This is caused by the existence of different visions for IoT[2]. The network-oriented vision focuses on the communication for IoT devices. The "Thing" vision focuses on identification through ID tags. The semantic oriented vision focuses on processing the massive information generated by the IoT. Despite several IoT architectures exist there is still an open question on how the architecture reference could be designed in a way that the domain modeling could be decoupled from the implementation of specific IoT technology. Decoupling these allows the knowledge about the domain to be engineered by domain experts while the technology experts focus on addressing the implementation of the IoT technology.

Addressing this research question, this paper proposes a unique perspective on IoT architecture that separates the design of the domain model and the implementation of the IoT technology. Supporting the proposed architecture, this work also proposes a model driven development (MDD) tool for linking the domain model with the IoT implementations. Based on the model definition, the tool will generate Java artifacts consisting of the domain model as a virtualization of smart objects linked to the concrete implementation of IoT technology. This allows application developers to develop IoT applications using the virtual objects without having to deal with the complexities of any IoT technology.

II. RELATED WORK

The first use of IoT term was coined by The Auto-ID Labs in their work to solve product traceability problems for the supply chain management[3]. Together with the EPCGlobal they have proposed a standard architecture for universally identifying goods with RFID tags and a service registry network for querying information of the tagged goods through third party service providers[4] (Fig.1). However a survey claims that RFID is only a part of broader IoT vision where smart objects autonomously cooperate with each other[2].

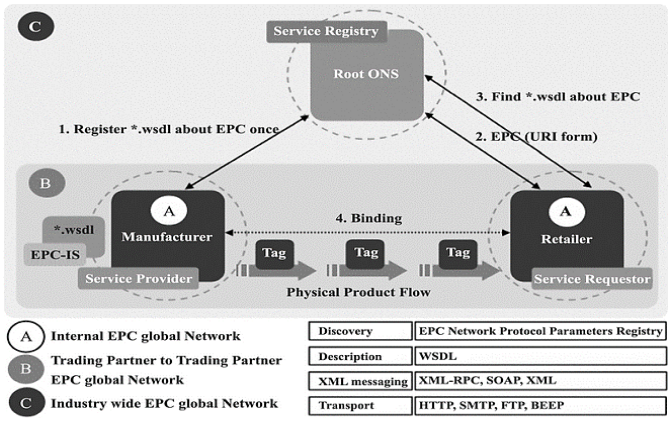

Figure 1. EPCGlobal Architecture[5]

Another survey presented a five layer architecture that placed the internet as a middle layer which functions as the main communication media (Fig.2) [6]. The edge layer manages

[1] http://www.arduino.cc/

devices such as embedded systems, sensors, actuators, and ID tags. The access gateway layer cares about bridging different communication technologies to the internet. The main task of this layer is performing a routing optimization, bridging the different communication protocols to the internet protocols (e.g. TCP/IP), and forwarding data from the edge nodes to the other end across the internet.

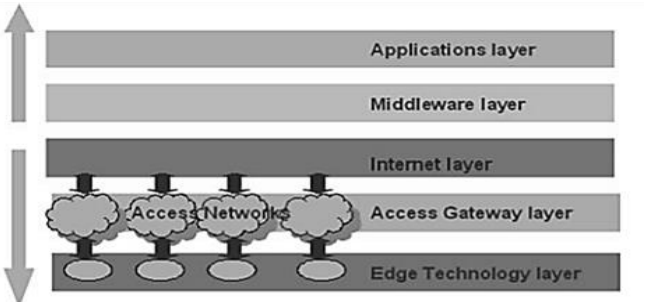

Figure 2. Generic Layered Architecture for IoT[6]

The middleware layer provides generic interfaces for the applications to communicate with the internet of things. Many approaches have been used for abstracting IoT devices e.g.: data oriented middleware uses SQL-like query languages to retrieve information from the sensor nodes, service oriented architecture (SoA) middleware has been proposed to support the integration of among "Things", legacy systems, and the necessary infrastructure while providing interoperable web services for the applications accessing them [7-9]. This layer may also perform device and information management by utilizing data fusion, semantic analysis, access control, information discovery.

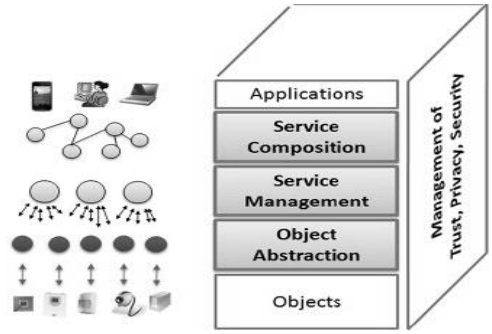

Figure 3. SOA-based architecture for the IoT middleware[2]

SoA middleware also introduces a service management layer that deals with service discovery, execution monitoring, and configuration. For the discovery purposes, a service registry is usually used. This approach provides an abstraction of various communication technology by encapsulating them with web services. SoA depends on workflow and web service composition languages such as WSBPEL[2] to provide services that are more complex.

III. ARCHITECTURE

Our work offers a unique perspective of an IoT architecture that places the domain modeling in the center of the architecture. This is done so to emphasize the seperation of the knowledge engineering that happens during domain modeling with the technical engineering that happens during the implementations. As a result, this approach enables domain knowledge to be modeled by the domain experts that are not familiar with programming languages but familiar with languages used for managing knowledge in the domain e.g. ontology. This approach will also allow the domain model and the device implementations to evolve independently over time without having to reengineer the whole systems.

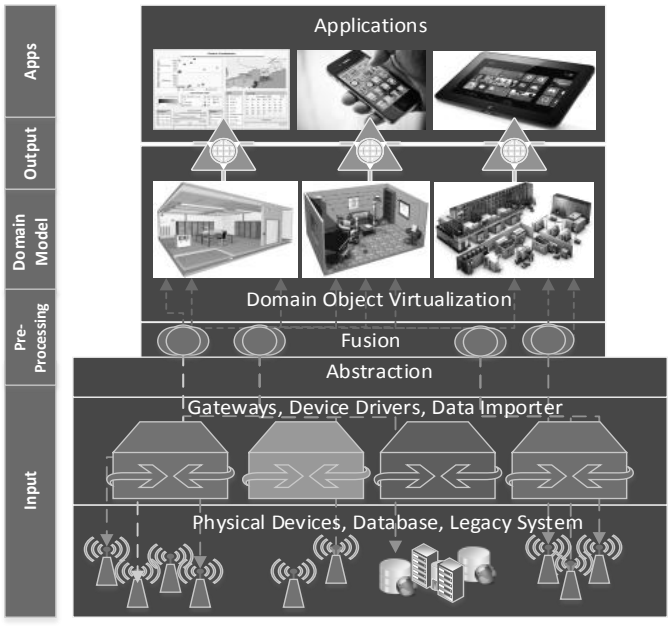

Figure 4. Architecture for object virtualization in IoT Applications

As depicted in Fig.4, the domain model layer in our architecture contains the domain knowledge such as the relationship of the objects, their capabilities and their properties as perceived by the domain experts. For instance when developing a monitoring application for a smart building, these objects may consist of the occupants, building structures (e.g. floors, rooms, windows), appliances (e.g. radio, monitor, air conditioner). Furthermore, this layer is responsible for virtualizing the physical "Things" that participate in the application domain. By virtualizing, we meant that the physical objects alone might not be able to interact with the applications without the support of devices such as sensors, actuators, and ID tags. Therefore, the representation of these objects may be composed of the supporting devices, which we refer as the enabler devices. These enabler devices should be completely transparent to the domain experts when they define the domain model.

Modeling the relations among virtual objects can be done using modeling languages such as UML, Ontology, or a simplified Domain specific language (DSL) that can be understood by the domain experts. As a proof of concept, this

[2] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

work uses a simplified graphical DSL designed for rapid prototyping (see section 4).

On the lowest layer, the enabler devices are managed and abstracted with a common interface that is understood by the upper layer. A common approach used in the lowest layer is the use of bridge or gateways for different networks that allows communication to be estabished with applications using TCP/IP protocol. For instance in industrial automation, an OPC server is often used as a bridge to access BUS networks. This architecture pattern is also used by the emerging IoT technology such Zigbee[10] and 6LowPAN[11]. In contrast, communicating with legacy systems require various technologies which are diverse from domain to domain. Some of the legacy applications offer an application programming interface (API) in propriatery languages, some of them use database and log files to retrieve data, and some of them provide web services. In the industrial setting where Enterprise Resource Planing (ERP) and Manufacturing Execution System (MES) are invoved, ISA-95[12] is the common standard to retrieve and store data from these systems.

Additionally, since the internet and web provide a huge amount of information that can be useful, the architecture should take into account that the applications might need to access the online services such as, weather forecasts, stock prices, exchange rates, or information about people from their social network sites. For this purposes normally web service technologies can be used.

Abstracting the heterogenous technologies involved in the lowest layer may use web service technologies such as SOAP[13] and REST[14] since they offer interoperable services supported by different programming languages. However, at the moment the practicability of web service technology for time critical applications and resource constrain devices is still debatable. A more resource efficient protocols such as CoAP[15] is being developed for this purpose.

On the upper layer, the data delivered from the lowest layer sometimes must be be pre-processed to extract contextual information that is useful for the applications. This information could be as simple as determining a room temperature from the thermometer readings to a more complex information such as determining the activities, taking place in a room based on several sensor readings, or providing a contextual information of the users (e.g.: if the room temperature is too cold for this particular user). Thus, the fusion layer provides several generic pre-processing modules that are usefull for e.g.: filtering outliers, averaging the data over time and space, applying high-pass - low pass filter, interpolating the data (e.g.: Kalman Filter). The fusion layer can be extended by providing more domain specific fusion modules to derive information that is not possible to be sensed by a type of sensor.

When the domain model has been defined and the technology implementations in the first and second layer have been done, these layers need to be mapped in order to produce a functional application prototypes. The mapping follows a simple input output interaction between components in different layers and the virtual objects. For instance, the property of the virtual object "room" is linked to a processing module "average" that is linked to two thermometers. Based on this linked components, the property of the room will automatically be updated with the values coming out from the processing module, which contains the averaged temperature data coming from the two linked sensors. As the properties of the virtual objects always contain the actual values, the developers simply need to work with the virtual objects without worrying the technical details to access the thermometers.

IV. Domain Modeling Tool Development

As a proof of concept of the proposed architecture, we developed a domain-modeling tool that supports domain experts designing domain models containing virtual objects. We build the tool based on the requirements of twelve developers involved in several European research projects dealing with IoT. The requirement elicitation was done through a focus group workshop where the developers are given scenario to build IoT prototypes. Then, we discussed about a visionary tool that could help them solving their tasks rapidly. The outcome from the focus group reveals that the developers are in favor of a graphical model driven tool that would help the domain experts designing the domain model and then allows them to map the virtual objects to the sensor and actuators. They would like to have the domain model defined using simple notations that can be quickly explained to non-computer scientist users. Finally, the tool should generate a Java code that can be extended to develop their final applications. After the requirements are collected, we designed the user interface mockup and iteratively evaluated it with the users who.

A. User Interface Mockup

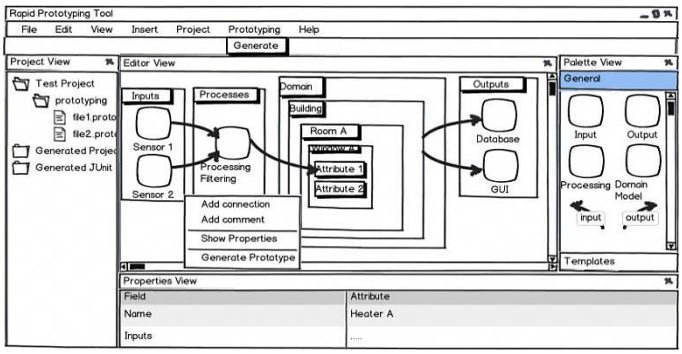

Figure 5. The mock up user interface for the development tool

The mockup GUI composed of several views including Project View, Editor View, Palette View, and Properties View (figure 5). On the toolbar, there is a button to generate the necessary Java artifacts from the defined model including the Java code of the virtual objects, the mappings to the physical objects, and the library to access the physical objects. The Project View may contain several projects. Each project may consist of several domain model diagrams, which each of them contains the definition of the virtual objects, their properties, and links to the processing modules, data providers, and actuators. We created a set of simple notations to simplify defining a domain model and the mappings to sensors and actuators. However, our notations are not as expressive as UML and Ontology as it is not intended to support development for complex applications. These notations are presented in the Palette view. The notations consist of rectangles that can be containers for other rectangles and linked using arrows to map

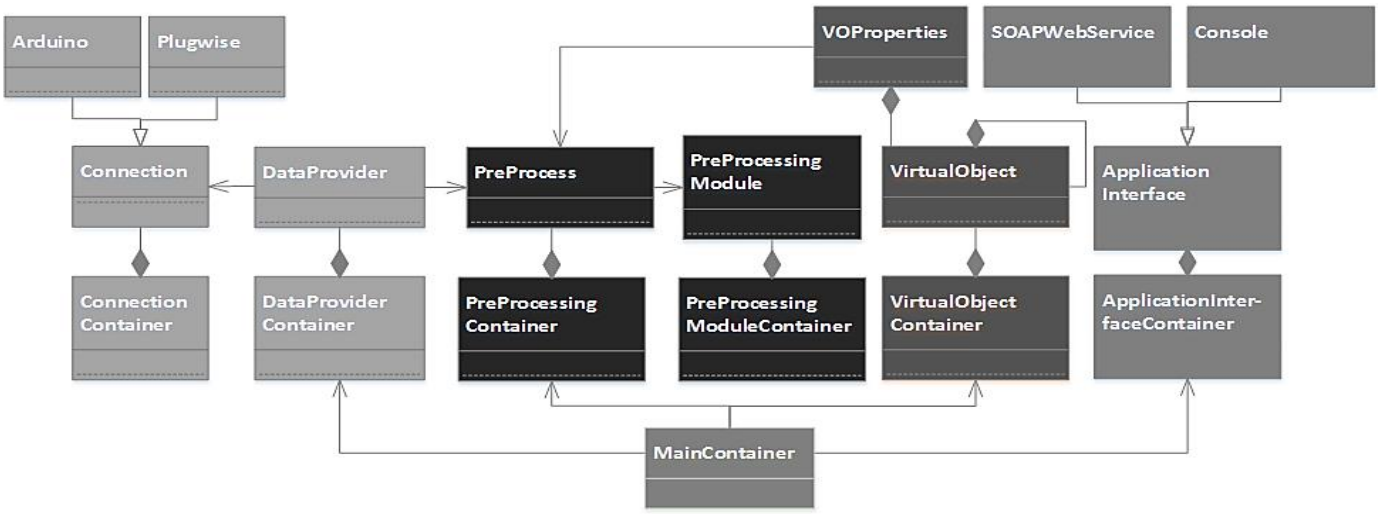

Figure 7. ECore Model of the Proposed Tool

the relationship among the objects. These atomic notations are grouped using a tabular menu depending upon the type of the notations. The Editor View is the main container where the developers could define the domain model using the provided notations. In the Property View users can modify the properties of the notations depending the type of the notation.

B. Modeling Tool

The modeling tool is built with the Eclipse Modeling Framework (EMF)[3]which is responsible for defining the meta model of the proposed architecture depicted in Fig.4. We use Graphical Modeling Framework (GMF) [4] for generating the MDD editor used for our domain specific language. Firstly, the tool was built by defining the *ECore* model, which is needed by EMF(Fig.7) as the meta model for the tool. The *ECore* model was defined to have a main container and several containers. These containers include containers for the Data Provider, Pre-Processing, Virtual Object, and Application Interface. Each of these containers could contain more than an implementation of the abstract classes depicted in the center row of Fig.7. We implemented the abstract classes as examples that are useful for evaluating the tool against the user requirements. These abstract classes are extendable when further IoT technologies to be added in the future.

After creating the meta-model, we use the EMF Generator Model to generate the plugin projects that we need to implement such as the "Model Code", the "Edit Code", the "Editor Code", and the Java Interfaces. After the skeleton is generated, we used GMF to create a diagram editor using GMF Tooling. We edited the *gmfgraph* to define the graphical notations and *gmftool* to define the tooling of the editor such as menu, and palate. Moreover, in the *gmfmap*, we mapped the notations to the domain model of the tool defined by the EMF. We use EMF also to provide serialization of the model defined by the users. Currently it only supports XMI[5] format, which can be stored and opened back to the editor view when the users want to continue working on them. The serialization can be done in other formats, however for the sake of simplicity we took the standard format provided by EMF.

The base classes are implemented as Eclipse plugins. This provides flexibility when further components need to be integrated into the tool. For instance the Connection base is implemented as an eclipse plugin which is extended by two other eclipse plugins containing implementations to create connections to the corresponding devices. These base classes provide an abstract factory to be used by the wizards in the eclipse IDE for retrieving the actual implementations of the plugins.

C. The code generator

The code generator is implemented using Xpand for generating Java code and the necessary artifacts based on a set of template codes. The template codes contain all implementations of the abstract classes defined in the ECore model that take into account adjustments that the users will define when modeling the prototype applications. These adjustments include adjusting the package and class names, assigning the values of the sensors to processing module, assigning the output of the processing modules to the properties of the virtual objects. These adjustments will be generated by the Xpand plugins that we have developed. After the template code is adjusted and generated, XPand additionally generate an eclipse Java Project and all the necessary artifacts such as libraries and a run configuration that the users need to run the generated project properly.

In the current implementation, the generated java project will consist of the chosen connections. There are two connections supported Plugwise[6] and Arduino which are connected through the serial ports.

D. The workflow of the tool.

When developing a new application prototype with the tool, several steps as depicted in Fig. 8 must be followed. The users start with creating a new project and entering its name. Next, the

[3] http://www.eclipse.org/modeling/emf/

[4] http://www.eclipse.org/modeling/gmp/

[5] http://www.omg.org/spec/XMI/

[6] http://www.plugwise.com

users create a new domain model diagram and enter its name. Then, the users can start designing the domain model on the editor view as depicted in fig. 12(B). Designing the domain is started by adding the main container for the virtual objects, and the virtual objects themselves in the container. The virtual objects can contain other virtual objects that denote "a part-of" relationship.

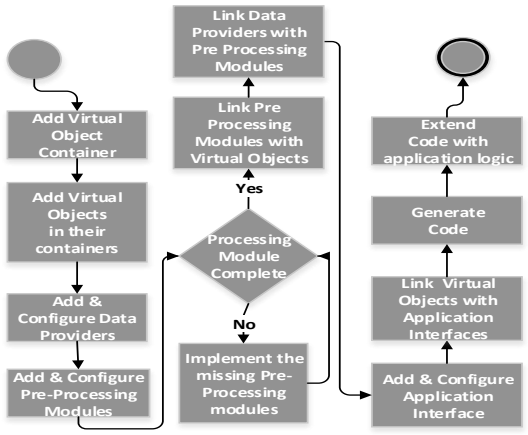

Figure 8. A simplified workflow of the development with the tool

After the domain model is defined, the users could add predefined processing modules. In case the needed modules is not available, software developers may implement the modules by extending the corresponding plugins. When the needed processing modules have been added to the Editor View, the users link the properties of the virtual objects to the modules and then the modules also have to be linked to the data providers. Finally, the users add the application interface such as SOAP, or REST, and link the application interface to the virtual objects so that the tool knows which virtual objects it needs to expose to the applications and with which technology it should be done. After all necessary associations have been done, the user can generate the Java project that can be run and accessed from their application through the chosen application interfaces.

V. EVALUATION

A software walkthrough [16] was performed to evaluate the users acceptance to the proposed architecture and the proposed MDD tool. The evaluation was done with 7 participants (6 male and a female). Six of them are system developers who are working in Fraunhofer FIT and participate in European research projects dealing with IoT implementations in different domains. A participant is a student of the technical university of Aachen (RWTH Aachen) who also works in an IoT project at Fraunhofer FIT. Their experience in application developments range between 2 until 6 years. The participants are between 25 and 35 years old.

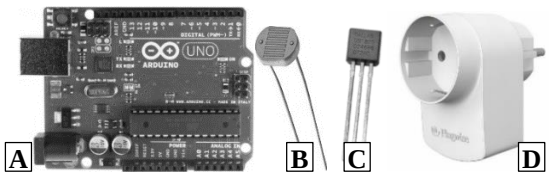

Figure 9. Arduino Board(A), Light Sensor(B), Digital Thermometer (C), Plugwise (D).

The equipment used to perform the evaluation consists of an Arduino board (Fig. 9. A), a light intensity sensor (Fig. 9. B), a digital thermometer (Fig. 9. C), and a Plugwise (Fig. 9. D). The Arduino board was used to retrieve data from the light and temperature sensors and send them through a serial port. To communicate with Plugwise, a Zigbee USB receiver was used.

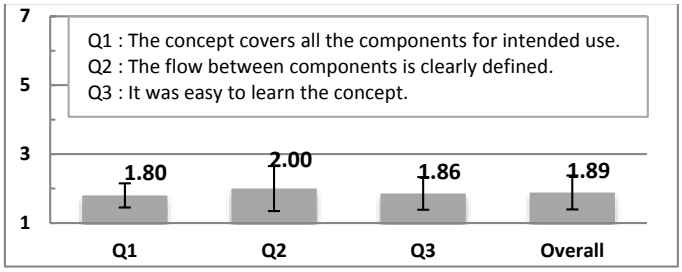

Figure 10. User satisfaction of the proposed architecture

The participants were given a task to display sensor values from a temperature, a light sensor attached to an Arduino board, and the power meter. After they had performed the task, they were given a questionnaire to review the proposed architecture, and the tool. To ensure the result of the study could be compared to similar works in the future, some questions of the questionnaire were taken from the IBM Computer System Usability Questionnaire[17]. The questions were presented with 7-level Likert-scale options where "7" denotes "Strongly Disagree" down to "1" which means "Strongly Agree".

As depicted in Fig. 10, the result of the questionnaire regarding the architecture shows that the users felt it was easy to understand *(M=1.86, SD=0.64)* and its functions were clearly defined *(M=2, SD=0.82)*. Secondly, the proposed architecture helped the user developing the intended functional prototype *(M=1.8, SD=0.4)*. Overall the users were satisfied to the architecture design *(M=1.89, SD=0.67)*.

Furthermore, to investigate the user satisfactions to the overall work, of the participants were given a task to solve with the proposed tool and Eclipse's EMF tool. The order of the tool used by each participant was exchanged to minimize the learning effect. Then the users were asked to rate the overall experience working with the tool using DSL notations compared to the EMF using UML notations.

The questions were divided into four categories that include the overall experience with the framework (Overall), the functionalities of the tool (Tool Functions), the workflow to be done when working with the tool (Workflow), and the user interface of the tool (UI). The questions are again adopted from the [17].

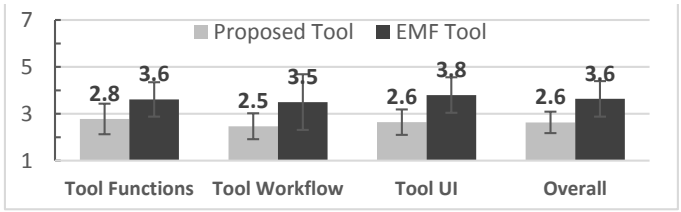

Figure 11. Comparisson between EMF tool & IoT Modeling Tool

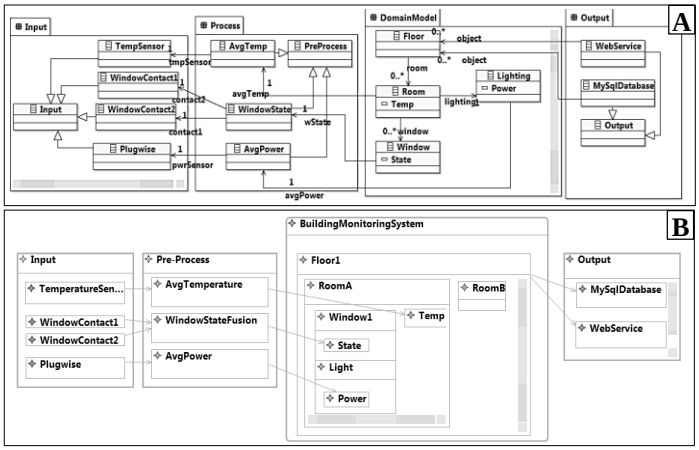

Figure 12. EMF with UML notation (A) vs. IoT Modeling Tool with DSL notation (B)

The score comparison between our work and EMF Tool are presented in Fig. 11. Overall, the writer's work scored better compare to the EMF tool with an UML diagram.

A paired sample T-Test analysis was performed to investigate if the difference between user's satisfaction to the writer's work and EMF tool is statistically significant. The result of the questionnaire shows that even though the proposubiquotous ed tool scored a better means in all categories there was no significant differences of user satisfaction for the Functions, Workflow, and Overall [T(7)=1.2, p>.5), (T(7)=-1.38, p>.5), (T(7)=-2.04, p>.5) respectively]. Interestingly, the user opinions were significantly affected by the user interface of the tools (T(7)=-2.66, p<.5).

The fact that the user opinions are affected by the user interface indicates that a simplified domain specific language serves a better purpose for simple prototyping tasks than complex modeling languages such as UML since the users are faced with simplicity and less options that may overwhelm them in solving the tasks.

VI. Conclussion & Future Work

The current approaches of IoT architectures have overlooked the importance of domain modeling in the application development. This work has presented a unique perspective that positions domain modeling and object virtualization in the center of the architecture. Moreover, this work has proposed a tool that augments the proposed architecture by allowing the domain model to be linked to the IoT implementations. Consequently, the complexity of IoT implementations is transparent for the application developers, as they only need to work with the virtual objects generated by the tool. The results of the preliminary evaluation support our claim that the proposed architecture and the model driven tool have a potential to ease IoT application development.

The next steps for this work are to provide an easy way for the domain experts to express their domain knowledge related to policies and rules that are difficult to express through a graphical notations. For this purpose, the tool could be extended by integrating a rule engine in the generated code. This approach allows policies to be dynamically modified without recompiling the application. Once the tool has enough features to support complex application developments, we also would like to perform evaluation with more users in a longer period of time.

VII. Acknowledgement

This work was co-funded by the European Commission through EBBITS (FP7-ICT-2009.1.3, GA No. 257852) and BEMOCOFRA (FP7-ICT-2011-EU-Brazil, GA No. 288133)

VIII. References

[1] Guillemin, P., and Friess, P.: 'Internet of Things: Strategic Research Roadmap', CERP-IoT Project, 2009

[2] Atzori, L., Iera, A., and Morabito, G.: 'The internet of things: A survey', Computer Networks, 2010, 54, (15), pp. 2787-2805

[3] Bose, I., and Pal, R.: 'Auto-ID: managing anything, anywhere, anytime in the supply chain', Communications of the ACM, 2005, 48, (8), pp. 100-106

[4] Johnson, F.A.J., Harrison, M., US, B.H.G., Mitsugi, J., Preishuber, J., CVS, O.R., and Suen, K.: 'The EPCglobal Architecture Framework', 2005

[5] Shih, D.-H., Sun, P.-L., and Lin, B.: 'Securing industry-wide EPCglobal network with WS-security', Industrial Management & Data Systems, 2005, 105, (7), pp. 972-996

[6] Bandyopadhyay, D., and Sen, J.: 'Internet of Things: Applications and Challenges in Technology and Standardization', Wireless Personal Communications, 2011, 58, (1), pp. 49-69

[7] de Souza, L., Spiess, P., Guinard, D., Köhler, M., Karnouskos, S., and Savio, D.: 'Socrades: A web service based shop floor integration infrastructure', The Internet of Things, 2008, pp. 50-67

[8] Jammes, F., and Smit, H.: 'Service-oriented paradigms in industrial automation', Industrial Informatics, IEEE Transactions on, 2005, 1, (1), pp. 62-70

[9] Eisenhauer, M., Rosengren, P., and Antolin, P.: 'A development platform for integrating wireless devices and sensors into ambient intelligence systems', in Editor (Ed.)^(Eds.): 'Book A development platform for integrating wireless devices and sensors into ambient intelligence systems' (IEEE, 2009, edn.), pp. 1-3

[10] Alliance, Z.: 'Zigbee specification', ZigBee document 053474r06, version, 2006, 1, pp. 378

[11] Mulligan, G.: 'The 6LoWPAN architecture', 'Book The 6LoWPAN architecture' (ACM, 2007, edn.), pp. 78-82

[12] Scholten, B.: 'The road to integration: A guide to applying the ISA-95 standard in manufacturing' (Isa, 2007. 2007)

[13] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., and Nielsen, H.F.: 'Simple object access protocol (SOAP) 1.2', World Wide Web Consortium, 2003

[14] Fielding, R.T.: 'Chapter 5: Representational State Transfer (REST)', Architectural Styles and the Design of Network-based Software Architectures, Dissertation, 2000

[15] Shelby, Z., and Team, C.A.: 'Constrained Application Protocol (CoAP) draft-ietf-core-coap-04', IETF work in progress, 2011

[16] , R.: 'The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company', 'Book The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company' (ACM, 2000, edn.), pp. 353-359

[17] Lewis, J.R.: 'IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use', International Journal of Human‐Computer Interaction, 1995, 7, (1), pp. 5