R-MOZART: A Reconfiguration Tool for WebThings Applications

Francisco Durán, Ajay Krishna, Michel Le Pallec, Radu Mateescu, Gwen Salaün

R-MOZART: A Reconfiguration Tool for WebThings Applications

Francisco Durán*, Ajay Krishna†, Michel Le Pallec‡, Radu Mateescu† and Gwen Salaün§
*ITIS Software, University of Málaga, Spain
†Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG 38000 Grenoble, France
‡Nokia Bell Labs 91620 Nozay, France
§Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG 38000 Grenoble, France

***Abstract*—The Internet of Things (IoT) is a network of physical devices and software entities that interact together for fulfilling an overall objective and thus providing added-value services. Designing such applications by selecting a set of candidate objects and defining how they interact with one another is a difficult and error-prone task. Moreover, IoT applications are not monolithic applications built once and for all. In contrast, they are constantly modified due to removal, replacement, or addition of new objects during the application's lifetime. In this paper, we present a tool built on top of the WebThings platform, which supports users when they want to dynamically change a running WebThings application. To do so, R-MOZART provides three components for (i) designing the new application using a user-friendly UI, (ii) verifying that this new application respects some consistency properties with respect to the current application, and (iii) deploying this new application in an automated manner. This tool was applied on several smart home applications for evaluation purposes. Video URL: https://youtu.be/bG4oiQUrWSQ**

I. Introduction

The Internet of Things (IoT) is a network of physical devices and software entities that interact together for fulfilling an overall objective and thus provide added-value services. Designing IoT applications by selecting a set of candidate objects and defining how they interact with one another is a difficult and error-prone task. Moreover, IoT applications are not monolithic applications built once and for all. Contrarily, they are constantly modified due to removal, replacement, or addition of new objects during the application lifetime. The reconfiguration of the application must be taken into account in the overall design process. Tool support is also required when dynamically reconfiguring IoT applications to simplify this task for end-users.

Several frameworks, such as IFTTT, Zapier and WebThings, promote to build IoT applications by using "*if event(s) then action(s)*" rules, *i.e.*, if an event is raised, then an action is triggered. In this paper, IoT applications are described by using a composition language built on top of such rules, which provides basic constructs such as the sequence of rules, the choice between several rules, the concurrent execution of several rules, or the repetition of rules. Given such a composition expression, we can rely on an execution platform, such as WebThings, in order to deploy and effectively run the application according to the composition expression defining how objects involved in the application are supposed to interact with one another.

In this paper, we present a tool called R-MOZART, which supports the reconfiguration of IoT applications described using composition expressions. R-MOZART is an extension of MOZART [1] and of the WebThings (formerly Mozilla WebThings) platform [2]. More precisely, R-MOZART implements three new components for supporting the reconfiguration process: a dedicated UI, a verification component, and a deployment manager. At the design level, a new UI allows the user to model the new application, starting from the current one, and adding / removing objects or rules. Once the new application is described, a verification component is called to check whether the reconfiguration properties are satisfied. One of the reconfiguration properties (called seamless) determines if the current state of each remaining object (an object present in the current and in the new application) is reachable in the new application. If this is the case, it means that replacing the current application by the new application can be achieved transparently from the user's perspective. These properties are checked using an encoding of the applications and of the properties into Maude's rewriting logic [3], and using Maude's tools to verify them. Once the reconfiguration has been validated, a deployment manager takes care of the replacement of the current application by the new one, while preserving the consistency of the remaining objects. Note that the only step of our approach requiring human intervention is the design of the new version of the application. The two other steps (verification and deployment) are fully automated by dedicated tools we have implemented and validated on several smart home applications.

The rest of this paper is organised as follows. Section II introduces models of IoT applications. Section III describes the R-MOZART functionalities to support the design of new applications, the verification of reconfiguration properties, and the application deployment. Section IV presents the evaluation of the tool. Section V surveys related work. Section VI concludes the paper.

II. Models

An IoT application consists of a set of IoT objects or *things* interacting all together to fulfil a certain overall goal. In this work, although inspired by the web of things description model [4], we prefer to rely on an abstract model for representing objects. Since the events / actions involved in an object

are executed in a specific order, we describe the behaviour of the objects using a Labelled Transition System (LTS). The transition labels in this LTS are the events or actions associated with the object and the system can move from a state to another by performing an event or an action.

In R-MOZART, an IoT application is described by a set of objects and a composition expression, which acts like an orchestrator indicating how the involved objects interact. We use a simple rule-based composition language for this purpose. This language assumes "*if event(s) then action(s)*" rules as basic elements. A rule is triggered when one or several events are issued by specific objects and, as a reaction, one or several actions are issued to other objects defined as target.

These rules can be composed to build more complex expressions, using operators such as sequence, choice, concurrent execution (parallel) or repetition (loop) of rules.

Let us now explain how an IoT application, consisting of a set of objects and a composition expression, executes. The communication model being asynchronous, each object is equipped with an input message buffer (FIFO). The composition expression and all objects start their execution from their initial states. Then, an application can evolve in two ways: execution of a rule or buffer consumption. In the first case, let us assume a basic rule with one event and one action. If the event appearing in the left part of the rule has been issued, the rule can be triggered and the action appearing in the right part of the rule is pushed to the corresponding object's buffer. In the second case, if there is something in its input buffer, one object can individually consume from its buffer according to its LTS model. A global state consists of the current state of all objects involved in the application and of the progress state of the composition expression.

III. Tool Support for Reconfiguration

This section first introduces the WebThings platform, and then presents the three components supporting reconfiguration in terms of UI, verification and deployment.

A. WebThings

WebThings [2] is a platform for monitoring and controlling devices over the web. It is based on Thing Description (TD). A TD describes the state attributes of an object that can be modified by interacting with the object. In our work, we use the WebThings platform because it is simple to use and extend. The Things UI component in WebThings allows users to build IoT automation in the form of "*If event(s) then action(s)*" Event-Condition-Action (ECA) rules. It also provides web APIs for monitoring and controlling IoT objects. Many of the popular objects are already supported by the platform and more objects are constantly being added.

MOZART [1] is a tool built on top of WebThings to support the design and deployment of complex applications. In addition to individual ECA rules, it allows users to compose these rules using the composition language described in Section II. Composition of rules enables the design of more expressive application scenarios. In this work, MOZART has been extended to support end-to-end reconfiguration. The next subsection presents the reconfiguration support in detail.

B. Reconfiguration Support

Support for reconfiguration requires the implementation of three components. First, at the design level, graphical user interfaces need to be available to specify changes to an existing application. Once the changes are specified, another component is required to verify that the new application satisfies several properties. Finally, a third component is in charge of the deployment of the new application.

User Interface. Initially, a first version of an IoT application is running. When the user starts the reconfiguration process by clicking on the corresponding button, a copy of the current application appears in a new window. Then, (s)he can modify this application by adding or removing objects or rules, and also by changing the way in which the rules are composed. The UI also provides several buttons to initiate the verification or the deployment of the new application.

Verification. Once the redesign is finalised from a modelling perspective, the user can compare the new application with the current one to check the reconfiguration impact. Given the current and new applications, and the global state of the current application, the main reconfiguration property (called *seamless*) determines if the given global state is reachable for objects remaining in the new application. If this is the case, it means that replacing the current application by the new application can be achieved transparently from the user's perspective. We also define two additional properties called *conservative* and *impactful* to check whether all former behaviours can still be executed in the new application, and whether all new behaviours can be executed after the reconfiguration, respectively. These three properties focus on the reconfiguration of an application given a global state. Complementary to these properties, functional properties of interest, like deadlock freeness, can be verified on the new application. When checking these properties, a result indicates whether they are satisfied or not. If they are satisfied, the user can decide to proceed with the deployment. If they are not, (s)he can revise the design or keep the application running as it is.

Deployment. The deployment manager performs two tasks: i) undeploy the removed objects while preserving the state of the remaining objects; ii) set up and deploy the new objects, and start the reconfigured application. First, current states of all objects are stored in a database along with the execution history of the application. This is followed by the disabling of rules. Then, rules are replaced or new rules are created depending on the reconfiguration. New rules are created using the Rules UI, and when they are enabled, event listeners associated to the events in these rules are created. Similarly, when the rules are disabled, their associated event listeners are removed. Here, we say rules and not individual objects because adding or removing objects is a modification to a rule, as objects are part of an event or an action. In other words, adding an object means including the object

in a rule, from the available pool of objects, and removing an object implies that it is no longer used in a rule. Now, these rules need to be deployed for the application to run. Deployment resumes the application from the state where it was before initiating the reconfiguration. Remaining objects maintain their previous states. As for newly added objects, we simulate the execution trace of the current application on the new composition expression. As a result, we obtain the states from where the new objects have to start when deploying the new application. Newly added rules are initialised in disabled states. As a last step, we use the execution history of the current application to compute the progress state from where the new composition expression should start, which allows us to determine the set of rules to be enabled. From here, the execution engine takes care of running the application. It follows the composition expression semantics by enabling or disabling relevant subsets of rules as the execution of the expression progresses.

Implementation Stack. The reconfiguration UI is built on top of the Things UI powered by Node.js. Users can drag and drop rules and composition operators to modify the composition. The reconfiguration check is achieved by transforming the current and new compositions, specified in JSON, to a Maude [3] specification using Java/SpringBoot and Freemarker libraries. Current states of the objects are collected using the monitoring APIs provided by the Things API and stored in an SQLite database. The state of the composition expression is updated by manipulating the event listeners. During deployment, new objects are moved to appropriate states using the control APIs that allow to set object states (e.g., switch on the lamp or change its colour to red).

Figure 1 gives a glimpse of our tool for reconfiguration. On the left of the figure, we can see the available rules and composition operators in the redesign screen. The top of the figure shows the reconfiguration scenario with the current and the new application. In the middle, the response from the analysis can be seen. Options for verification and deployment are shown on the right. The available devices are shown on the bottom of the screenshot.

IV. Evaluation

The tool has been tested to evaluate the usability, performance and correctness of the approach.

Setup. R-MOZART was hosted on a local machine (PC) and on a Raspberry Pi 3, connected to a private wireless network. Then, we added a set of connected devices which included Philips Hue lights, Hue motion sensors, Hue Play lights, connected thermometer, and connected speakers. As the devices were on the same network, they were easily discovered and added to the monitoring interface of the WebThings UI.

Usability. The tool provides an interface based on the WebThings UI elements for reconfiguration, analysis, deployment of reconfigured applications. Six users (age range 23-45) with varying programming skills (none to expert) were given a short training ($\sim$ten minutes) on the usage of the tool and its features. Then, they were given a description in natural language about the reconfiguration of two existing applications having four and five rules in them. The IoT devices required for the new application were already connected to the WebThings platform. Users had to create new rules and update the composition expression. All the users were able to complete the given tasks in under ten minutes (time-based efficiency of 0.24 goals/min) (Figure 2).

Finally, the users were asked a Single Ease Question (SEQ) at the end of the tasks in a 7-point Likert Scale and the users found the interfaces intuitive (average SEQ score: 5.83). Beyond usability, users found the notion of conservative and impactful useful and the utility of seamless reconfiguration was evident for them when we explained the reconfiguration scenarios.

Performance. The time taken to transform the new composition to Maude encoding and to perform the formal analysis for different applications was always in hundreds of milliseconds as the analysis syntactically compares the traces from the global state. Moreover, the objects involved in IoT applications are simple and the number of objects is usually not very high, explaining why the results are almost instantaneous. The readers can find online [5] several applications and the corresponding generated Maude code for verifying properties.

Correctness. Once validated, applications were deployed through the newly developed deployment manager. Reconfiguration is correct / seamless if after deployment of the new application, each remaining object remains in its current state. Upon deployment, we indeed observed that the applications kept working smoothly without any change of state for the remaining objects.

V. Related Work

There are several tools available for end-users to design and deploy IoT applications. IFTTT [6] relies on ECA rules and provides a large repository of pre-defined ECA rules called Applets. The Applets use single event triggers in the ECA rules. Similarly, Node-RED [7] provides Recipes to connect IoT devices visually. OpenHAB [8] is another home automation software that can be programmed by advanced users. It provides rule scripts in the form of *WHEN something-happens THEN do-something* and it allows logical disjunction of multiple triggers in the rules, if-else expressions, for loops, etc. webCoRE [9] is another programmable rule engine to build advanced automation. Samsung SmartThings and Apple Workflow provide automation support in the form of routines. Finally, WebThings in its current form supports rules with multiple triggers and actions. In all these tools, rules are executed in parallel and there is no support for composition of rules. Further, there are no mechanisms to support the reconfiguration of IoT applications ensuring specific properties.

Seamless reconfiguration is not a new notion and was used in several works focusing on dynamic reconfiguration, e.g., [10], [11]. As an example, [11] presents a flexible approach to seamless reconfiguration of Entreprise JavaBeans applications. This work provides generic and reusable procedures for automatically supporting reconfiguration tasks. The

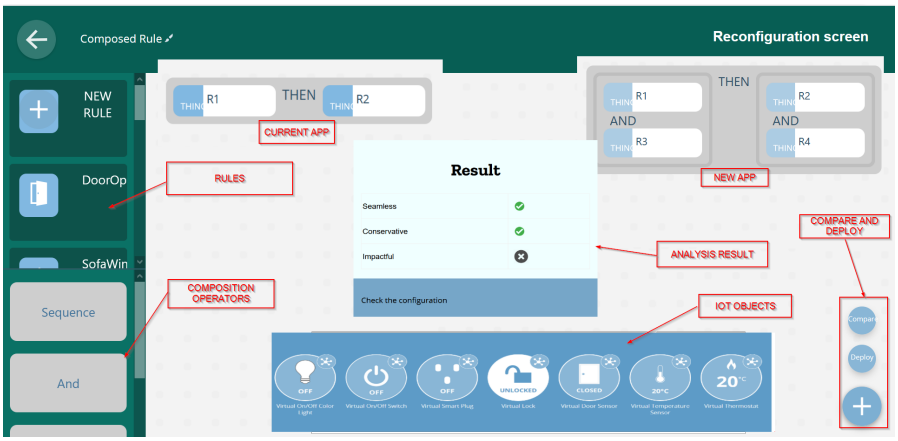

Fig. 1. Screenshots of the reconfiguration tool

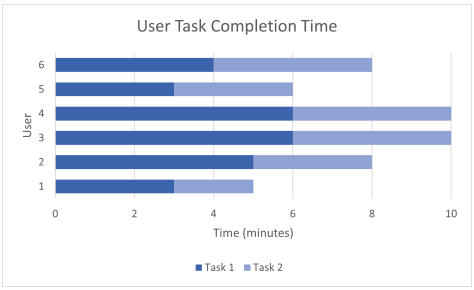

Fig. 2. Task completion time

role of the administrator is reduced to selecting an appropriate strategy and creating a reconfiguration plan that configures a generic procedure for a concrete reconfiguration. Our goal here is to propose a tool providing formal guarantees before triggering the reconfiguration process. Our work presents seamless reconfiguration in the context of the IoT and we also propose two new properties (impactful and conservative).

The approach presented in [12] proposes to extend semantic application descriptions (called recipes) with constraints to enable dynamic and automatic reconfiguration of IoT applications. Using recipes, dynamic choreographies can be created that self-adapt to changing device states without human intervention. [13] introduces the OpenPnP reference architecture, which allows a significant reduction of configuration and integration efforts during industrial plant commissioning. The OpenPnP architecture reduces configuration and installation time by up to 90 percent, while scaling to IIoT systems with many nodes. OpenPnP also provides concepts for replacing malfunctioning devices. In our work, we propose analysis techniques for reasoning on the behaviour of the application before deciding actual reconfiguration, and a solution to deploy this new application on the WebThings platform.

VI. Concluding Remarks

In this paper, we have presented R-MOZART, an extension of WebThings and MOZART to support the reconfiguration (design, verification, deployment) of WebThings applications modelled using compositions of ECA rules. The main perspective of this paper is to extend this work with quantitative analysis results in order to check if the reconfiguration also preserves quality-of-service properties.

VII. Data Availability

The code related to the tool is available at https://zenodo.org/badge/latestdoi/312541691.

References

[1] A. Krishna, M. L. Pallec, A. Martinez, R. Mateescu, and G. Salaün, "Mozart: Design and deployment of advanced iot applications," in *Companion Proceedings of the Web Conference 2020*, ser. WWW '20. New York, NY, USA: ACM, 2020, p. 163–166.

[2] WebThings, "WebThings Open Platform," 2021. [Online]. Available: https://webthings.io/

[3] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, Eds., *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, ser. LNCS. Springer, 2007, vol. 4350.

[4] W3C, "Web of Things at W3C," 2020. [Online]. Available: https://www.w3.org/WoT/

[5] F. Durán, A. Krishna, M. L. Pallec, R. Mateescu, and G. Salaün, "R-MOZART: A Reconfiguration Tool for WebThings Applications (Online Code)," https://github.com/ajaykrishna/rmozart/tree/main/maude, February 2021.

[6] IFTTT, "Do more with the things you love," 2020. [Online]. Available: https://ifttt.com/

[7] JS Foundation. (2020) Node-red: Flow-based programming for the IoT. [Online]. Available: https://nodered.org/

[8] openHAB. (2020) openHAB: Empowering the smart home. [Online]. Available: https://www.openhab.org/

[9] webCoRE. (2018) webcore: The web community's own rule engine. [Online]. Available: https://wiki.webcore.co/webCoRE

[10] L. Rosa, L. E. T. Rodrigues, and A. Lopes, "A Framework to Support Multiple Reconfiguration Strategies," in *Proc. of Autonomics 2007*, ser. ACM International Conference Proceeding Series, vol. 302. ACM, 2007, p. 15.

[11] T. Vogel, J. Bruhn, and G. Wirtz, "Autonomous Reconfiguration Procedures for EJB-based Enterprise Applications," in *Proc. of SEKE'2008*. Knowledge Systems Institute Graduate School, 2008, pp. 48–53.

[12] J. Seeger, R. A. Deshmukh, V. Sarafov, and A. Bröring, "Dynamic IoT Choreographies," *IEEE Pervasive Computing*, vol. 18, no. 1, pp. 19–27, 2019.

[13] H. Koziolek, A. Burger, M. Platenius-Mohr, J. Rückert, and G. Stomberg, "OpenPnP: A Plug-and-produce Architecture for the Industrial Internet of Things," in *Proc. of ICSE'19*. IEEE / ACM, 2019, pp. 131–140.