IoT Compositions by and for the Crowd*

Ignacio Mansanet, Victoria Torres, Pedro Valderas, and Vicente Pelechano

Universitat Politècnica de València, Spain
Pros Research Center
{imansanet,vtorres,pvalderas,pele}@pros.upv.es

Abstract. The Internet of Things (IoT) offers a new eco-system of heterogeneous and distributed services that is available anytime and anywhere and that can be potentially accessed by any properly connected device. However, these available services are usually consumed in isolation, missing the potential that their combined usage can bring as new added-value services. In addition, the massive end-user adoption and usage of smartphones together with their powerful capabilities turn this type of devices into a promising platform to develop and execute these added-value services compositions. Moreover, end-users are nowadays getting more and more familiar with technology, fact that allows them to participate more actively in the development of new types of applications. However, this will not happen until we provide end-users with more powerful and easy-to-use tools. To this end, this paper presents an architectural solution to allow end-users building IoT services compositions by just focusing on domain-logic issues.

Key words: Service Compositions, IoT, End-user Development

1 Introduction

The advent of technological advances and the changes that this produces in human behaviour are having a big impact in society. In particular, in the way humans, and more specifically users without IT knowledge, are interacting with devices and applications. On the one hand, technological advances have empowered end-users with tools and gadgets that a few years ago were only accessible to professionals. In fact, nowadays, end-users can afford powerful devices that offer a wide range of features (e.g., media support, bluetooth connectivity, voice recognition, fingerprint identity sensors, etc.) and applications (e.g., weather forecast, games, calendar, health apps, etc.). In addition to this, the Internet of Things (IoT) offers a new eco-system of heterogeneous and distributed services that is available anytime and anywhere and that can be potentially accessed by any properly connected device. However, currently, all these services are being used as stand-alone units, losing the potential added-value that their combined

* This work has been developed with the support of MINECO under the SMART-ADAPT TIN2013-42981-P project and the SITAC IPT-2012-0839-430000 project and co-funded by the ERDF.

usage can bring to a specific user or community. On the other hand, users are getting more and more familiar with all these technological advances, fact that has allowed them to overcome the fear of technology. As a result, end-users are changing the way in which they interact with all this technology, playing now a more active role in their management and usage. Moreover, considering that end-users are the ones having the domain knowledge required to build applications, they seem to be the most adequate players to specify how all these reachable services could be composed to create such new added-value services/applications. However, to successfully achieve this, end-users need to be supported by tools that hide all technological issues, allowing them to just focus on domain-related issues of services compositions. Even though the research community is already investigating and making progresses on the development of services compositions tools for mobile devices [1, 2, 3, 4], these are not yet powerful enough nor easy-to-use to be transferred to industry and adopted by end-users. In fact, the most challenging issues that these tools have to face include: (1) technology aspects should be completely hidden to end-users, (2) code generation should be automatically performed along the whole process, and (3) end-users should just focus on aspects related to the domain of the application being defined.

Within this context, this paper presents the architecture of a solution that endows end-users with the necessary tools to create applications by composing services that are available in the IoT. This work is being developed in the context of the SITAC European project[2], where end-users and social networks turn into key players in the application development process.

The remainder of the paper is structured as follows. Section 2 provides an overview of the architecture of the proposed solution. Then, sections 3, 4, and 5 go a bit deeper into the three different building blocks that make up the proposed architecture. Finally, section 6 summarizes and concludes the paper.

2 Architecture Overview

This section provides an overview of the different building blocks that have been proposed to properly support end-users in the composition and execution of services through a mobile device such an a smartphone. These building blocks give support to end-users in two different phases of the development process, i.e., at *design* and *run* time. While at design time end-users define new services/applications from the composition of services created previously by developers or other end-users, at run time services/applications are executed according to the logic defined in their underlying compositions.

Fig.1 shows the three building blocks that make up the proposed architecture as well as the different roles that interact with them, i.e., end-users and professional developers. On the one hand, professional developers register new services through the *Registration Front-End* (step 1) into the platform to provide access to data and functionality coming from physical and logical sources

[2] http://sitac.wp.tem-tsp.eu/

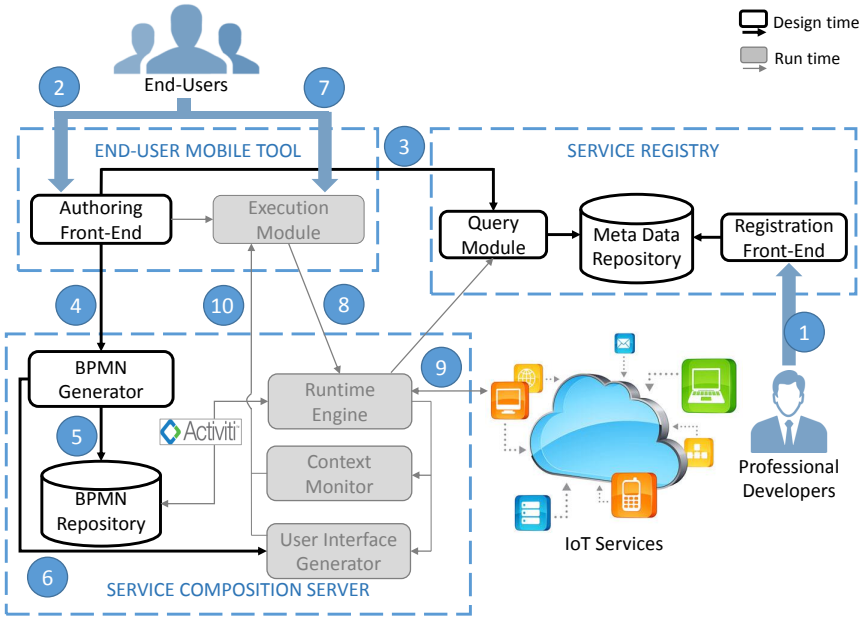

Fig. 1. Architecture overview

(e.g., light sensors or energy consumption) and devices (e.g., electric or mechanical actuators). On the other hand, end-users design and create new added-value services/applications through the *Authoring Front-End* (step 2) from the services that are available in the Service Registry (Step 3). Once service compositions are finalized, these are sent to the *BPMN Generator* component to produce a complete and executable BPMN description of the composition (step 4). The BPMN executable description is then stored in the *BPMN Repository* (step 5) so it can be deployed and executed afterwards on-demand. In parallel to tasks 4 and 5, the *User Interface Generator* configures all the predefined templates to generate all the user interfaces that are needed to properly execute the services that make up the running composition (step 6). From now on, end-users can already execute the compositions by means of the *Execution Module* (step 7). This module interacts with the *Runtime Engine* (step 8) which is in charge of deploying and executing the selected compositions as well as to invoke the services that make up the composition (step 9). Finally, the *Execution Module* interacts with the *Context Monitor and User Interface Generator* (step 10) to check, if necessary, end-user environment conditions and to serve end-users with the appropriate user interfaces respectively. Next sections explain in more detail each of these building blocks.

All these building blocks are organized in five components as shown in Fig.2. While components 1, 2 and 4 (service registry, service composition and application execution respectively) require human intervention, components 3 and 5

(service composition deployment and UI automatic generation respectively) are automated so their work is kept hidden to the end-user. This figure also shows the different artifacts that are produced and consumed along the process.

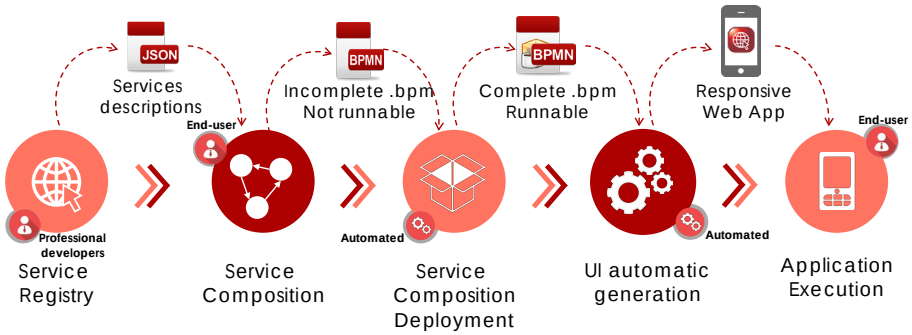

Fig. 2. Artifacts, agents and phases involved in the process

3 End-User Mobile Tool

The end-user mobile tool is a tool made up of two components, one that allows end-users to create and share service compositions (c.f. Section 3.1) and another one to run them as web applications (c.f. Section 3.2). Both components have been designed to be run as responsive web applications. The technology used to implement these web applications include HTML5, CSS3, and Javascript. The main advantage of applying this adaptive design is that a unique implementation of the application can be accessed from different types of devices (e.g., smartphones, tablets and/or desktop computers) in an appropriate way, i.e., adapting content and the interaction with the user according to the specific characteristics of the device being used. As a result, end-users experience a more enjoyable access to the applications according to their current needs. In addition, this solution results more agile for end-users since avoids installing and updating the tool as it would happen with native applications. Next subsections provide more details about the two components that make up this tool.

3.1 Authoring Front-End

Considering the premise that most end-user will use mobile devices to access this component, its design was made taking into account not only good practices on end-user development but also making use of change patterns [5]. On the one hand, the application of end-user development techniques [6, 7, 8, 9, 10] allows us to build a more friendly and usable tool. In this case, the set of guidelines applied in the design of this component include the usage of the catalogue pattern, which offer end-users a set of predefined elements to face the selection barriers [8, 9],

tabs which improve the learning and memorization of the steps to be performed and keep end-users from missing important things [9], textual descriptions of the results in order to facilitate application understanding, and designing all the interfaces following a similar structure. This will facilitate user interpretation of the information.

On the other hand, the composition abstractions taken from the set of change patterns developed by Weber et al. [11] endow end-users to create compositions faster and free of errors. These abstractions refer to patterns that allow inserting fragments to the composition of services in serial or parallel, and to embed fragments either in a loop or in a contional branch.

Once end-users finalize the composition of services, the development process to create ready-to-use services and/or applications continues with a set of automated tasks that are transparent to the end-user. These tasks are performed by the Service Composition Server as described later in Section 4.

3.2 Execution Module

This component allows end-users running service compositions by means of their associated web applications. In particular, this component is in charge of interacting with the Service Composition Server (c.f. Section 4) in order to execute a specific composition. This implies two main tasks: (1) to inform the server about the composition that end-users want to execute and (2) to act as a proxy between end-users and the server during execution. Regarding this last task, note that service compositions are executed by the Service Composition Server, but they may include the execution of services that require some input data from end-users or need to sense the end-user environment conditions (e.g. current location). In this case, appropriate user interfaces are served by the *User Interface Generator* (c.f. Section 4.3) to allow the end-user the execution of the current service.

4 Service Composition Server

This component participates in the development process at both, design and run time, making all the technology related issues transparent to the end-user.

On the one hand, at design time, the Service Composition Server is in charge of (1) completing the compositions created by end-users to obtain a BPMN executable description (e.g., implementing the java classes needed to invoke the corresponding web services), (2) deploying these compositions in a process engine to make them ready when end-users need to execute them, and (3) generating the web application that will allow end-users running the associated services composition (see Fig.3).

On the other hand, at run time this component is in charge of invoking the services that conform the BPMN composition. This is done by a Java web application deployed into a Tomcat server. We have chosen this technology in

order to reuse the Web version of the Activiti BPMN engine, which allow us to execute BPMN descriptions. In addition, Activiti is complemented with three modules to support the execution of services at runtime. These are explained in the following subsections.

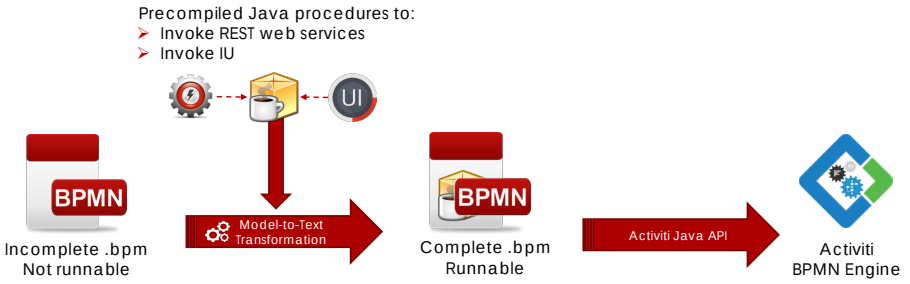

Fig. 3. BPMN deployment process

4.1 Runtime Engine

BPMN service compositions are basically defined from a set of activities, and other constructors that define the composition logic. BPMN provides mechanisms to semantically describe these activities, but there is no support to include the invocation data that is required when the service associated to a process activity has to be executed (e.g. service url, parameters, user and password, etc.). In addition, a composition may include conditions that can be defined over contextual properties that must be obtained in runtime (e.g. user position, temperature, time and date, etc.). The runtime engine is in charge of providing Activiti with this information by interacting with the Context Monitor and the Service Registry.

4.2 Context Monitor

This module is in charge of managing context properties over time. These properties are data such as end-user preferences, locations, environment conditions, system state, and so on. This module is based on the context monitor that was developed in a previous work in order to manage context in pervasive systems [12]. It has been adapted to interact with end-user's mobile device.

4.3 User Interface Generator

As it has been introduced above, services that are included in a composition may need some data from end-users in order to be executed. For instance, a service that allows end-users to book a hotel room need to know the city, and the check-in and check-out dates of the stay. This data must be introduced by

end-users at runtime through the platform they are using to execute service compositions. Thus, if any of the services used in the composition require data from the end-user, appropriate user interfaces are generated and configured from existing templates. To do so, this module interacts with the runtime engine, which passes it the service meta-data available at the service registry (i.e., the json description of the service) in order to analyse service parameters and generate the proper user interface. As shown in Fig.4, a model-to-text transformation is in charge of generating the user interfaces needed to execute the composition.

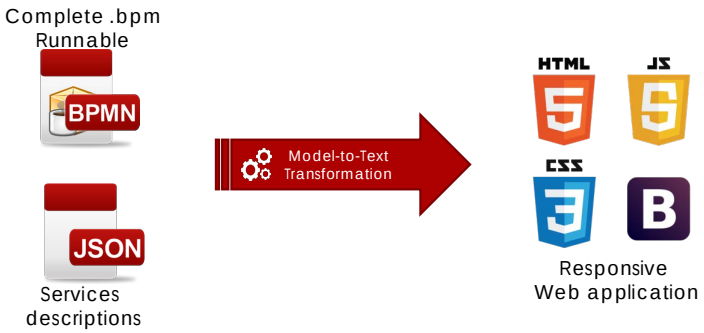

Fig. 4. IU generation process

5 Service Registry

Its main goal is to store meta-data about both services created by professional developers and compositions made by end-users. The registered meta-data includes information of two types: (1) semantic information of a service or a composition, which is used by end-users in order to create a composition through the Authoring Front-end; and (2) invocation data such as service url, number of parameters and type, or user and password, which is used by the Runtime Engine module of the Service Composition Server in order to execute services. Registry's data is managed by to modules: the *Query Module*, which provides the data when it is required, and the *Registration Module*, which is in charge of inserting new data into de registry. If this new data corresponds to a newly created end-user service composition, it is provided by the Authoring Front-end. In order to insert data about services created by professional developers a web-based Registration Front-end is available.

The end-user mobile tool, the service composition server and the service registry need to interact among them in order to both create service compositions and execute them. These interactions are done through the HTTP protocol by using GET connections in order to retrieve data, and POST connections in order to send it. The data that is interchanged is defined by using JSON objects. In addition, note that the Service Registry, together with the Service Composition

Server act as a mediator between end-users and real service implementations, abstracting them from technological issues when consuming services in compositions.

6 Conclusions

This paper has presented the architectural solution that has been designed to support end-users in the composition of IoT services by using mobile devices. The design has been performed taking into account that this solution is mainly targeted to end-users, which means that all technological-related issues have to be hidden to end-users. To do so, all the proposed techology-related tasks are delegated to components that work automatically in a transparent way to the end-user. In fact, end-users should only focus on the domain-specific aspects of service compositions; i.e., they just need to select and combine the services from the registry to create new added-value services.

References

1. Danado, J., Davies, M., Ricca, P., Fensel, A.: An authoring tool for user generated mobile services. In Berre, A., Gmez-Prez, A., Tutschku, K., Fensel, D., eds.: Future Internet - FIS 2010. Volume 6369 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 118–127
2. Cuccurullo, S., Francese, R., Risi, M., Tortora, G.: Microapps development on mobile phones. In Costabile, M., Dittrich, Y., Fischer, G., Piccinno, A., eds.: End-User Development. Volume 6654 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 289–294
3. Maximilien, E.M.: Mobile mashups: Thoughts, directions, and challenges. In: ICSC, IEEE Computer Society (2008) 597–600
4. Cappiello, C., Matera, M., Picozzi, M.: End-user development of mobile mashups. In Marcus, A., ed.: Design, User Experience, and Usability. Web, Mobile, and Product Design. Volume 8015 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 641–650
5. Mansanet, I., Torres, V., Valderas, P., Pelechano, V.: A Mobile End-User Tool for Service Compositions. In: Actas de las X Jornadas de Ciencia e Ingenieria de Servicios (JCIS). (2014) 25–35
6. Repenning, A., Ioannidou, A.: What makes end-user development tick? 13 design guidelines. End User Development **9** (2006) 1–41
7. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37** (2005) 316–344
8. Ko, A.J., Myers, B.A., Aung, H.H.: Six learning barriers in end-user programming systems. In: Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing. VLHCC '04, Washington, DC, USA, IEEE Computer Society (2004) 199–206
9. van Welie, M., Traelig;tteberg, H., Trastteberg, H.: Interaction patterns in user interfaces. In: Proc. Seventh Pattern Languages of Programs Conference: PLoP 2000. (2000) 13–16

10. Mick, C.P., Roger, T., Frederick, C.G., Scott: What They See Is What We Get: Response Options for Web Surveys. Social Science Computer Review **22** (2004) 111–127
11. Weber, B., Reichert, M., Rinderle, S.: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. Data and Knowledge Engineering **66** (2008) 438–466
12. Serral, E., Valderas, P., Pelechano, V.: Towards the model driven development of context-aware pervasive systems. Pervasive and Mobile Computing **6** (2010) 254–280