From IoT Mashups to Model-based IoT

Christian Prehofer, Luca Chiarabini
fortiss GmbH, Munich, Germany
{Prehofer, Chiarabini}@fortiss.org

Abstract— In this paper, we consider tools and methodologies for the development of application for the Internet of things. We compare tools for application mashup with approaches for model-based development. For this, we show an example and then identify several key differences. Based on these, we discuss how both approaches can benefit from each other. For instance, model-based approaches have more expressiveness to model different views and behavior and then to generate code from models for different platforms.

I. INTRODUCTION

The basic idea of the Internet of Things vision is the pervasive presence around us of a variety of things or objects – such as Radio-Frequency IDentification (RFID) tags, sensors, actuators, mobile phones, etc. – which are able to interact with each other and cooperate with their neighbors to reach common goals [1][2]. There is enormous research on new technologies and novel applications for the Internet of things, as well as in related areas like sensor networks and pervasive and ubiquitous systems. Lots of work for enablers and middleware for the Internet of things (IoT) have been developed, e.g. [3].

In this paper we consider tools and methodologies for the development of application for the Internet of things. We have seen that open, connected platforms with sensors are a very attractive target for applications. One challenge is to provide SW development tools for these platforms. As devices and SW platforms in the IoT are very heterogeneous, it is difficult to write code for multiple platforms. Also, different tool chains may be needed. This significantly hinders the development of widely available applications.

For application development, a popular approach in the IoT is the concept of mashups, for which several tools exist, e.g. Clickscript [8], WotKit [5] as well as Paraimpu [6]. These permit visual, interactive modeling of the message flow between devices, as e.g. shown in Figure 1. On the other hand, also model-based approaches have been proposed for the Internet of things, e.g. ThingML [9] among other proposals [12][13] [14]. However, there is very little discussion on how these two approaches relate to each other. While mashup tools permit very quick prototyping, modeling approaches permit very expressive modeling of systems, possibly with code generation. The goal of this paper is to compare both approaches in more detail and to propose fruitful combinations.

In the following, we first review mashup tools in Section II, and then discuss model-based approaches in Section III. In Section IV, we compare both approaches and discuss opportunities in combining both.

II. IOT MASHUP TOOLS

Mashup tools have been proposed as a simple way to develop applications by composing, or mashing up, existing services in the Web. This was supported by increasingly uniform communication protocols and APIs based on REST principles. Early mashup tools are Microsoft Popfly and Yahoo Pipes; for an overview we refer to [4].

In the recent years, there has been a lot of interest in applying the same ideas to the Internet of Things, also building on REST interfaces for the Internet of things [3][8]. Well known examples for such tools are Clickscript [8], WotKit [5] as well as Paraimpu [6]. For all three of these, commercial companies have been created to bring these innovations to the market.

According to [7], mashup tools include the following:

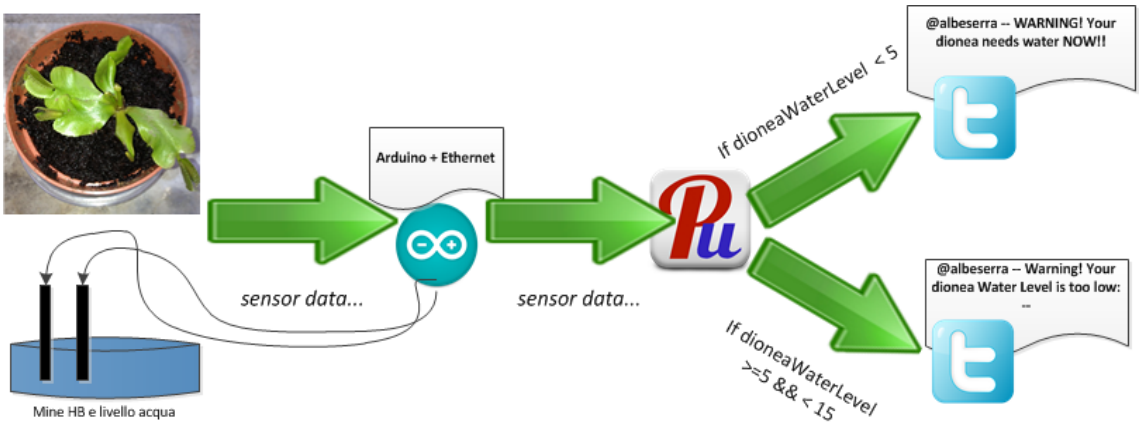

Figure 1: Paraimpu Example (From Paraiumpu.com)

1. Data mediation involves converting, transforming, and combining the data elements from one or multiple services to meet the needs of the operations of another.

2. Connecting different services to create a new process. This includes invoking the various service methods, waiting for asynchronous messages, and sending any necessary confirmation messages.

For the connecting services, there are different concepts as discussed in [10]. The main, predominant one is modeling data flow. For others, mainly in the enterprise area, also centralized approaches are considered. As an example, we show an illustration from Paraimpu in Figure 1. This shows a typical flow from sensor data, to some processing up to the connection with Web-based services.

For the communication, asynchronous messages are used, typically using REST-style communication. The orchestration between services can be directed by describing the data flow and/or workflow, or via a callback or through publish-subscribe model (see [10]).

Mashup tools typically provide a graphical editor for the composition of services for one application. This typically models the message flow between the components. Components can be sensor nodes, processing or aggregation entities as well as external Web-based services. Thus, mashup tools can also be seen as specific forms of end-user programming [11], but are however limited to the specific model of describing message flow. In addition, some mashup tools provide simulation tools and also interoperability for messaging between different platforms.

III. Model-Based Approaches for IoT Applications

In the following, we introduce our model-based approaches for the Internet of things. There is a broad range of model-based approached, up to domain specific modeling languages. Here, we mainly assume general purpose modeling tools like UML, even though many more specific approaches exits. For instance, there are several proposals for model-based approaches for developing IoT applications, e.g. the ThingML language [9] as well as other proposals [12][13] [14].

The motivation for model-based development is to describe a system on a higher level of abstraction. Typically, this is done in UML and other languages by diagrams modeling specific aspects or views of a system. Typical in our setting are architecture models and state machines.

Architecture models may describe the logical role of classes by a class diagram, a logical component model by a component diagram, as well as deployment diagrams, which show the mapping of components to physical entities.

Behavior can be described by examples in sequence diagrams, or by state machines and activity diagrams. Activity diagrams describe the data and event flow, similar to mashup tools. State diagrams are used in many embedded domains to model the behavior of specific objects. Also, state diagrams can be analyzed and verified formally (see e.g. [14]) and code can be generated automatically. In this way, it is also possible to generate code for different platforms, even though this still requires to consider different APIs for different platforms.

A high level view is shown in Figure 2, with a schematic state transition diagram as the model of a component. Development includes two steps:

1. Model and design the application in device-independent model, here state (transition) diagrams

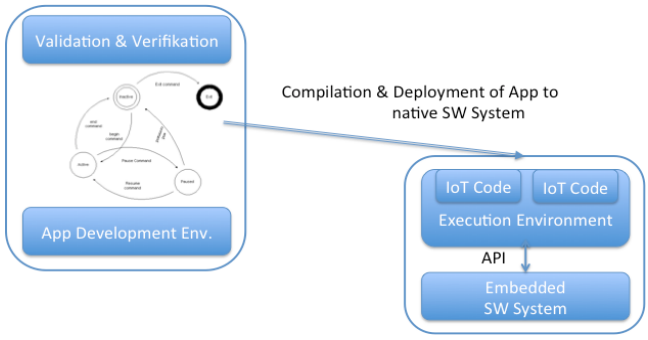

Figure 2: Model-based Development and Code

and architecture models (not shown).

2. Code generation and compilation to device-specific, native code

While state machines provide a limited programming model, the advantage is that such models can be efficiently analyzed by model checking tools (see e.g. [14]) and other tools.

An advantage of modeling tools is that there is considerable work on semantics, which means that common understanding of diagrams is formally defined. While there are challenges for semantics for UML [15] with all existing features, there exist subsets which are semantically better understood.

A. Example

To underline and to make the differences between mashup and model-based tools more explicit, we consider again the Paraimpu example of Figure 1. We can model (in UML) the logical structure of such scenario with the component diagram in Figure 3. Here, each involved component – physical or logical – is represented with a corresponding box. Boxes are

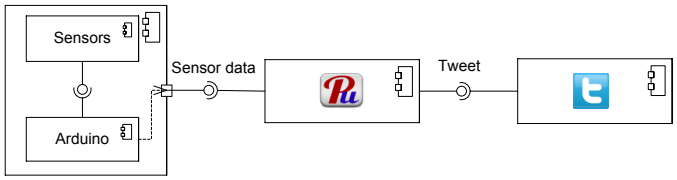

Figure 3: Component Diagram

connected through channels, where (c1 —◎— c2) stands for "the component c1 produce/offer a service, consumed/required by c2". In our case, sensors and the Arduino board – gathered in the same component – produce temperature data, that are

used by Paraimpu for internal computation. In turn, Paraimpu will produce tweets, used as principal input by Twitter.

If the component diagram models the overall logical structure of our system, the activity diagram (Figure 4) make the data flow explicit. In our scenario, the data coming from sensors

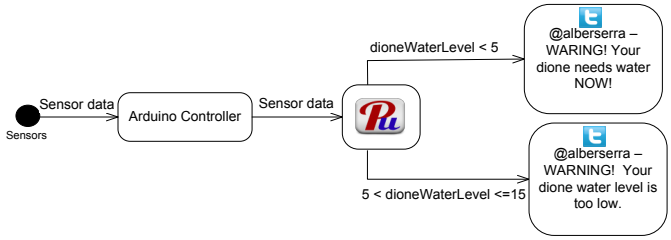

Figure 4: Activity Diagram

is collected by the Arduino controller and afterward is sent to Paraimpu. Paraimpu in turn, based on their value, will produce a corresponding tweet published on Twitter.

Based on these diagrams, we note already an important aspect: mashup models (e.g. the Paraimpu model in Figure 1), in the model-based approach, essentially correspond to activity diagrams i.e. message flow between components.

Finally, the discrete behavior of each diagram´s component, in a model based environment, is usually defined through finite state machines. For example, the behavior of the Paraimpu component (Figure 3) could be simply defined with the following automata:

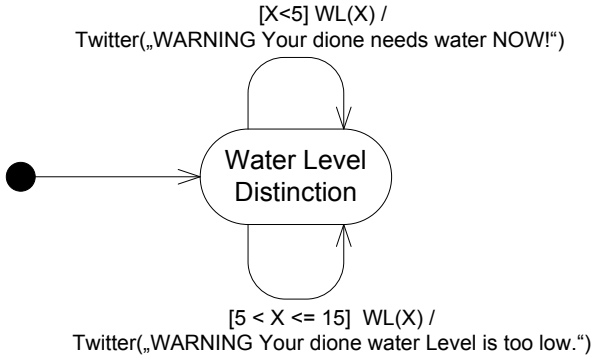

Figure 5: State Machine for Component Behavior

In the diagram in Figure 5, WL(X) is an event that, when detected –i.e. we have new input data from sensors– entail the instantiation of X with the current water´s level and enable the two transitions – labeled with *guard/action*– to be, eventually, triggered.

IV. Comparing Mashup and Model Based Approaches

In the following, we compare both approaches and discuss opportunities in combining both. When comparing model-based approaches and mashup tools, we need to realize that model-based approaches have a much wider set of modeling techniques and a much more detailed separation of different views and concerns. Thus, we now aim to discuss how mashup tools fit into the world of model based approaches.

A discussed above, model-based approaches distinguish (logical) objects, components and also between the deployments of components. Components have well defined interfaces and ports, thus matching the notion of components found in mashup tools.

In contrast, mashup tools mainly define the message flow between components. As all message flows are described in one diagram, this also describes the system architecture by showing the connected components in the work flow. Thus, mashup tools integrate a component model with a deployment model.

This is the first main difference to model-based approaches. Separating deployment and logical components creates a layer of indirection and makes prototyping less immediate, compared to mashup tools. In these, it is very easy to develop concrete systems. On the other hand, in realistic development, applications need to be mapped to different target deployments. For instance, in ThingML, so called configurations map a logical model to a specific deployment. (see http://thingml.org/pmwiki.php?n=Main.Configurations)

Secondly, mashup tools define a message flow between components. In terms of UML, this closely matches activity diagrams, where events are exchanged. Activity diagrams have typically a semantics based on events, which drive the control flow. According to [17], there are synchronous and asynchronous semantics for control flow in activity diagrams. In mashup models typically asynchronous messages are exchanged. However, most mashup systems do not describe semantics formally, and thus this is an open question. In practice, most IoT applications do not have the problem of messages queuing up, as messages are sent sparsely. Under this assumption, these semantics coincide.

A third, main difference is that components in mashup tools are either black-box entities or need to be programmed in a usual programming language, e.g. C or Java. This shows another major difference to modeling tools, which also provide modeling concepts for the behavior of components. A widely used approach for developing embedded systems is to use state machines, which are also found in ThingML. A main reason for this success is the option to generate code for different platforms from such models.

Finally, we should note that general purpose modeling languages can be complex and at the same time not ideally tailored to some domain, reason for the recent considerable research on domain specific modeling languages. Following, the above, we can see mashup tools as domain specific modeling languages, even though a detailed analysis is beyond the scope of the paper here.

V. Summary and Conclusions

In this paper, we have compared both approaches and we have now identified three key differences. This also gives rise to several opportunities. Regarding the first difference, mashup tools are nice for fast prototyping, but then need to be manually

adapted or copied for new deployments. Regarding modeling of asynchronous messages, it would be important to fix a specific semantics which is important for common understanding, testing and code generation or simulation.

On the other hand, model-based approaches have more expressiveness to model behavior and then to generate code from models. Also, different level of abstractions can be used. However, when used in basic examples, it appears that mashup tools are quite efficient in describing system architecture, message flow (like activity diagrams), and also deployment. Model-based approaches also support models for behavior. This typically done by state machines, from which code can be generated.

Overall, this paper claims that mashup tools can benefit from concepts in model-based approaches. Similarly, model-based approached can be tailored better to the Internet of things and provide easy to use tools. A main advantage of model-based techniques is the platform independent modeling, which permits code generation for specific platforms.

REFERENCES

[1] D. Giusto, A. Iera, G. Morabito, L. Atzori (Eds.), The Internet of Things, Springer, 2010. ISBN: 978-1-4419-1673-0.

[2] Luigi Atzori, Antonio Iera, Giacomo Morabito, The Internet of Things: A survey, Computer Networks, Volume 54, Issue 15, 2010

[3] C. Prehofer, J. van Gurp, V. Stirbu, S. Sathish, P. Liimatainen, C. di Flora, S. Tarkoma, Practical Web- based Smart Spaces, IEEE Pervasive Computing, ISSN: 1536-1268, 2010

[4] Hoyer, Volker, and Marco Fischer. "Market overview of enterprise mashup tools." *Service-Oriented Computing–ICSOC 2008*. Springer Berlin Heidelberg, 2008. 708-721.

[5] M. Blackstock and R. Lea. IoT mashups with the WoTKit. In Internet of Things (IOT), 2012 3rd International Conference on the, pages 159–166, 2012.

[6] Pintus, Antonio, Davide Carboni, and Andrea Piras. "Paraimpu: a platform for a social web of things." *Proceedings of the 21st international conference companion on World Wide Web*. ACM, 2012.

[7] Maximilien, E. Michael, et al. "A domain-specific language for web apis and services mashups." *Service-oriented computing–ICSOC 2007*. Springer Berlin Heidelberg, 2007. 13-26.

[8] Guinard, Dominique, Vlad Trifa, and Erik Wilde. "A resource oriented architecture for the web of things." *Internet of Things (IOT), 2010*. IEEE, 2010.

[9] Fleurey, Franck, et al. "MDE to manage communications with and between resource-constrained systems." *Model Driven Engineering Languages and Systems*. Springer Berlin Heidelberg, 2011. 349-363.

[10] Yu, Jin, et al. "Understanding mashup development." *Internet Computing, IEEE* 12.5 (2008): 44-52.

[11] Wong, Jeffrey, and Jason I. Hong. "Making mashups with marmite: towards end-user programming for the web." *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2007.

[12] Pramudianto, Ferry, Indra Rusmita Indra, and Mathias Jarke. Model Driven Development for Internet of Things Application Prototyping. The 25th International Conference on Software Engineering and Knowledge Engineering (SEKE 2013), Boston, USA

[13] Riedel, Till, et al. "A model driven internet of things." *Networked Sensing Systems (INSS), 2010 Seventh International Conference on*. IEEE, 2010.

[14] Prehofer, Christian. "From the Internet of Things to Trusted Apps for Things." *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013.

[15] Broy, Manfred, et al. "2 nd UML 2 Semantics Symposium: Formal Semantics for UML." *Models in Software Engineering*. Springer Berlin Heidelberg, 2007. 318-323.

[16] Behrmann, G.; David, A.; Larsen, K.G.; Hakansson, J.; Petterson, P.; Wang Yi; Hendriks, M., "UPPAAL 4.0," *Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on* , vol., no., pp.125,126, 11-14 Sept. 2006

[17] OMG, OMG. "Unified Modeling Language (OMG UML)." (2007): 1-2.