MOZART: Design and Deployment of Advanced IoT Applications

Ajay Krishna
Univ. Grenoble Alpes, Inria, CNRS,
Grenoble INP, LIG
Grenoble, France

Michel Le Pallec
Nokia Bell Labs
Paris Saclay, France

Alejandro Martinez
Univ. Grenoble Alpes, Inria, CNRS,
Grenoble INP, LIG
Grenoble, France

Radu Mateescu
Univ. Grenoble Alpes, Inria, CNRS,
Grenoble INP, LIG
Grenoble, France

Gwen Salaün
Univ. Grenoble Alpes, CNRS,
Grenoble INP, Inria, LIG
Grenoble, France

ABSTRACT

The Internet of Things (IoT) aims at sensing and altering our surrounding environment through connected objects to improve everyday life. IoT applications are built using interconnected objects with a goal to provide added-value services. However, there are still challenges in providing a secure, robust and easy-to-use end-user platform for development of such applications. In this paper, we present a end-user tool for supporting the design and deployment of smart home IoT applications. The tool first provides a graphical user interface to specify an IoT application using a rule-based composition language. Automated analysis techniques can then be called for verifying that the designed application is correct (e.g., free of deadlocks). Finally, the tool provides a rule execution engine to support application deployment. The tool is built by implementing a set of components on top of Mozilla WebThings platform, which is a concrete implementation of W3C's Web of Things specification.

CCS CONCEPTS

• **Software and its engineering → Graphical user interface languages**; **Formal software verification**.

KEYWORDS

IoT, composition, formal verification, web of things

1 INTRODUCTION

The Internet of Things (IoT) originally refers to everything connected via the internet. It recently evolved to represent physical devices and software objects deployed on lightweight nodes that communicate with each other and results in new added-value applications built on top of those devices and objects. The growth of connected device market and its central role in many application areas (healthcare, transportation, agriculture, manufacturing, smart homes and cities, etc.) has turned IoT at the heart of the digital economy. The global market value of IoT is projected to reach $520 billion by 2021 [1] and DBS expects ~125 billion connected IoT devices to be deployed by 2030 [11]. People are using connected objects more than ever to build applications which would make their daily lives easier.

The advent of IoT has introduced a number of challenges such as security, privacy, platform heterogeneity, lack of universal standards, device autonomy and application design. In this work, we focus on the aspects of application design. Designing IoT applications tackles different characteristics and problems that have to be worked out to allow anyone to easily build rich, correct, and efficient applications based on interaction between objects. The first issue concerns existing models and languages used for describing objects and more specifically their interfaces. Several levels of expressiveness can be taken into account at this level for specifying functional, semantic or behavioural aspects, and there is no common language recognized yet for writing such models. The second question concerns composition of existing objects. Currently, popular platforms rely on simple rule-based languages that allow one to define a set of event-action rules (when an event occurs, an action is triggered as reaction). These languages are expressive enough for simple applications and scenarios but are not sufficient for more advanced (yet realistic) applications. Another problem concerns correctness of the designed applications, which are intrinsically highly distributed thus making their design possibly erroneous. There is a need for validation techniques at design time to ensure that those applications will behave correctly (e.g., there is no blocking situation known as deadlock). Finally, design of IoT applications must be as automated as possible making the task of end-users simpler. Those techniques should not only support the design of the application but also their final deployment to finish with a running application and not just a model of it.

In this paper, we propose some techniques to support end-users when designing IoT applications that are not simplistic, thus cannot be built by simple rules or existing recipes using popular tools such as IFTTT [13]. We propose a composition language, which allows one to write not only event-action rules, but also to compose these rules to impose an order of execution between two rules, a choice among several rules or a simultaneous execution of several rules. The writing of a composition expression is achieved using a user-friendly interface we implemented, which does not require a high-level of domain expertise. Building such advanced compositions can be error-prone (e.g., resulting into erroneous or deadlocking executions), therefore, we also provide analysis techniques for automatically detecting errors in the composition. Last but not least, our approach supports not only the design of the application but also its deployment.

WWW '20 Companion, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7024-0/20/04.
https://doi.org/10.1145/3366424.3383532

The whole approach was implemented using three different components gathered under the name of MOZART (portmanteau of Mozilla and Advanced Rule Triggers). First, we extended the Mozilla Web of Things (WoT) [4] platform by developing an interface to support advanced compositions. This interface also enables users to perform the successive steps, i.e., design, analysis, and deployment. Second, the analysis part is achieved by encoding the objects and composition expression into a formal specification language and by invoking the verification tools available in the CADP toolbox [5]. Third, application deployment is managed by an execution engine built on top of the Mozilla WebThings platform. For evaluation purposes, we carried out extensive validation on many real-world IoT applications.

The organization of this paper is as follows. Section 2 presents the models for objects and the composition language. Section 3 introduces our tool support. In Section 4, we survey the experiments we carried out for validation of our solution. Section 5 has information on a demonstration scenario. Section 6 overviews related work and Section 7 concludes the paper.

2 MODELS

In this work, objects or things are modelled as specified in the Thing Description (TD) [18] of W3C's WoT framework. The TD specification contains the vocabulary for describing physical and virtual objects, which are composed of one or more objects. This specification is complemented with a behavioural model, which makes explicit the change of state in objects and the specific order in which they occur (e.g., an alarm can be silenced only if it has been triggered beforehand). Concretely, this behavioural model is described in terms of a Labelled Transition System (LTS).

To define a composition consisting of several objects, we propose a simple yet expressive rule-based composition language. This language assumes 'IF *events* THEN *actions*' rules as basic units. A rule is triggered when one or several events are emitted by specific objects and, as a reaction, one or several actions are sent to other objects defined as targets. These rules can be composed to build more complex applications using a set of operators. More precisely, we rely on simple operators that allow one to impose an order of execution between two rules (Sequence), to make a choice among several rules (Choice), to execute simultaneously several rules (Parallel), and loop a part of the expression (Repeat).

An application consists of a set of objects and a composition expression built upon a set of rules. Communication between objects occurs via a gateway. Running of an application involves processing of the composition expression and execution of the individual rules appearing in the expression. When an expression is processed, a subset of the rules is enabled, depending on the way the rules are composed. A rule may contain one or more trigger events and a set of resulting actions. When the events occur, the gateway is notified of the changes, and the actions in the rule need to be executed. As the processing of the composition expression proceeds, a relevant subset of rules is enabled and disabled by the gateway.

3 TOOL SUPPORT

WebThings by Mozilla[1] is a WoT platform for monitoring and controlling objects. It provides Things UI (backed by Things Gateway) to monitor and control IoT objects via a unified web interface. The UI allows users to discover objects available on the network, build rules and deploy them. We extended this platform to support design and deployment of advanced applications specified using the composition language described in Section 2. We chose to extend an existing platform as it provides proven benefits over starting from scratch. First, it saves the effort of building another UI for rule-based automation. Importantly, the list of devices supported by WebThings platform is constantly growing and if we had implemented our own discovery and deployment, additional effort would be spent on adding APIs to support new devices in our platform. Finally, WebThings implements the abstract W3C WoT specification.

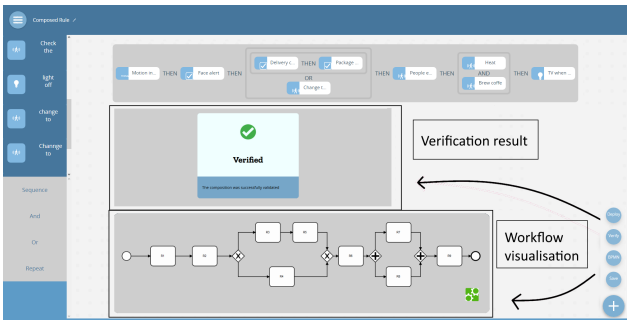

Figure 1: New UI to compose rules

As part of the extension, we developed three major components. First, we designed a new **user interface** (UI) to compose advanced rules. Users create rules using the existing WebThings UI. Once the rules are created, they can use the new interface to compose rules using the composition language. A screenshot from the newly built UI to design advanced applications is shown in Figure 1. Second, we built a **verification component**, which transforms WoT models and composition to formal specification to perform property verification such as the absence of deadlocking situations, completeness of execution, and probability of successful execution. Once the composition is built, the interface allows users to verify if the properties hold true. Upon clicking the *verify* button, users are provided a choice to verify generic properties or to write functional and quantitative properties. As soon as the user input is registered, the tool transforms the composition to process algebraic specification and generates the required model checking scripts as an input to CADP. The results of verification are obtained from CADP and returned to the web interface. If the properties are not satisfied, a diagnostic message is displayed to the users to help them redesign the application so that all properties of interest are satisfied. Third, we built an **execution engine** to activate these rules respecting the composition language semantics. When a composed model is validated, users can proceed to deploy the application. As soon as the *deploy* button is clicked, the newly developed execution engine kicks in. It creates appropriate listeners to capture events and

[1] https://iot.mozilla.org/

executes relevant actions respecting the composition semantics. In addition to these features, users familiar with Business Process Model Notation (BPMN) can visualise the BPMN model of the application as process oriented notation as shown in Figure 1, which can be more intuitive for advanced scenarios [2].

Technology Stack: A simplified view of the components and technologies used in the tool is shown in Figure 2. Mozilla WebThings Gateway is built on NodeJS. Our extension takes advantage of the existing packages available in the platform to implement the execution engine. Rules and composite rules are stored in a file-based SQLite database. BPMN visualisation is handled using bpmn.io JavaScript library. The backend transformation and verification component is implemented as a Spring Boot application hosted on an embedded Tomcat server. Communication with the CADP verification toolbox is done via system calls. The codebase of the implementation is available on Github[2].

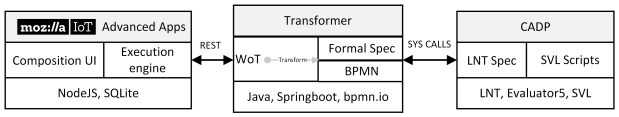

Figure 2: MOZART: Components and technology stack

4 EVALUATION

This section details the experiments which were carried out to evaluate the applicability, usability and performance of the tool.

In order to check the applicability of the tool for advanced applications, a set of 36 scenarios were chosen from the literature. These scenarios can be categorised as advanced as they can not be easily described using simple 'IF *events* THEN *actions*' rules. Two programmers (P1 and P2) with 3 and 1 years of home automation experience, respectively, and knowledge of our work were asked to translate the textual descriptions of the scenarios to single event trigger rules (IFTTT style), Mozilla WebThings rules and as a composition of rules using the operators described in the paper. Initially, P1 and P2 independently classified whether the scenarios could be built using single event triggers, Mozilla WebThings rules and composition of rules in MOZART. Once P1 and P2 completed their designs, their responses were compared with each other. P1 and P2 built were in agreement for 88.89% (κ=0.71) of the scenarios, *i.e.*, their designs were the same or semantically similar. After discussions between the two programmers, a common translation was identified for scenarios that had diverging translations. A plot showing the list of scenarios and if they could be implemented using the three programming models is shown in Figure 3. Except for two scenarios, MOZART can implement all the scenarios compared to IFTTT and Mozilla WebThings.

End-user usability of the tool was measured using two experiments. First, 26 users were involved in an online study. They were given a short training (~10 minutes) on the usage of the tool using a brief description of the tool and of its features. Then, users were shown 8 advanced scenario descriptions in natural language. Further, users were shown how each of these scenarios can be designed using MOZART. After walking through these scenarios, users were

[2]https://github.com/ajaykrishna/mozart/

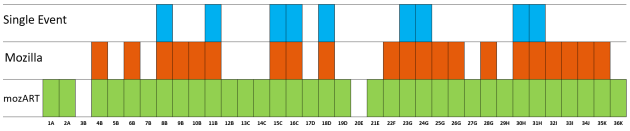

Figure 3: Plot showing the scenarios with presence of bars indicating the possibility of implementing the scenarios.

asked how comfortable they would be to use the tool to build advanced applications. As shown in Figure 4, nearly 60% of the users felt that they could use the tool to build advanced applications with the provided training. As a second experiment, 6 users with varying programming skills (none to expert) from the 26 users were given 2 scenarios described in natural language and were asked to design the application themselves using the tool. The IoT devices required for the scenario were already connected to the WebThings platform. Users had to create individual rules and compose them to build the application. All the users were able to design 2 correct scenarios, which took on average 6 and 8 minutes. The general consensus was that Sequence and Parallel operators were useful in designing applications. Interestingly, non-programmers were not convinced of the Choice operator as it could introduce non-deterministic behaviour thus giving the impression that the actions triggered in the composition are not fully under their control.

Finally, time taken to perform verification tasks was measured. Applications were designed with varying number of objects, rules and operators, and they were checked for deadlock freedom using formal verification tools. The largest example we tried consisted of 16 objects and 12 rules, and verification tasks were performed in less than 3 minutes. In practice, typical applications with advanced scenarios found in the literature [2, 6, 10, 17] rarely require more than 10 objects and 10 rules, and the verification time is quite reasonable (less than a minute) for such applications.

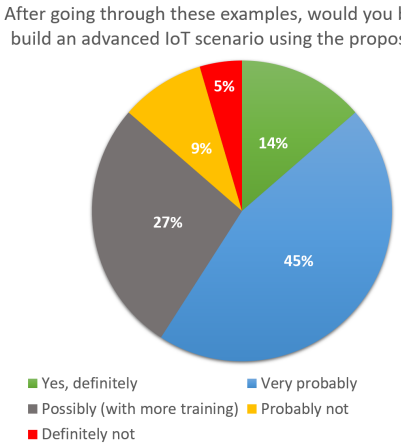

Figure 4: Advanced composition: Usability

5 DEMONSTRATION

Let us consider that a user wants to automate the following scenario: When she/he enters the house, the lights need to be turned on.

Then when she/he switches on the TV, ambient lighting should be turned on and also if the temperature is less than 20 degrees Celsius, heating needs to be turned on. Finally, when the door is closed, TV and lights must be turned off. This scenario can be expressed as a composition of four rules.

```
R1: IF motion(true) THEN light(on)
R2: IF tv(on) THEN ambient_light(on)
R3: IF thermo(temp<20) THEN heater(on)
R4: IF door(closed) THEN tv(off) AND light(off)
```

These rules can be composed as $R1; (R2 \| R3); R4$, where ; and || denote sequential and parallel composition, respectively. At design time, user can check for deadlocks, check if all actions in the composition are reachable, and verify properties such as *lights are never left on once the door is closed*. If the user checks this property of lights being turned off, the tool would indicate that ambient lighting is not turned off after the closing of the door. Once the design is found to be correct, the rules can be deployed through the execution engine. This demonstration can be carried out in a web browser using real and virtual objects as shown in the introductory video[3].

6 RELATED WORK

In this section, we focus on existing tools supporting the design of IoT applications.

Node-RED [7] and IFTTT [13] are two industrial tools that provide graphical support for visually building applications consisting of IoT objects. SmartThings [16] platform provides ability to write compound rules for home automation using SmartRules app [15]. Sharp Tools [14] has a visual builder and dashboard to automate and monitor smart homes. webCoRE [19] is a community rule engine that allows users to create scripts that are interpreted and executed by SmartThings. openHAB [12] is an open source home automation software that allows one to define rules based on events, time and other triggers. Our choice was to propose a more expressive language than those based on rules as IFTTT but avoid to have a full-fledged programming language like webCoRE. A certain level of expressiveness is indeed required for designing advanced IoT applications and as the user-base is consumers, we wanted to keep the learning required to use the tool to minimum. In [3], the authors present a solution to the dynamic composition of services. To do so, they rely on stateful models of services, contextual information, a goal description and planning techniques in order to generate automatically a resulting composition of services. [8, 9] proposes to model objects using behavioural models. An application is then described as a graph of connected objects by explicitly specifying bindings between objects. A compatibility analysis ensures that all the bindings defined in the composition can effectively be executed when the application is deployed. A deployment plan is finally generated and can be executed in order to configure and run the application. The main difference compared to those works is that our approach relies on a high-level language for describing advanced IoT applications and supports this language from design and analysis to final deployment.

[3]https://vimeo.com/379985695

7 CONCLUSION

In this paper, we have presented a solution for supporting end-users when designing new IoT applications using available objects and devices. We have proposed a simple yet expressive enough composition language for describing specific scenarios consisting of successive events and actions organized in a certain order. For simplifying usage of this language by end-users, such compositions can be easily designed using a newly developed graphical user interface. Once the composition expression is defined, analysis techniques can be triggered to check that the intended application fulfils some correctness properties. Once the composition is valid, it can be deployed in a click of a button, through an execution engine. As for implementation, we decided to extend an existing Mozilla WebThings platform, which is a complementary implementation of W3C WoT Candidate Recommendation. We carried out a number of experiments by deploying advanced compositions using the proposed tool, which indicate that users can build verified applications easily and in a reasonable amount of time. The main perspective is to extend this work to support large-scale applications in the Industrial IoT domain.

REFERENCES

[1] Bain & Company. 2018. IoT market size will more than double. https://www.bain.com/about/media-center/press-releases/2018/bain-predicts-the-iot-market-will-more-than-double-by-2021/.
[2] Julia Brich, Marcel Walch, Michael Rietzler, et al. 2017. Exploring End User Programming Needs in Home Automation. *ACM Trans. Comput.-Hum. Interact.* 24, 2 (2017), 11:1–11:35.
[3] Antonio Bucchiarone, Annapaola Marconi, Marco Pistore, et al. 2017. A context-aware framework for dynamic composition of process fragments in the internet of services. *J. Internet Services and Applications* 8, 1 (2017), 6:1–6:23.
[4] Ben Francis. 2019. Web Thing API. https://iot.mozilla.org/wot/
[5] Hubert Garavel, Frédéric Lang, Radu Mateescu, et al. 2013. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *STTT* 15, 2 (2013), 89–107.
[6] Justin Huang and Maya Cakmak. 2015. Supporting mental model accuracy in trigger-action programming. In *Proc. of UbiComp'15*. 215–225.
[7] JS Foundation. 2018. *Node-RED: Flow-based Programming for the IoT*. https://nodered.org/
[8] Ajay Krishna, Michel Le Pallec, Radu Mateescu, Ludovic Noirie, and Gwen Salaün. 2019. IoT Composer: Composition and deployment of IoT applications. In *Proc. of ICSE'19 (ICSE-Companion), Montreal, Canada*. IEEE, 19–22.
[9] Ajay Krishna, Michel Le Pallec, Radu Mateescu, Ludovic Noirie, and Gwen Salaün. 2019. Rigorous design and deployment of IoT applications. In *Proc. of FormaliSE'19, Montreal, Canada*. IEEE Press, 21–30.
[10] Marco Manca, Fabio Paternò, Carmen Santoro, and Luca Corcella. 2019. Supporting end-user debugging of trigger-action rules for IoT applications. *Int. J. Hum.-Comput. Stud.* 123 (2019), 56–69.
[11] Sachin Mittal, Wang Tsz Tam, and Chris Ko. 2018. *Internet of Things: The Pillar of Artificial Intelligence*. https://www.dbs.com/content/article/ePub/aics/insightsreports/sector/63-sector/index.html.
[12] openHAB. 2018. *openHAB*. https://www.openhab.org/
[13] Steven Ovadia. 2014. Automate the Internet with If This Then That (IFTTT). *Behavioral & social sciences librarian* 33, 4 (2014), 208–211.
[14] Sharp Tools. 2018. *Sharp Tools*. https://sharptools.io/
[15] SmartRules. 2018. *SmartRules: Rule your smart home*. http://smartrulesapp.com/
[16] SmartThings. 2018. *SmartThings*. https://www.smartthings.com/
[17] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, et al. 2014. Practical trigger-action programming in the smart home. In *Proc. of CHI'14*. 803–812.
[18] W3C. 2019. Web of Things at W3C. https://www.w3.org/WoT/
[19] webCoRE. 2018. *webCoRE*. https://wiki.webcore.co/webCoRE