IoT Composer: Composition and Deployment of IoT Applications

Ajay Krishna*, Michel Le Pallec†, Radu Mateescu*, Ludovic Noirie† and Gwen Salaün‡
*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) consists of interconnected physical devices and software components. These connected things or objects exchange information in order to provide an end-user service. To fulfil this objective, such applications have to be designed by composing existing objects. However, this is a very difficult task mostly due to the heterogeneity and diversity of available objects. The IoT Composer tool was developed for supporting the development of IoT applications by first providing a behavioural model for objects and their composition. IoT Composer is developed as a web application that provides graphical support for composing available objects. The tool also provides automated validation techniques for verifying that the composition behaves correctly. Finally, a deployment plan is generated and can be run for effectively binding and instantiating all objects involved in the composition. IoT Composer was applied successfully to several real-world case studies.**
Video URL: https://youtu.be/6Cn3CUm5-qU

***Index Terms*—IoT, behavioural models, composition, formal verification, deployment**

I. Introduction

The Internet of Things (IoT) is a network of physical devices and software components that are connected together in order to exchange information and to fulfil an IoT service. IoT applications are supposed to empower interconnected objects in order to build powerful and added-value services. The hardware devices used for building IoT applications are already available, but there are only a few management systems that support the design, deployment and maintenance of such applications. These tasks are particularly ambitious due to the specificities of IoT applications that are by nature highly distributed, heterogeneous and dynamic. Indeed, IoT applications induce a high level of concurrency, distribution, and collaboration. Designing concurrent applications is by nature a difficult and error-prone task. Moreover, IoT applications have to deal with heterogeneous IoT hardware and communication layers. Therefore, there is a need for a common model for specifying all objects and the way they interact altogether. Last but not least, modern systems are no longer designed and implemented once-for-all, but they may evolve over time, which requires simple techniques for updating and changing those applications.

In this paper, we present the IoT Composer tool, which supports all the steps of the design, composition, and deployment of an IoT application by selecting, configuring and binding available objects. Figure 1 gives an overall view of the IoT Composer functionalities. Given a repository of objects, the first step is to select some objects as candidates to participate in the composition-to-be. This part of the approach is achieved by relying on an existing recommendation system [1]. Then, IoT Composer graphically exposes the interfaces of the selected objects and allows the user to bind interfaces with respect to the IoT service expected from the composition. It is worth noting that only this first step of the approach requires human intervention, all the following steps being fully automated. In a second step, given a set of selected objects and a set of bindings, we generate code in the LNT process algebraic specification language [2]. This specification is fed as input to the CADP verification toolbox [3] in order to check several properties of interest. The most important one is *compatibility*, which consists in checking whether the set of objects and bindings will behave correctly, avoiding erroneous situations. The tool also allows the verification of other properties such as the absence of deadlocks. If the selected objects and bindings are not satisfactory, the user can return to the first step to update the selection or bindings wrt. the feedback given during the verification step. The final step of the approach aims at generating a deployment plan that is executed for effectively configuring and running the application.

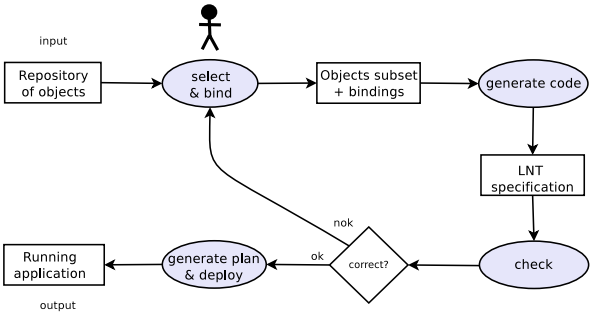

Fig. 1. Overview of the tool functionalities.

IoT Composer is a web application with an intuitive user interface. It has been applied to several real-world case studies

for validation purposes and it turns out to be very useful as it simplifies the design of new IoT applications. Note that our approach supports the development of IoT applications from the beginning (selection) to the end (deployment). In doing so, it also ensures reliability of the applications by formally verifying the proposed solution in the background. The targeted users are any end-users who may want to deploy reliable added-value IoT services on top of objects available in their smart homes. The tool is available as open source online [1] with several examples we used during its evaluation.

The rest of this paper is organized as follows. Section II introduces the models used for specifying objects and their composition. This section also presents the notion of compatibility that allows one to check that the composition executes as expected. Section III describes the IoT Composer tool. Section IV presents experiments performed on several real-world smart home use-cases. Section V overviews related work and Section VI concludes the paper.

II. Models and Compatibility

As far as models are concerned, an IoT object is modelled as a Labelled Transition System (LTS), i.e., a state-transition graph representing the states of the object and its interactions with the environment. Each transition is labelled by an action corresponding to an input or output message. An LTS can be viewed as a behavioural interface description language.

A composition of objects, also referred to as a composite service, is then built by combining various objects, which are bound to each other relying on their LTS interfaces. The interfaces in the LTS are derived from the application interfaces of IoT objects. Given two objects, a binding aims at connecting one output message of one object with one input message of another object. A composition is thus defined by a set of objects and a set of bindings relating these objects. In this work, we assume that the objects involved in a composition interact using binary communication: an interaction occurs between one output message and one input message (same message name) belonging to two different objects. Additionally, we consider handshake communication, also known as synchronous communication model. This means that the sender is blocked until the receiver is able to interact with it. Then, both objects synchronize at the same time.

Compatibility checking aims at ensuring that a service composition involving a set of objects and a set of bindings is correct, i.e., all bindings can effectively be executed and all reachable actions unbound in the composition do not prevent the bindings to be executed (Unbound interfaces refer to the interfaces which are not involved in the composition). This compatibility notion is checked by first automatically encoding the set of behavioural models and the set of bindings into the LNT process algebraic specification language [4]. Then, we make use of the existing tools available in the CADP toolbox [3], which accepts LNT as input, for (i) compiling the LNT specification into an LTS describing all the possible executions of the composition (called composition LTS), and for (ii) traversing this LTS in order to verify that all bindings do appear in the composition LTS.

Note that we take advantage of this LNT encoding for verifying other properties of interest. This is the case, for instance, of the absence of deadlocks, which can be automatically analyzed using model checking techniques available in CADP.

[1] https://ajaykrishna.github.io/iotcomposer/

III. IoT Composer

The IoT Composer tool aims at assisting the user during the composition and deployment tasks when developing a new IoT application. Indeed, end-users may not have the required skills to build reliable and bug-free compositions of objects. The tool guides the user towards reliable application design by hiding the underlying complexities. It is hosted as a web application on an Apache Tomcat server with a simple user interface based on jQuery and Semantic UI. The data format used is JSON and the integration with other services is achieved via REST API calls.

IoT Composer performs three major functions: (i) composition by allowing users to compose IoT objects through binding devices spread over different smart environments, (ii) compatibility check to ensure correctness of the designed IoT application, and (iii) automated deployment of the validated composition. Figure 2 shows a screenshot of the web application and of its functionalities on a simple example involving four objects.

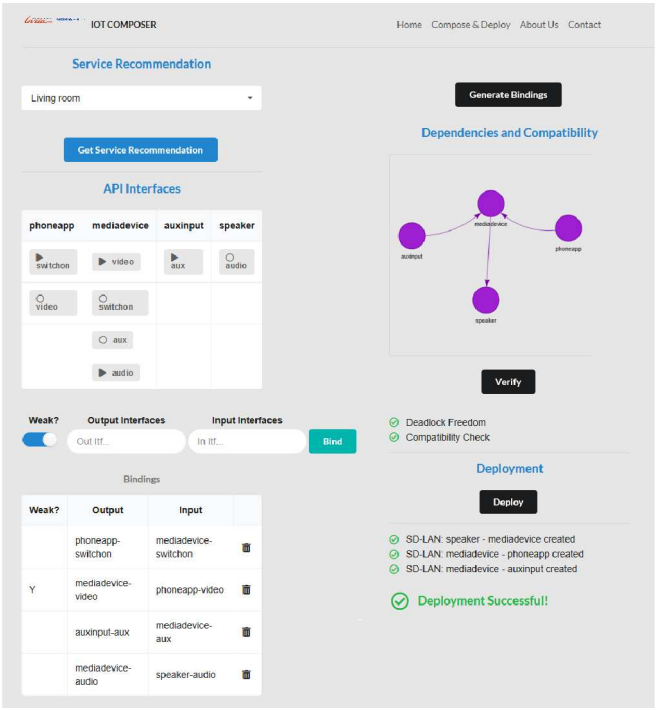

Fig. 2. Screenshots of the web interface.

The very first step (object composition) relies on the recommendation service presented in [1]. The authors proposed

an automated recommendation of IoT objects based on the interaction between objects and their environments. We use this approach for an early identification and selection of objects suitable for the design of an IoT service. When the user has selected some of the objects suggested by the recommendation system, the tool loads in the web interface this list of objects with the set of input/output messages provided in their respective application interfaces. From this set of recommended objects, the user can then graphically define a set of bindings between these interfaces to generate a composition (selected objects + bindings). The defined bindings are stored in a JSON file. The web interface provides an option to display in the web application the graph making explicit the dependencies among objects.

In the next step, the tool connects to the backend formal verification toolset (CADP) to check the compatibility of the composition. Internally, the tool takes JSON models corresponding to the objects and the file storing the JSON bindings as input. Then, it performs a model-to-model transformation to generate the required LNT code and Script Verification Language (SVL) [5] scripts for automating the whole analysis step. The result of the compatibility check (true or false) is returned to the user. In case of an incompatible composition, the user is able to restart the composition process. (S)He can update the bindings or choose a new set of objects and check if they are compatible. When a compatible solution is found, (s)he can proceed to the (final) deployment step.

In the final step, a deployment plan is first generated by relying on a set of defined actions involving both application and network layers. This set of actions (add object, bind objects and start object) must be applied in a specific order for respecting the objects' functional dependencies. This plan is obtained considering the composition as a directed graph (objects as nodes and bindings as edges) and applying topological sorting in order to extract the sequence of operations to apply. Given such a plan, the tool is interfaced with Majord'Home [6], [7], a modular IoT platform to deploy IoT applications. Majord'Home follows the Software Defined Network (SDN) [8] paradigm, which offers simplified interfaces for the network configuration of IoT services involving multiple smart environments. For each IoT service, a Software-Defined LAN (SD-LAN) is setup by configuring the residential gateways which delineate different local networks. Using this SD-LAN functions, connected objects spread across different smart environments can communicate as if they were in a same smart environment. Additionally, the SD-LAN solution allows a connected object to interact with different IoT services, yet preserves network isolation and discovery properties. IoT Composer manages and controls SD-LANs by using custom REST APIs. The overall integration of the IoT Composer components with other services (recommendation system, CADP, Majord'Home) is shown in Figure 3. Each component or service indicates its inputs and outputs as well as its implementation technology stack.

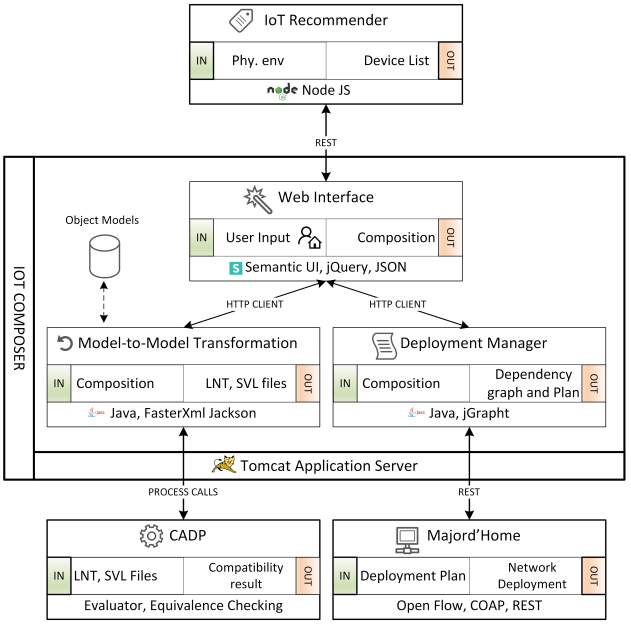

Fig. 3. Tool components and integration with various services.

IV. Evaluation

In order to validate our approach, we relied on various IoT objects available in a smart home environment and designed several IoT services by composing these objects. Some of the experimental results are shown in Table I. The host machine was running Xubuntu 18.04 on a hardware consisting of Core i7-7600U processor, 256GB M.2 PCIe SSD, and 16GB of RAM. In Table I, *Objs* denotes the number of IoT objects used in the composition. *Bind* refers to the number of bindings. The next two columns denote the number of states and transitions in the composition LTS. The time taken to generate the LTS (in seconds) and the time taken to perform the compatibility check (in seconds) are shown in the last two columns. The deployment time is not shown in this table. The plan generation time is negligible for dozens of objects since it is the time required for topological sorting, which is linear with respect to the number of objects and bindings. The deployment itself is achieved via the Majord'Home platform [7]. It takes a few seconds and this time does not depend on the number of objects.

As it can be observed in Table I, the time taken to generate the LTS and check the compatibility does not proportionally increase with the number of objects and bindings. It takes at most 12 seconds for the most complex example (20 objects), which is reasonable since this check is performed at design-time before deploying the IoT application. The final examples in Table I take more time because they involve more objects and therefore generating the corresponding composition LTS and checking compatibility are more time-consuming. We originally targeted to compose and deploy IoT services in the residential context (smart homes). In such a context, one may

Use case	Objs	Bind	LTS States	LTS Trans	Gen	Compat
Thermo	3	4	3	7	2s	4s
SmartDoor	3	4	8	13	2s	5s
LightSense	4	6	4	8	2s	5s
BabyMonitor	4	6	13	17	3s	6s
SmartAccess	5	8	8	9	2s	5s
MultiDoor	6	6	6	7	2s	4s
MultiCase	10	12	36	154	2s	5s
MultiCase2	13	15	176	892	4s	9s
IndepCase	16	18	876	5134	5s	11s
IndepCase2	20	24	10501	79886	5s	12s

TABLE I
EXPERIMENTS SHOWING DIFFERENT IOT USE CASES.

have a home with dozens or even hundreds of objects, but for a given concrete application it would barely need more than a few objects. To sum up, the results of these experiments confirm that, as far as scalability is concerned, the approach is satisfactory for IoT deployments in the smart home context.

V. RELATED WORK

In this section, we compare IoT composer with existing tools supporting the design of new applications by composition of IoT objects.

Node-RED [9] is a programming tool for wiring together hardware devices, APIs and online services. It provides a browser-based editor that makes it easy to wire together flows using a wide range of nodes. The final application can be deployed to its runtime in a single-clic. The light-weight runtime environment is built on Node.js, and can be run at the edge of the network on low-cost hardware (such as Raspberry Pi) or in the cloud. IFTTT [10] (If This Then That) is a free web-based service to graphically create chains of simple conditional statements, called applets. An applet is triggered by changes that occur within other web services. In addition to the web-based application, the service runs on iOS and Android. Zenodys [11] proposes a visual IoT platform for easily building IoT applications. Their approach relies on connectors for collecting and binding data sources. Deployment is supported as well as a set of data analytics in order to monitor running applications. Compared to these three tool-supported solutions, a first difference concerns modelling aspects. None of them assume that objects can be described using behavioural models whereas in our approach some services provided by an object should be executed in a certain order. This assumption particularly makes sense when focusing on composite applications. In addition, these solutions do not provide any support for checking that the application preserves some correctness criteria before deployment.

In [12], 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. Similarly to [12], we rely on behavioural models of objects. In our approach, we preferred semi-automated composition techniques (bindings required as input) to keep the user in the loop and we provide automated techniques for verifying correctness of the composition. We also propose full automation of the final application deployment, which makes our approach supporting the development of IoT applications from the selection of objects to their deployment.

VI. CONCLUDING REMARKS

IoT Composer provides a solution for supporting the design and deployment of IoT applications. It allows developers to build correct compositions of software objects and devices by using behavioural modelling languages and automated verification techniques. The compatibility check ensures that all the bindings defined in the composition can effectively be executed when the application is deployed. It also checks that the composition is free of deadlocking situations. Once the composition is validated, a deployment plan is generated and can be executed in order to effectively configure and run the application. The IoT Composer tool is available online and has been applied successfully on several real-world case studies.

In the future, we first plan to extend the IoT composer tool to provide new functionalities supporting the reconfiguration (addition and removal of objects) of a deployed and running application. This should be possible via the web application and must preserve the level of correctness ensured during the deployment phase. A second perspective of this work concerns the verification step. Beyond analysis of compatibility and other properties of interest, we would like to provide quantitative verification techniques to carry out QoS analysis and performance evaluation of the IoT applications. This would require extending the models of IoT objects and composition with quantitative information (e.g., probabilities, costs, etc.).

REFERENCES

[1] M. L. Pallec, L. Noirie *et al.*, "Digital assistance for the automated discovery and deployment of IoT services," in *Proc. of ICIN'18*, 2018, pp. 1–3.

[2] H. Garavel, F. Lang, and W. Serwe, "From LOTOS to LNT," in *ModelEd, TestEd, TrustEd*, 2017, pp. 3–26.

[3] H. Garavel, F. Lang *et al.*, "CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes," *STTT*, vol. 15, no. 2, pp. 89–107, 2013.

[4] D. Champelovier, X. Clerc *et al.*, "Reference Manual of the LNT to LOTOS Translator (Version 6.7)," 2018, INRIA/VASY and INRIA/CONVECS, 153 pages.

[5] H. Garavel and F. Lang, "SVL: A scripting language for compositional verification," in *Proc. of FORTE'01*, ser. IFIP Conference Proceedings, 2001, pp. 377–394.

[6] M. Boussard, D. T. Bui *et al.*, "Software-Defined LANs for Interconnected Smart Environment," in *Proc. of ITC'15*, 2015, pp. 219–227.

[7] M. Boussard, D. T. Bui *et al.*, "The Majord'Home: a SDN approach to let ISPs manage and extend their customers'home networks," in *Proc. of CNSM'14*, 2014, pp. 430–433.

[8] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," *ONF White Paper*, vol. 2, pp. 2–6, 2012.

[9] JS-Foundation. (2018) Node-RED: Flow-based Programming for the IoT. [Online]. Available: https://nodered.org/

[10] S. Ovadia, "Automate the internet with if this then that (IFTTT)," *Behavioral & social sciences librarian*, vol. 33, no. 4, pp. 208–211, 2014.

[11] Zenodys. (2018) IoT Platform. [Online]. Available: https://www.zenodys.com

[12] A. Bucchiarone, A. Marconi *et al.*, "A context-aware framework for dynamic composition of process fragments in the internet of services," *J. Internet Services and Applications*, vol. 8, no. 1, pp. 6:1–6:23, 2017.