



# Towards the Composition of Services by End-Users

## A Mobile-Based Solution

Pedro Valderas · Victoria Torres · Vicente Pelechano

Received: 28 February 2018 / Accepted: 24 June 2019 / Published online: 20 August 2019  
© Springer Fachmedien Wiesbaden GmbH, ein Teil von Springer Nature 2019

**Abstract** Nowadays, we live surrounded by heterogeneous and distributed services that are available to people anytime and anywhere. Even though these services can be used individually, it is through their synchronized and combined usage that end-users are provided with added value. However, existing solutions to service composition are not targeted at ordinary end-users. In fact, these solutions require technical knowledge to deal with the technological heterogeneity in which they are offered to the market. To this end, the paper presents a tool-supported platform that is aided by: (1) EUCalipTool, an end-user mobile tool that implements a Domain Specific Visual Language, which has been specifically designed to compose services on mobile devices; (2) a Faceted Service Registry, which plays the role of gateway between service implementations and end-users, hiding technological issues from the latter when including services in a composition; and (3) a Generation Module, which transforms end-user descriptions into BPMN specification that are interpreted by an execution infrastructure developed for that purpose.

**Keywords** Service · Composition · End-user development · BPMN

## 1 Introduction

Technologies and applications are evolving to create new ecosystems of heterogeneous and distributed services that are available to people anytime and anywhere. Nowadays, our environment abounds with services that support our lifestyles: services that track our activity through smart-phones, that enable efficient use of home heating and lighting, that allow us to interact with social networks, that provide us with the weather forecast or traffic reports in real time, and so on.

Although these services can be used individually, it is their combined usage that has the potential to create new value-added services for end-users. In addition, in a world where end-users play an ever more important role in the development of content, it makes sense to consider the possibility of end-users creating new services through the combination of pre-existing ones. By upgrading end-users to *prosumers* (producer + consumer) and involving them in the process of service creation, both service consumers and service providers can benefit from a cheaper, faster, and better service provisioning (Yu et al. 2012).

However, services are implemented by using heterogeneous technologies such as SOAP or REST, which are difficult for non-technical end-users to understand and use, preventing them from composing services based on their preferences and/or needs. For instance, if an end-user wants to program her/his home air conditioning based on the weather forecast and then publish the air conditioning temperature on a social network such as Facebook, she/he would have to deal separately with the three services

---

Accepted after three revisions by Matthias Jarke.

---

**Electronic supplementary material** The online version of this article (<https://doi.org/10.1007/s12599-019-00617-z>) contains supplementary material, which is available to authorized users.

---

P. Valderas (✉) · V. Torres · V. Pelechano  
Pros Research Center, Universitat Politècnica de València, Camí de Vera/N, 46022 Valencia, Spain  
e-mail: pvalderas@pros.upv.es

V. Torres  
e-mail: vtorres@pros.upv.es

V. Pelechano  
e-mail: pele@pros.upv.es

involved in the example, handling maybe different URLs, protocols, as well as data formats, and then integrate them. In order to automate a scenario like this, existing composition environments (e.g., Intalio, Activiti, Signavio or Bonita BPM) and service composition languages or notations (e.g., Petri nets, EPC, YAWL, BPMN or UML Activity Diagrams) can be used. However, they are not yet targeted at ordinary users, since their usage requires a programming or modelling background. The complexity of this problem increases further if we consider that mobile devices have become the universal interface between services and end-users. Furthermore, the need for service compositions very often arises spontaneously, in a moment of on-the-go inspiration, outside the office environment, with no access to desktops computers or laptops. However, existing mobile solutions mainly support the composition of services through condition-action rules, avoiding the creation of compositions with complex logics. In addition, the list of services that are available to do so is usually hard-coded in the mobile app, making it difficult to evolve. A more detailed motivating example can be found in Section A1 of the Appendix (available online via <http://link.springer.com>).

### 1.1 Research Questions

Considering the motivation presented above, we think that end-users require tools to define the service compositions they need via mobile devices. Thus, we stated the problem to be solved in this paper through the following four research questions:

*Research Question 1.* How can we support end-users in the creation of service compositions with complex logics through the use of a mobile device?

*Research Question 2.* How can we provide end-users with services from different vendors in such a way that technological matters are hidden?

*Research Question 3.* How can we achieve a slight evolution of the provided list of services in a transparent way for end-users?

*Research Question 4.* How can we obtain executable specifications from the descriptions created by end-users with their mobile devices on-the-go?

### 1.2 Main Contributions

The main contributions of this work aim to answer the four research questions presented above. They are the following:

1. A *Domain Specific Visual Language (DSVL)* that allows end-users to easily create service compositions on mobile environments. This DSVL is supported by EUCalipTool, a mobile authoring application for end-users. This contribution aims to answer RQ1.
2. A *Faceted Service Registry* that plays the role of a gateway between end-users and service implementations. This contribution aims to answer RQ2 and RQ3. On the one hand, it introduces the concept of *service facet* to maintain a separation between the semantic data and the technological data of services, providing end-users with the former when they are composing services and hiding the latter. On the other hand, there is no dependency between service implementation and EUCalipTool, which facilitates the evolution and maintenance of the provided services.
3. A *Generation Module* that allows end-users to generate BPMN executable service compositions from the descriptions they make with their mobile devices. These BPMN compositions can be executed by a BPMN engine immediately after these are obtained from end-user descriptions, thus providing the on-the-go aspect. To achieve this, a supporting execution infrastructure is developed. This contribution aims to answer RQ4.

The research methodology followed to perform this work can be found in Section A2 of the Appendix.

### 1.3 Structure of the Paper

The remainder of the paper is organised as follows: Sect. 2 presents the related work. Section 3 introduces an overview of our solution. Section 4 introduces EUCalipTool and its supported DSVL. Sections 5 and 6 present the Faceted Service Registry and the Generation Module respectively. Section 7 introduces the environment in which the generated BPMN specifications are executed. Section 8 discusses the evaluation of this work. Finally, the conclusions and further work are presented in Sect. 9.

## 2 Related Work

In this section, we analyse various contributions that deal with the composition of services in the context of mobile end-user development. In Section A3 of the Appendix the reader can find an additional discussion of some non-mobile solutions as well as some approaches dealing with the problem of integrating services that are technologically different. Some insights extracted from the analysis of the related work can be also be found in this section of the Appendix.

Mobile end-user development approaches can be classified into two categories: (1) those that automate

behaviour by means of condition action rules, and (2) those that provide a language with a richer expressivity in order to define more complex flows of activities.

## 2.1 Trigger Action Programming

Trigger-action programming allows end-users to configure the behaviour of a system by specifying triggers (e.g., “if there is motion”) and their resultant actions (e.g., “turn on the lights”). Following this approach we find iCAP (Dey et al. 2006), a visual, rule-based system that allows end-users to build, prototype, test, and deploy interactive context-aware applications without writing any code. Other approaches similar to iCAP are Atooma (2015), Tasker (2015) and Locale (2015). Lucci and Paternò (2014) presented a comparison between Atooma, Tasker, and Locale with the objective of analysing the expressiveness and usability of this type of tool. The obtained results illustrate that the most expressive environment (Tasker) was also the most difficult to use, which reinforces the need to provide end-user tools with high expressiveness, but without compromising usability. Context Studio (Häkkinen et al. 2005) allows the creation of applications that activate mobile functions when a defined context-action rule is satisfied.

The works presented above have the limitation of the semantics provided by trigger-action rules (“if-then”), which is not enough to answer the proposed RQ1. RQ2 is satisfactorily answered since these tools provide end-users with metaphors that hide technological issues. However, only the functionality that is locally available on the mobile device can be executed. Thus, RQ3 – relating to the slight evolution of the provided services – cannot be adequately answered. Regarding RQ4, these works answer the question satisfactorily since they provide the end-user with authoring environments that allow the on-the-go execution of the created compositions.

Another solution based on condition-action rules is IFTTT (2015). In this case, the solution is not only focused on composing the inbuilt functions of a mobile device. It provides a complete software platform that connects apps, devices and services from different developers in order to trigger one or more automations. The idea behind this solution is similar to that proposed in this paper: services are registered in a proprietary repository through a website, and they can be connected by the end-user with a mobile app. Thus, any time a service is registered in their platform, end-users can access it immediately in the mobile app. This leads us to consider RQ3 as satisfactorily answered. RQ4 is also supported since all the recipes defined by end-users are executable on-the-go, without requiring the installation of any generated app. In addition, IFTTT provides abstract descriptions of services that hide technology issues, which

properly support RQ2. However, it suffers from the main drawback of solutions based on “if-then” rules: it does not have enough expressivity to properly answer RQ1.

## 2.2 Complex Flows of Activities

There are some other works that provide the possibility to create service compositions with more complex logics than condition-action rules.

MicroApp (Cuccurullo et al. 2011) provides end-users with a graphic environment where they can create applications by including the actions that are offered by a mobile phone (e.g., take a picture, send an email) in vertical columns, allowing the specification of both a sequence and a parallel execution.

Microservices (Danado et al. 2010) is an authoring tool to create mobile applications. There are two different views, the beginner’s view, which is targeted at users with no programming skills, and the advanced view, which is targeted at more advanced users. In the beginner’s view, users are assisted during the application creation, whereas in the advanced view, users have more freedom and control when defining the behaviour of the application.

These two approaches provide a composition language for end-users with a high expressiveness and provide high level descriptions of services, which answer RQ1 and RQ2 satisfactorily. However, the internal implementation of the tools and the adopted architectural design make it difficult to properly satisfy RQ3, since services are coupled to end-users’ mobile devices, complicating their evolution and updating. Regarding RQ4, both works provide an architecture where end-user descriptions are interpreted by a proprietary engine, without requiring the compilation and installation of any code.

Another work that must be referenced is Workflow.is (2018). This is an application that has two versions: one for the web and another for Apple devices, including iPhones. It allows end-users to create complex compositions through the definition of a flow of actions in a vertical layout. These flows can include repetitions and conditions. Thus, we can consider that RQ1 is answered satisfactorily. RQ2 is not properly supported since end-users must configure the invocation data (i.e., protocol, host, port, and so on) to execute external services. There is no mechanism to easily evolve the list of provided services, since this tool only focuses on the composition of the actions that are provided by iOS devices (RQ3 is not satisfied). RQ4 is satisfactorily answered since a fully operative execution environment is provided that facilitates the execution of compositions without deployment tasks.

Puzzle (Danado and Paternò 2014) is a framework that allows end-users to create mobile applications directly on a mobile device. It allows combining the functionality

provided by the device itself, smart objects, and web services, as if they were jigsaw pieces. However, only sequences of actions and iterations can be defined. No support is provided for conditional actions or parallel executions. Thus, although this work provides a language with more expressiveness than, for instance, “if–then” rules, it cannot be considered to answer RQ1 satisfactorily. The use of the jigsaw metaphor allows users to compose services without handling technical issues. Thus, RQ2 is properly supported. This work provides an HTTP architecture to include external services in composition time. This architecture also supports the execution of compositions. Thus, RQ3 and RQ4 can be considered satisfactorily answered.

Finally, TouchDevelop (Athreya et al. 2012) is a mobile programming environment targeted at end-user programmers, i.e., users with programming knowledge, to create mobile applications. In this case applications are provided as scripts written in the TouchDevelop language, which is not targeted at ordinary end-users, so RQ1 is not supported adequately. In the same way, technological issues need to be managed to properly call services, which means RQ2 is not supported. The flexibility of using low-level scripts allows including any services easily so RQ3 can be considered properly supported. The requirement of RQ4 can be considered to be met, since the created apps are executed on the mobile phone itself.

### 2.3 Conclusions

The analysis of the related work illustrates that the four research questions proposed in this work are not answered satisfactorily by any of the approaches considered here. A final comparison is shown in Table 1. We can see from Table 1 that almost all the analysed approaches support the execution of the service composition (RQ4). Note, however, that few approaches support the execution of compositions with complex logics (RQ1). Most of them provide views of services that hide technological issues (RQ2), but the provided services are limited in some

approaches to only inbuilt functions of mobile devices. External services are not supported. Regarding the slight evolution of the provided list of services (RQ3), we can see that few approaches give a proper solution.

### 3 Conceptual Design

The main goal of this work is to provide end-users with mobile tools to compose services that are provided by vendors that may use different implementation technologies. To do so, we present a solution whose main pillars are EUCalipTool, a Faceted Service Registry, and a Generation Module. The rationale behind the decisions that we have taken to develop our solution can be found in Section A4 of the Appendix.

The three proposed contributions are organized into a three-layer architecture (Fig. 1):

The *Service Layer* encompasses the services developed by professionals. Services are implemented by using the technology that each professional has considered to be appropriate (e.g., SOAP or REST).

The *Application Layer* provides end-users with EUCalipTool, an end-user authoring tool for mobile devices. EUCalipTool interacts with a Faceted Service Registry to access high-level descriptions of services in order to provide them for end-users. The end-users use the DSVL that supports EUCalipTool to compose these service descriptions.

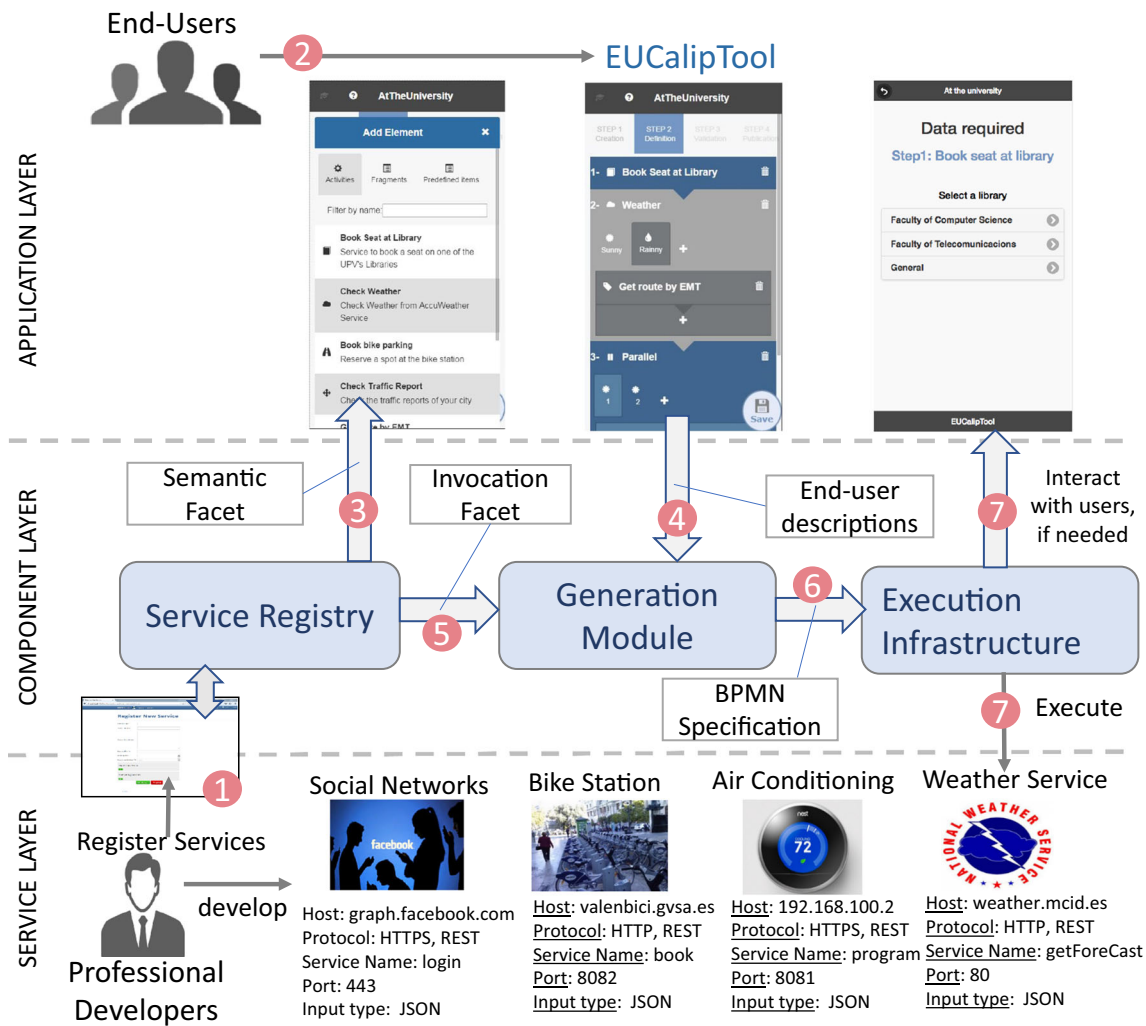
The *Component Layer* hosts the software artefacts required to connect the two layers presented above, allowing end-users to create and execute service compositions. On the one hand, the *Faceted Service Registry* plays the role of a gateway between service implementations and end-users, hiding service technological issues from the latter. It maintains two facets of services: an invocation facet, which is used to invoke services; and a semantic facet, which is used by EUCalipTool. To make a service available to end-users, developers must register it with the registry by defining both types of data (invocation and semantic). Details about how developers can register services in the repository can be found in Section A8.1 of the Appendix. End-users only need to interact with the high-level representation provided by the semantic facet.

On the other hand, once end-users have finished a composition, EUCalipTool submits it to the *Generation Module*, which connects to the Faceted Service Registry in order to obtain the invocation facet of each service of the composition. Then, this data is used together with the end-user description in order to generate an executable BPMN specification, which is interpreted by an *Execution Infrastructure* developed for that purpose. This infrastructure is embowed with a BPMN engine such as Activiti.

**Table 1** Comparison of mobile end-user development approaches

Approaches	RQ1	RQ2	RQ3	RQ4
iCAP, Atooma, Tasker and, Locale	No	Yes <sup>a</sup>	No	Yes
Ifittt	No	Yes	Yes	Yes
MicroApp, Microservices	Yes	Yes <sup>a</sup>	No	Yes
Workflow.is	Yes	No	No	Yes
Puzzle	No	Yes	Yes	Yes
TouchDevelop	No	No	Yes	Yes

<sup>a</sup>Only inbuilt functions of a mobile device



**Fig. 1** Architecture of the proposed solution

In order to better understand how end-users can create and execute a composition of services, we summarise the steps that each participant of the proposed architecture performs. These steps are illustrated in Fig. 1 and are as follows:

- Professional developers implement services by using the technology they consider opportune and register them in the Faceted Service Registry. They define the invocation and semantic facets of each service.
- End-users create an empty composition with EUCalipTool in order to include the desired services.
- EUCalipTool provides end-users with high-level descriptions of services. These descriptions are provided by the Faceted Service Registry (semantic facet of services).
- End-users complete a composition by using the composition constructors provided by EUCalipTool and send it to the Generation Module.
- The Generation Module accesses the Faceted Service Registry in order to obtain the invocation data of each service included in the composition (invocation facet of services).
- From both the end-user description and the invocation data of services, the Generation Module creates a BPMN executable specification that is sent to the execution infrastructure.
- The Execution Infrastructure is supported by a BPMN engine such as Activiti that executes the service composition. In order to interact with end-users if some data must be requested or shown, the infrastructure interacts with EUCalipTool, which provides end-users with the proper user interface.

#### 4 A DSVL for Creating Service Compositions with EUCalipTool

In this section, we present the main concepts of the DSVL supported by EUCalipTool, which allows end-users to compose services by using the respective high-level description. This DSVL was presented in detail in a previous work (Valderas et al. 2017).

EUCalipTool proposes the creation of a composition by always using the metaphor of “adding an element” to a container. The composition is the main container and end-users can add activities or fragments. A detailed description of its metamodel can be found in Section A5.1 of the Appendix.

*Activities* are high-level representations of the services that are developed by professionals and have been registered in the Service Registry. We use the term “activity” instead of service because it is closer to end-users’ mental models (Engeström et al. 1999).

*Fragments* are based on the resultant structures of applying change patterns (which are abstractions of BPMN constructors, see Webber et al. (2008)). There are fragments of three types: (1) the *Parallel Fragment*, which can contain *branches* of activities that must be performed at the same time; (2) the *Loop Fragment*, which contains a set of activities that must be performed in an iterative way; and (3) the *Conditional Fragment*, which contains conditioned *branches* to perform activities when a condition is satisfied.

Fragments are structures that add specific logics (i.e., loop, parallel, and conditions) to a set of activities. However, end-users do not need to worry about the creation of complex elements that represent such logics. Instead, they are just required to add elements to a specific container (a fragment). We have created an analogy between the activity of adding elements, which is well-known by end-users, and the composition of services. Note that analogies are powerful cognitive mechanisms for constructing new knowledge from knowledge already acquired and understood (Repenning and Ioannidou 2006).

With regard to the visual aspect, in Danado and Paternò (2014) different metaphors were evaluated by end-users in order to identify which ones were most suited to intuitively connect components and compose various arrangements. The jigsaw and workflow metaphors were the two most highly ranked. We have based our work on these two metaphors to create the graphical representation of a service composition.

On the one hand, we use the workflow metaphor to define the elements of a composition since it is easy to use in a mobile device. Graphically, the workflow of the elements of a composition is represented by applying the List layout (Fig. 2), which is widely used in mobile design to facilitate the scrolling of a collection of elements. The order in which elements are displayed (from top to bottom) represents the order in which they will be considered at runtime.

On the other hand, each element of a composition is connected graphically to the next by a small inverted triangle. This aspect was inspired by the jigsaw metaphor (Renger et al. 2008), which defines pieces inserted into others to reinforce the notion of connection or combination of elements. We have used a similar solution to evoke the idea of connecting activities and/or fragments.

Figure 5 shows an example of a composition’s graphical representation. Note that end-users can add elements to a composition or to a fragment by using a button with the “+” symbol, which is located either at the end of the composition or at the end of a fragment’s content. This button is placed in the location where the new element will be added in order to help end-users to create a mental image of the result of the action before performing it. There is also a delete button that makes it possible to remove composition and fragment elements. This is an icon-based button that shows the image of a trashcan, which is broadly accepted to represent the action for removing/deleting. It is displayed to the right of each element.

A component with three tabs (Fig. 3) has been defined to add elements: the first two (Fig. 3a, b) allow activities and fragments to be added. The third represents predefined items (Fig. 3c), which are subsequently explained in detail.

Note that some activities may need some inputs to be executed. The user interface designed to do that is introduced in Section A5.2 of the Appendix.

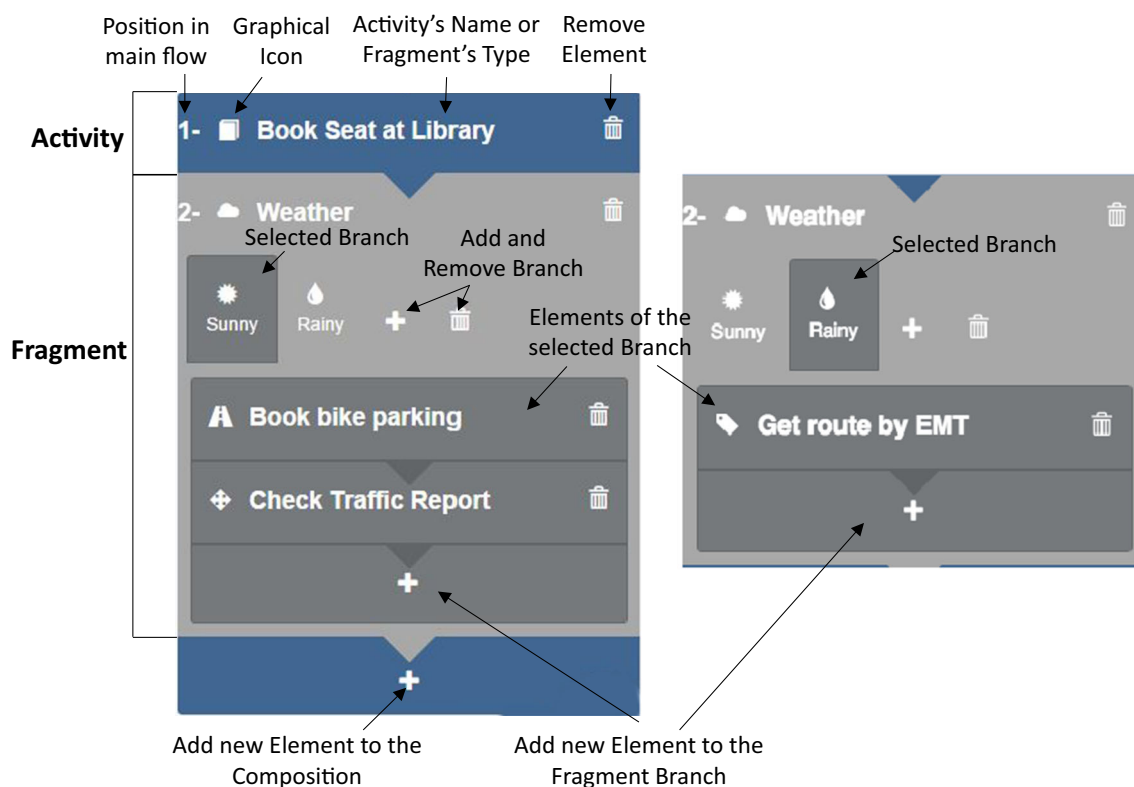
##### 4.1 Predefined Items

EUCalipTool provides end-users with *Predefined Items*, which are conditional fragments with a predefined condition. It allows end-users an easier definition of actions that depends on conditions such as weather, location, time, etc. We have been inspired by end-user guidelines that promote the provision of predefined components (Segal 2005). See Valderas et al. (2017) for more details.

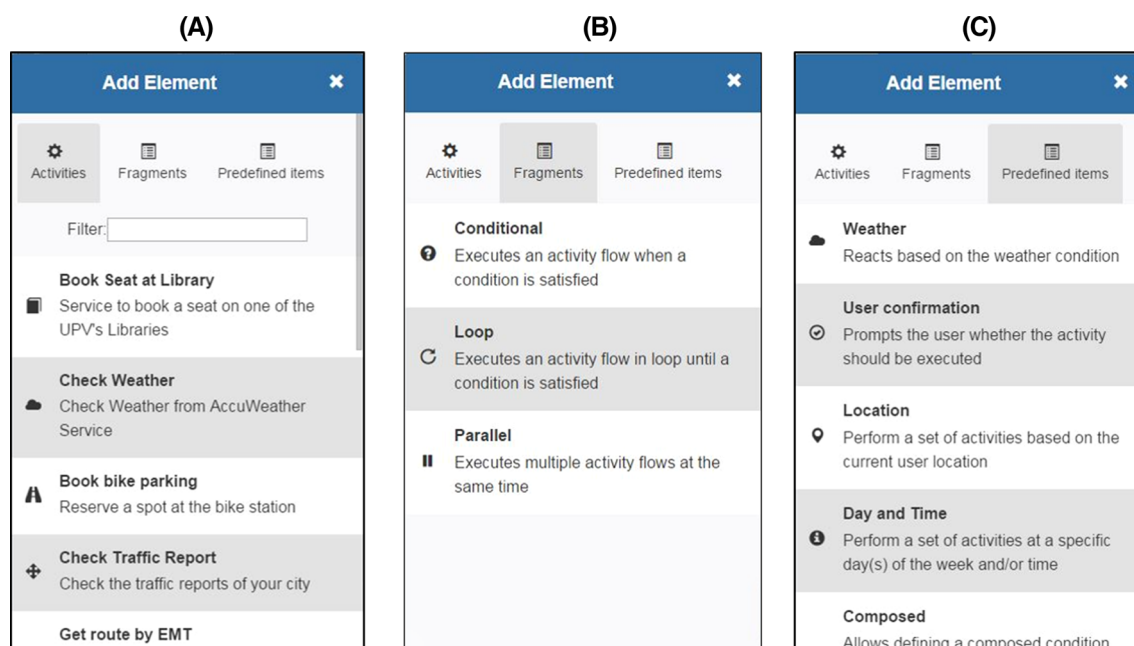
In order to configure the predefined conditions, specific graphical components were designed. They were all defined by taking into account the study presented in Galitz (2002), which recommends selecting data instead of typing it in, to avoid end-user errors. As representative examples, Fig. 4 shows the screens that allow end-users to configure a weather condition (A), a location condition (B) and a day and time condition (C).

##### 4.2 The Tool in Action

This section presents an example in which John, a university student, automates some actions in an exam period (see Section A1 of the Appendix for a detailed description of this scenario).



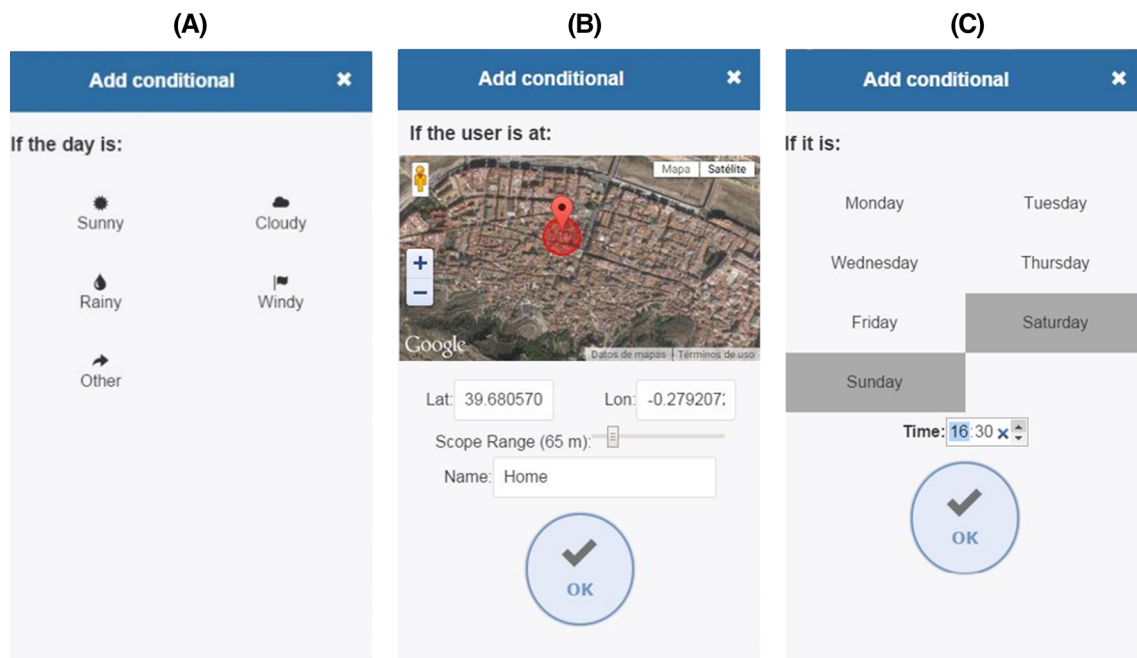
**Fig. 2** Graphical representation of a composition



**Fig. 3** Tabs for adding elements

After creating an empty composition (Fig. 5), John accesses its graphical representation, which is shown in Screen 1. Initially, it is represented by an empty list.

From the empty composition, John clicks the button with the “+” symbol, and accesses Screen 2 where the available services are shown as activities. From the list of activities, John adds the activity “Book Seat at Library” to



**Fig. 4** Configuration of the predefined items

the composition (see result in Screen 3). Next, he clicks the “+” button again and selects a *Weather Predefined Item* (see the list of available predefined items in Screen 4).

The Weather Predefined Item is configured with a branch associated to the ‘Sunny’ state (see Screen 5). Next, John adds the activities “Book Bike Parking” and “Check Traffic Reports” to the newly created branch by following the same steps as shown before (see result in Screen 6). Thus, these activities will be executed if it is a sunny day. Next, he adds another branch associated to the ‘Rainy’ state by clicking the “+” button in Screen 6, and adds the activity “Get route by EMT” to it (see result in Screen 7).

By clicking the “+” button located just below the composition again, John adds a *Parallel Fragment* (see the list of available fragments is in Screen 8). He adds the activity “Publish on Facebook” in one branch of the fragment, and the activity “Publish on Twitter” in the other by following the same steps as before (see the final result in Screen 9). Thus, the activities included in each branch will be executed at the same time, i.e., in parallel.

## 5 A Faceted Service Registry

The proposed service registry acts as a gateway between end-users and service implementations, managing both data types of services, which are represented by two facets: semantic and invocation. On the one hand, EUCalipTool provides end-users with a list of activities that correspond with high-level descriptions of the services that are

available on the Faceted Service Registry (semantic facet). On the other hand, the Faceted Service Registry also maintains the invocation data of services (invocation facet) in order to enable the execution of service compositions in a real environment. This data is not used by EUCalipTool, but is sent to a Generation Module in order to be managed.

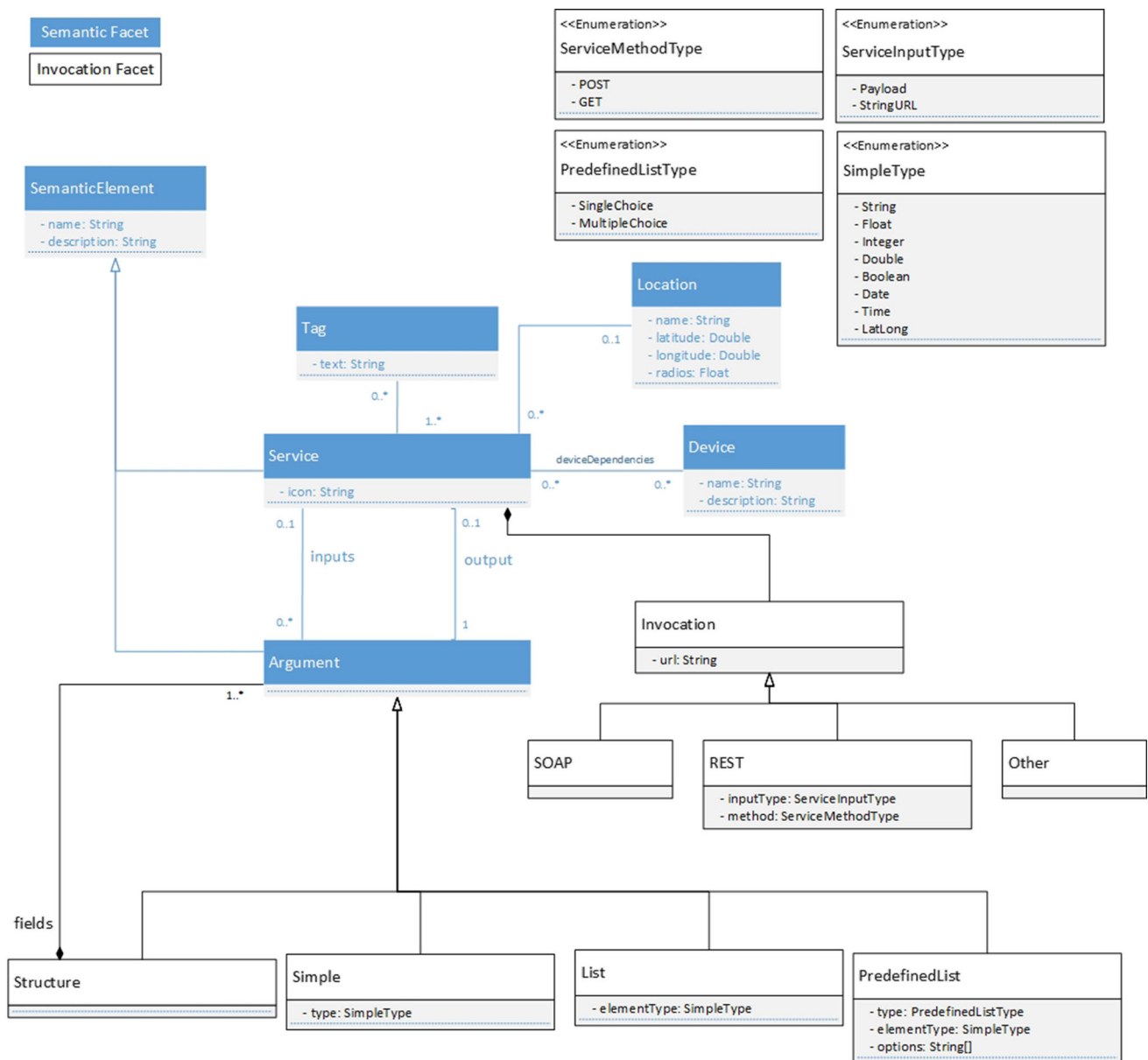
Figure 6 describes the information that is stored in the Faceted Service Registry for each of the facets by means of a UML Class Diagram (Rumbaugh et al. 2004).

### 5.1 Semantic Facet

This facet describes the logics of the service. It is focused on helping end-users to understand the internal behaviour of services when creating a service composition.

In order to define this facet, we collected properties incorporated in existing service profiles (Amir and Zeid 2004; Klusch and Sycara 2001; Ermagan and Krüger 2007; Paolucci et al. 2002), and asked a group of end-users to indicate which of these properties they found understandable. To carry out this task, we arranged some focus group sessions with a total of 15 participants, aged between 20 and 43 years old (five females and eight males). The majority of them used mobile devices daily, but none had a background in programming. In each session, we gave participants the list of properties extracted from the different analysed profiles, and asked them to explain what each property meant.

**Fig. 5** Example of usage of the proposed end-user mobile tool



**Fig. 6** Faceted Service Registry data

Based on the explanations we obtained, those properties that most of the participants could explain properly were included in the semantic facet. They are the following:

- *Name*: name of the service.
- *Purpose*: a high-level description of what constitutes a (typical) successful execution of a service.
- *Description*: a brief, human-readable description of the service, summarising what the service offers or what capabilities are requested.
- *Location*: geographic scope of the service, (e.g., university, home). This property is useful to characterise services such as those that are closely coupled with the physical environment in which they are executed.
- *Inputs*: values that are required to execute a service. Each input is characterised by a *name* and a *textual description*.
- *Output*: value obtained after the execution of a service. It is defined by a *name* and a *textual description*.
- *Type*: classification of the service according to its specific domain (e.g., weather, social networks, teaching, commerce, and so on).

Additionally, we proposed other properties by analysing some case studies developed through studies on context-aware services (Serral et al. 2013), activity and task

modelling (Uden et al. 2008; Valderas et al. 2006), and adaptive business processes (Ayora et al. 2013). After checking with end-users which of them were understandable, we added the following:

- *Semantic tags*: a list of keywords that characterise the internal behaviour of a service.
- *Device dependencies*: a list of physical devices that are needed to execute a service. Note that a service may be intrinsically linked to a specific device (e.g., services that control an air conditioning machine or a smart TV that need to interact with a mobile device).
- *Icon*: a graphical representation of the services.

As can be seen in Fig. 6 (elements depicted with blue background headers), these properties are internally represented as follows: a *service* has a *name*, a *purpose*, a *description*, a *type*, and a graphical *icon* must be introduced. A service may have *input* and *output* arguments, which have a *name*, and a *description*. Services also have a *location*, which is represented by a *name*, *latitude*, *longitude* and *radius*. Additionally, each service is semantically marked with a list of *tags* and *device dependencies* that are used to characterise their internal behaviour.

## 5.2 Invocation Facet

This facet defines the information that is required to execute a service at runtime (depicted with white background headers in Fig. 6).

For each *service*, the registry maintains *invocation* data, which can contain either the *URL* of the service if it is a SOAP service; or the *URL method* to be use in the invocation (POST or GET) and the way of passing the input values (*inputType*, which can indicate the use of the payload of the HTTP connection or a string codification in the URL) if it is a *REST* service. This information is used to invoke the service at runtime.

Furthermore, there are different types of service arguments. In particular, an argument can be:

- *Simple*, which indicates that the argument has a simple type value. Simple types are defined by the *SimpleType* enumeration (String, Float, Integer, Double, Boolean, Date, Time and LatLong).
- a *List* of values, which indicates that the argument may contain a list of simple type values. These values must be defined at runtime.
- a *PredefinedList* of values, which indicates that the argument has a predefined list of *options* to be selected. According to the *PredefinedListType* enumeration, there are two types: Single Choice Lists, in which users must select one—and only one—option; and

Multiple Choice Lists, in which users can select more than one option.

- a *Structure of fields*, which indicates that the argument is composed of several values.

## 6 BPMN Generation Module

The end-user descriptions that are created with EUCalipTool are not executable, therefore we need to use a mechanism that allows us to ‘compile’ them into an executable version. To achieve this, we have implemented a module that generates BPMN specifications from end-user descriptions.





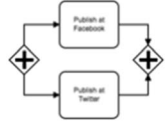



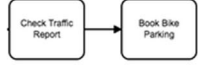

When end-users complete a composition, a JSON description is sent by EUCalipTool to the BPMN generator module. Then, this module parses the received description with a JSON parser, and a set of transformation rules (TR) are applied to generate a BPMN specification by using an XML parser. The main BPMN elements that are generated through these rules are tasks, gateways, and sequence flows. These rules access the Faceted Service Registry to obtain the required service invocation data, which is included by means of XML elements such as extensionElements. The rules are all summarized in Table 2. Section A6 of the Appendix shows some representative examples.

## 7 The Execution Environment

In this section, we present an HTTP-based execution infrastructure to allow for the generated BPMN specification to be executed on-the-go from the mobile devices of end-users. Once a BPMN specification is generated (Fig. 7), it is stored in a repository (step 0), and its execution is performed as follows:

1. In addition to the authoring environment, EUCalipTool provides end-users with an *Execution Module* that allows them to execute any service composition they have made. Once a composition is selected for execution, EUCalipTool interacts with a *BPMN Launcher* of the execution infrastructure in order to request the execution of the selected composition.
2. The execution infrastructure is composed of the Activiti BPMN engine together with two additional modules. One of these modules is the *BPMN launcher*, which is in charge of loading BPMN specifications from the BPMN repository and passing them on to the Activiti engine for them to be executed.
3. Activiti is in charge of executing each BPMN specification it receives by respecting the blocks that are

**Table 2** Summary of the rest of the transformation rules

DSVL	BPMN
<b>Activity</b> 	<b>- Service Task</b> 
<b>Conditional fragment</b> <b>Weather predefined item</b> <b>Location predefined Item</b> <b>Day and Time predefined Item</b> <b>User Confirmation Predefined Item</b> <b>Composed Predefined Item</b>	<b>- Service Task(s) to check condition(s)</b> <b>- Fork Exclusive gateway</b> <b>- Sequences of service tasks</b> <b>- Join Exclusive gateway</b> 
<b>Parallel fragment</b> 	<b>- Fork Parallel gateway</b> <b>- Sequences of service tasks</b> <b>- Join Parallel gateway</b> 
<b>Loop fragment</b> 	<b>- Fork Exclusive gateway</b> <b>- Service task sequence</b> <b>- Service task to check condition</b> <b>- Join Exclusive gateway</b> 
<b>Sequence of activities or fragments</b> 	<b>- Service task or fragment elements</b> <b>- Sequence</b> <b>- Service Task or fragment elements</b> 
<b>Activity's inputs</b> 	<b>- ExtensionElement</b> <pre> &lt;extensionElements&gt;   &lt;activiti:field name="inputs"&gt;     &lt;activiti:string       [id=1;name=psf     &lt;/activiti:string&gt;   &lt;/activiti:field&gt; &lt;/extensionElements&gt; </pre>

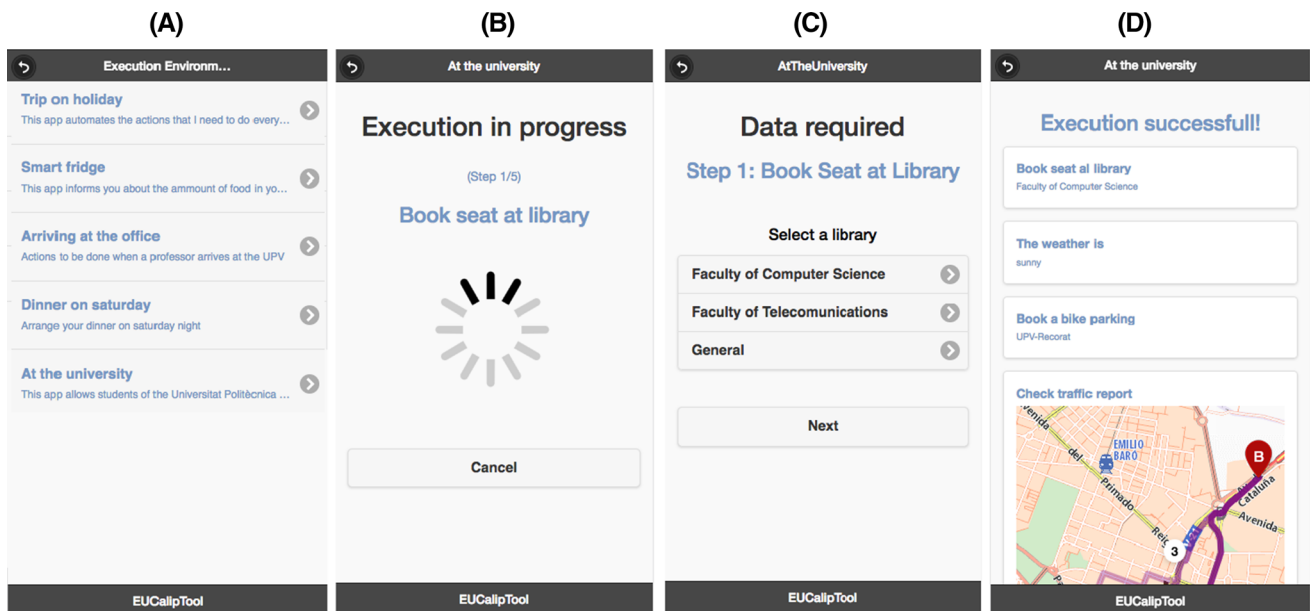
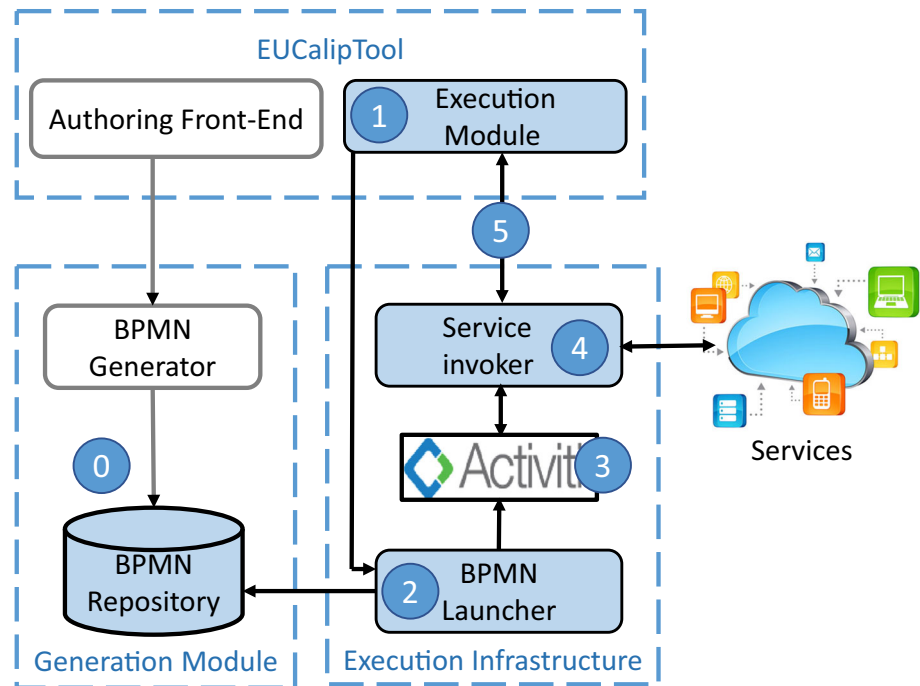
defined (sequences, conditions, parallels, loops). Any Service Task that must be executed Activiti sends to the second module of the infrastructure engine: the *Service Invoker*.

- The *Service Invoker* is in charge of executing the service that is represented by a service task. To do so, it uses the invocation data (i.e., host, URL, arguments, and so on) that is included at the time of generation (see Sect. 6). The Service Invoker is complemented with adapters that focus on managing the service execution of a specific technology. Currently, we support REST and SOAP services, as has already been explained when introducing the invocation facet of the Service Registry. Other technologies require the implementation of the proper adapters by service

developers (this is explained in Section A8.2 of the Appendix).

- Once a service composition is launched, the *Execution Module* provides the end-user with a user interface that is used to either show the results of a composition or to request end-users to input some data that is required to execute a service.

Next, we present some snapshots of the screens that end-users interact with when executing service compositions. Figure 8a shows the list of the compositions that are available for execution. Figure 8b shows an intermediate screen that informs about the execution process. Figure 8c shows a screen that requests end-users to select a library, which consists of data that must be introduced at runtime. Finally, Fig. 7d shows the results of the execution.

**Fig. 7** Execution environment architecture**Fig. 8** Snapshots of the execution of a service

Section A7 of the Appendix presents the strategy used to create these screens at runtime.

## 8 Evaluation

To evaluate our approach, we conducted several experiments. In this section, we present an experiment to evaluate the satisfaction of end-users with the complete process of

defining a composition, generating the executable specification, and executing it. We also used this experiment to understand how end-users envision the use of the platform in a real scenario. In Section A10 of the Appendix the reader can find the additional evaluations of our work.

*Participants* To perform this experiment, we recruited 13 end-users (8 male/5 female) that were familiar with mobile devices and did not have knowledge of using programming languages. We contacted them through e-mail or

personal invitation. As to their jobs, 10 participants had occupations not related to the academic environment or computer science; and 3 of them belonged to the administrative staff of our research centre. Regarding their experience with mobile devices, all participants had moderate to considerable experience using them to browse the web, read e-mails or use social networks; 30.7% of them also indicated that they play games with mobile devices. With respect to desktop environments, 69.2% stated that they daily interact with computers or laptops to read emails, use word processors or spreadsheets, or to browse the web. Finally, regarding other environments, 46.15% of the participants indicated that they used or were interested in the use of home assistants such as Alexa or Google Home.

*Design* First, we arranged a working session to train all participants in the use of EUCalipTool. Note that the usability of the authoring tool was evaluated in another experiment (see Section A10.1 of the Appendix). Next, we conducted an evaluation that consisted of accomplishing a specific task and a post-test questionnaire. Our main idea was to propose to participants to perform an open task and allow them to create the composition they preferred. However, by using an open task there is the possibility of obtaining unprecise answers, and there is also the risk that the task remains unperformed (Danado and Paternò 2014). Thus, the steps that we followed to perform the experiment were the following:

1. We asked participants to describe a scenario where they envisioned that it would be appropriate to use the platform. When participants gave a precise answer, we checked that EUCalipTool provided all the services required to support such scenario. If this was not the case, we included the missing services into the Registry. In case participants could not provide a precise answer, a list of scenarios was presented in order to allow them to select the one that better fitted their interest. The proposed scenarios can be found in Section A9 of the Appendix.
2. Once participants indicated the scenario they preferred, and we had checked that the proper services were registered with the Faceted Service Registry (or added them if it was necessary), we requested them to create a service composition with EUCalipTool and execute it.
3. Afterwards, we measured the satisfaction level of participants. To do so, we used the Microsoft Product Reaction Cards (Benedek and Miner 2002). This method consists on providing participants with a list of 118 words, and asking them to choose the words that they would use to describe a product (we limited the number of words to be selected to 5 to keep the

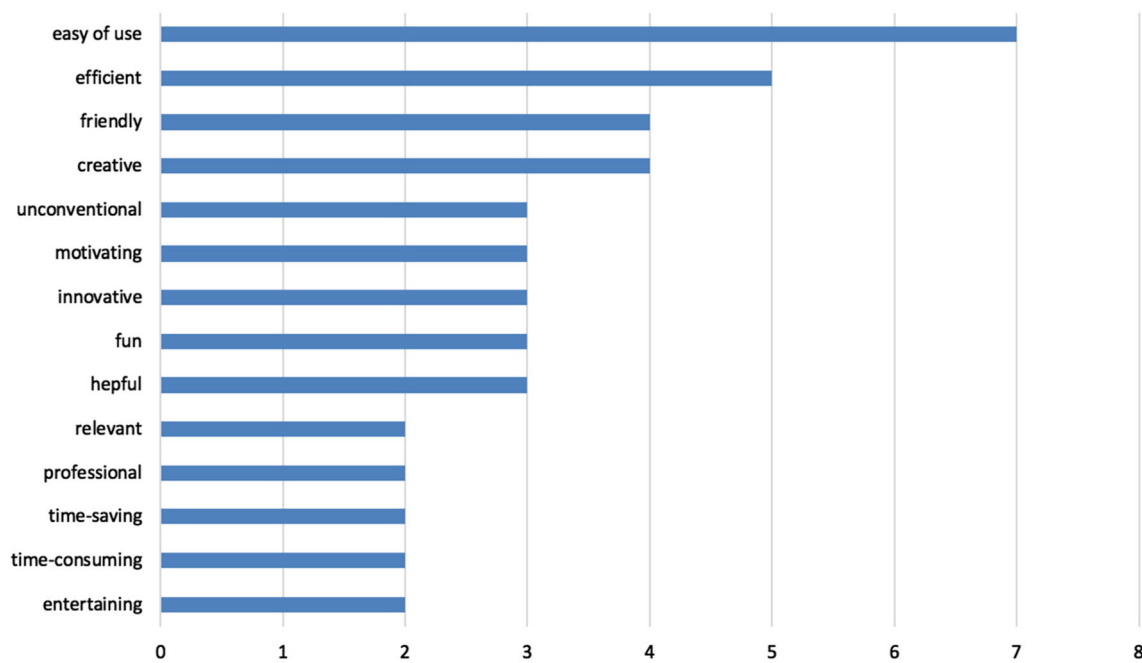
exercise short). For each selected word participants were asked why they had chosen that particular word. The list included positive words like ‘Useful’ and ‘Engaging’, but also negative words, such as ‘Frustrating’ and ‘Ineffective’. We also asked participants to complete a questionnaire in order to know their perceptions about the usability of the execution interface. We used an adapted SUS questionnaire (Broke 1996) that included a total of 10 questions following a five-point Likert scale response ranging from 1 (Strongly Disagree) to 5 (Strongly Agree).<sup>1</sup>

4. During task performance, we were present to clarify any doubts participants could have, and took notes on the way they completed the tasks. After finishing the experiment, we had informal interviews with participants to comment on the notes made, to ask them for the reasons for the word selection according to the MRC method, and to discuss with them the comments they provided in the questionnaire.

*Results* Regarding the application scenarios, only 3 participants were able to give a precise answer to the proposed open question. The scenarios they described were based on smart home environments. The rest of participants selected one of the proposed scenarios as follows: 3 of them selected a scenario based on a smart home; 3 of them selected a scenario based on smart cities; 2 of them selected a scenario based on integration of mobile devices with social networks; and 2 participants selected a scenario based on services which support sport activities. Although the proposed scenarios may be biased by our experience, they presented heterogeneous application situations and give a preliminary idea of where end-users would foresee EUCalipTool to be used. In particular, most of the participants felt comfortable using the platform in Smart Home or Smart Cities environments.

As to the end-user satisfaction level with the whole platform, which was evaluated through the use of Microsoft Reaction Cards, Fig. 9 shows a graph with occurrences of the words that received more than 2 mentionings. As we can see, “Easy to use” and “Efficient” were the participant’s most selected keywords. Both keywords show that subjects were pleased with the functions provided by the platform and the way of using them. It is worth noting that both “time-saving” and “time-consuming” were selected, which can be a little contradictory. After asking the participants who selected these keywords we understood that: those who selected “time-saving” were considering that the service composition they created can help them to save time (they were thinking of the final product); those who selected “time-consuming” were considering that it

<sup>1</sup> The questionnaire can be found at <https://goo.gl/forms/ihvdX5BxEIwC5IGi2>.



**Fig. 9** Satisfaction evaluation with Microsoft react cards

requires some time to create and execute a composition (they were thinking of the composition and execution process). In any case, the fact that some participants considered that the platform was “time-consuming” reinforced our understanding that it is useful to provide mechanisms such as the predefined items in order to facilitate the use of the platform. If end-users are provided with predefined items that just need to be configured they may need less time to create service composition, which may improve the perception of “time-consuming”.

Finally, the results obtained in the usability evaluation of the execution environment are quite satisfactory (Fig. 10). The answers given by participants allowed us to conclude that 76.9% of the participants (10 out of 13) considered the execution interface understandable; 69.2% (9 out of 13) found that the screens for introducing data

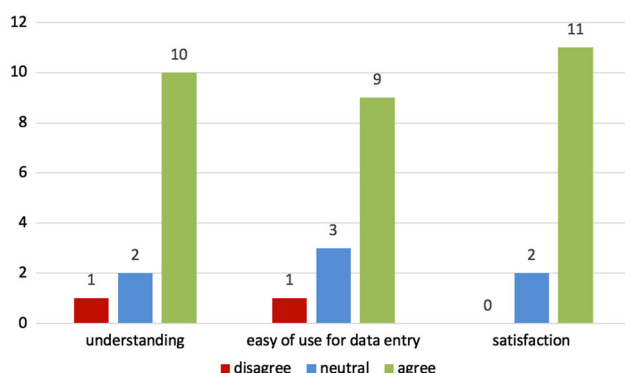
were easy to use; and finally, 84.6% (11 out of 13) were satisfied with the overall interaction design. The main problem they detected was related to the way EUCalipTool alerted them to enter data. Currently, only one screen to request data entry is displayed. There is no other mechanism implemented to alert end-users (for example, sound alert, vibration, etc.). Some end-users found this aspect a little uncomfortable and proposed some kind of notification that would relieve them from having to be aware of the interaction screens during the whole execution process.

## 9 Conclusions and Further Work

In this paper, we have presented a tool-supported platform to help end-users in the composition of technologically heterogeneous services by using their mobile devices. The development and validation of this work has allowed us to learn some lessons that are summarised in Section A11 of the Appendix.

The proposed platform is based on three key pillars:

- EUCalipTool provides end-users with an intuitive mobile environment that allows end-users to create complex compositions by means of a DSL specifically defined for mobile devices.
- A Faceted Service Registry plays the role of a gateway between end-users and service implementation. This aspect allows us to achieve the goal of keeping end-users unaware of any technological issue related with services. In addition, descriptions of available



**Fig. 10** SUS Scores obtained for the execution interface

services are totally decoupled from the end-user tool, facilitating its evolution and maintenance.

- A Generation Module transforms end-user descriptions into specifications that can be executed in a real environment. In addition, a specific Execution Infrastructure has been developed in order to execute these BPMN specifications on-the-go.

However, this is an open research work. For instance, we need to manage the security aspects and service authorization. Many services require users to be registered so that they can be executed by means of the combination of a user and password, or of some key. We plan to extend the EUCalipTool platform by an *identification data wallet* in which users can store their data the first time they introduce it, allowing for its use in further executions.

Current trends turn the social aspect into a key pillar of software solutions. Thus, we are working on extending our work in order to provide EUCalipTool with an added social value that facilitates the development of services by and for “the crowd”.

Finally, we also plan to enrich EUCalipTool with context-aware features that can benefit from the rich ecosystem of internet connected devices. We intend to apply the People as a Service (PeaaS) paradigm (Guillen et al. 2014) that proposes using the capabilities of modern —mobile devices to identify the sociological profiles of their owners.

## References

- Amir R, Zeid A (2004) A UML profile for service-oriented architectures. In: Companion to the 19th annual ACM SIGPLAN conference on object-oriented programming systems, languages, and applications, Vancouver. ACM, New York, pp 192–193
- Athreya B, Bahmani F, Diede A, Scaffidi C (2012) End-user programmers on the loose: a study of programming on the phone for the phone. In: IEEE symposium on visual languages and human-centric computing, Innsbruck. IEEE, pp 75–82
- Atooma (2015) Atooma, a touch of magic. <https://www.atooma.com/>. Accessed 1 Oct 2018
- Ayora C, Torres V, Weber B, Reichert M, Pelechano V (2013) Enhancing modeling and change support for process families through change patterns. In: Nurcan S et al (eds) Enterprise, Business-Process and Information Systems Modeling. BPMDS 2013, EMMSAD 2013, vol 147. Lecture Notes in Business Information Processing. Berlin, Heidelberg, pp 246–260
- Benedek J, Miner T (2002) Measuring desirability: new methods for evaluating desirability in a usability lab setting. In: Proceedings from the Usability’s Professionals Association (UPA)
- Broke J (1996) SUS. A “quick and dirty” usability scale. In: Jordan P et al (eds) Usability evaluation in industry. Taylor & Francis, London, pp 189–194
- Cuccurullo S, Francese R, Risi M, Tortora G (2011) MicroApps development on mobile phones. In: Costabile MF, Dittrich Y, Fischer G, Piccinno A (eds) End-User Development. IS-EUD 2011, vol 6654. Lecture Notes in Computer Science. Berlin, Heidelberg, pp 289–294
- Danado J, Paternò F (2014) Puzzle: a mobile application development environment using a jigsaw metaphor. J Vis Lang Comput 25(4):297–315
- Danado J, Davies M, Ricca P, Fensel A (2010) An authoring tool for user generated mobile services. In: Berre AJ, Gómez-Pérez A, Tutschku K, Fensel D (eds) Future internet—FIS 2010. FIS 2010, vol 6369. Lecture Notes in Computer Science. Berlin, Heidelberg, pp 118–127
- Dey AK, Sohn T, Streng S, Kodama J (2006) iCAP: interactive prototyping of context-aware applications. In: Fishkin KP, Schiele B, Nixon P, Quigley A (eds) Pervasive Computing. Pervasive 2006, vol 3968. Lecture Notes in Computer Science. Berlin, Heidelberg, pp 254–271
- Engeström Y, Miettinen R, Punamäki RL (1999) Perspectives on activity theory. Cambridge University Press, Cambridge
- Ermagan V, Krüger IH (2007) A UML2 profile for service modeling. In: Engels G, Opdyke B, Schmidt DC, Weil F (eds) Model Driven Engineering Languages and Systems. MODELS 2007, vol 4735. Lecture Notes in Computer Science. Berlin, Heidelberg, pp 360–374
- Galitz WO (2002) The essential guide to user interface design: an introduction to GUI. Design principles and techniques. Wiley, New York
- Guillen J, Miranda J, Berrocal J, Garcia-Alonso J, Murillo JM, Canal C (2014) People as a service: a mobile-centric model for providing collective sociological profiles. IEEE Softw 31(2):48–53
- Häkkinä J, Korpipää P, Ronkainen S, Tuomela U (2005) Interaction and end-user programming with a context-aware mobile application. In: Costabile MF, Paternò F (eds) Human-Computer Interaction—INTERACT 2005, Lecture Notes in Computer Science, vol 3585. Berlin, Heidelberg, pp 927–937
- IFTTT (2015) IFTTT, if this then that. <https://IFTTT.com/>. Accessed 1 Oct 2018
- Klusch M, Sycara K (2001) Brokering and matchmaking for coordination of agent societies: a survey. In: Omicini A, Zambonelli F, Klusch M, Tolksdorf R (eds) Coordination of Internet Agents. Springer, Berlin, Heidelberg, pp 197–224
- Locale (2015). <http://www.twofortyfouram.com>. Accessed 1 Oct 2018
- Lucci G, Paternò F (2014) Understanding end-user development of context-dependent applications in smartphones. In: Sauer S, Bogdan C, Forbrig P, Bernhaupt R, Winckler M (eds) Human-Centered Software Engineering. HCSE 2014, Lecture Notes in Computer Science, vol 8742. Berlin, Heidelberg, pp 182–198
- Paolucci M, Kawamura T, Payne TR, Sycara K (2002) Semantic matching of web services capabilities. In: Horrocks I, Hendler J (eds) The Semantic Web—ISWC 2002, Lecture Notes in Computer Science, vol 2342. Berlin, Heidelberg, pp 333–347
- Renger M, Kolfschoten GL, de Vreede GJ (2008) Challenges in collaborative modeling: a literature review. In: Advances in enterprise engineering I, vol 10, Montpellier, pp 61–77
- Repenning A, Ioannidou A (2006) What makes end-user development tick? 13 design guidelines. In: End user development. Human-computer interaction series, vol 9. Springer, Berlin, pp 51–85
- Rumbaugh J, Jacobson I, Booch G (2004) The unified modeling language reference manual. Pearson, London
- Segal J (2005) Two principles of end-user software engineering research. ACM SIGSOFT Softw Eng Notes 30(4):1–5
- Serral E, Valderas P, Pelechano V (2013) Context-adaptive coordination of pervasive services by interpreting models during runtime. Comput J 56(1):87–114
- Tasker (2015) Tasker, total automation for Android. <http://tasker.dinglish.net/>. Accessed 1 Oct 2018
- Uden L, Valderas P, Pastor O (2008) An activity-theory-based model to analyse web application requirements. Inf Res 13(2):1

- Valderas P, Pelechano V, Pastor O (2006) A transformational approach to produce web application prototypes from a web requirements model. *Int J Web Eng Technol* 3(1):4–42
- Valderas P, Torres V, Mansanet I, Pelechano V (2017) A mobile-based solution for supporting end-users in the composition of services. *Multimed Tools Appl* 76(15):16315–16345
- Workflow.is (2018) Workflow. Spend less taps, get more done. <https://workflow.is/>. Accessed 1 Oct 2018
- Yu J, Sheng QZ, Han J, Wu Y, Liu C (2012) A semantically enhanced service repository for user-centric service discovery and management. *Data Knowl Eng* 72:202–218