Valderas Aranda, PJ.; Torres Bosch, MV.; Mansanet Benavent, I.; Pelechano Ferragud, V. (2016). A mobile-based solution for supporting end-users in the composition of services. Multimedia Tools and Applications. 1-31. doi:10.1007/s11042-016-3910-4.

A mobile-based solution for supporting end-users in the composition of services

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

Universitat Politècnica de València, Spain

Pros Research Center

{pvalderas,vtorres,imansanet,pele}@pros.upv.es

Abstract. Currently, technologies and applications evolve to create eco-systems made up of a myriad of heterogeneous and distributed services that are accessible anytime and anywhere. Even though these services can be used individually, it is their coordinated and combined usage what provide an added value to end-users. In addition, user's wide adoption of mobile devices for daily activities have fostered a shift in the role played by end-users towards Internet data and services. However, existing solutions to service composition are not targeted to ordinary end-users. More easy-to-use tools have to be offered to end-users to make sure that they are successfully accepted and used by them. To this end, the work presented in this paper supports end-users in the creation of service compositions by using mobile devices. We present a Domain Specific Visual Language (DSVL) for end-users that allows them to create service compositions. A tool specifically designed for mobile devices supports this DSVL.

1. Introduction

In an increasingly dynamic, intelligent, and decentralized open world, technologies and applications evolve to create eco-systems made up of a myriad of heterogeneous and distributed services that are accessible anytime and anywhere. Some examples are: 1) those services provided by governments, public organizations, or scientific communities via Open Data[1] (e.g., weather forecast, traffic status, air and water quality, etc.); 2) those provided by the emerging Internet of Things (IoT) [Gubbi et al. 2013], which allow users to interact with smart objects (e.g., electrical appliances, wearable elements, or any other kind of "connected" object); or 3) those that are intensively used by end-users for interacting with each other, for instance, with social networks (e.g. Facebook or Twitter).

Even though these services can be used individually, it is their composed usage what has the potential to create new value-added services for end-users. Considering that end-users play a more and more important role in the development of content, it makes sense to think on the possibility of enabling them to create their required compositions. By upgrading end-users to *prosumers* (producer+consumer) and involving them in the process of

[1] Open Data is an initiative with the aim of making free and available certain data to anyone with the purpose of using, modifying, and sharing it.

service creation, both service consumers and service providers can benefit from a cheaper, faster, and better service provisioning [Yu et al. 2012].

However, 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) are not targeted to ordinary users, since their usage requires programming or modelling background. We need tools that help and assist end-users to define the service compositions they need. Note also that professional developers lack many times the domain knowledge that end-users are not always able to articulate [Liberman et al. 2006]. Therefore, in a world with millions of services with different semantics and purpose, the use of tools for allowing end-users to express their needs is even more important [Bohem et al. 2000].

The complexity of this problem is further increased if we consider that end-users often need the composition of services on the move. Currently, mobile devices have become into the universal interface between services and end-users, and the need for their compositions usually rise in a spontaneous and inspired on-the-go way, outside office environments, with no access to desktops computers or laptops. In many cases, mobile devices are the only platform that end-users are working with in order to perform their daily tasks. Thus, it makes sense to think that mobile devices are a good platform to allow end-users not only to consume their preferred services but also to create new ones by composing them. In this sense, one of the main challenges to be faced is the design of a service composition environment that considers intrinsic characteristics of mobile devices such as restricted screen sizes, usage of touch-based interactions, or different contexts of usage.

In this work, we face the problem of supporting end-users in the composition of services with mobile devices. We present a Domain Specific Visual Language (DSVL) specifically designed to be used in mobile environments. This DSVL is supported by *EUCalipTool*, a mobile application targeted to end-users without programming skills that allows them to easily compose services. With this work, we aim to improve the research on mobile end-user development to encourage end-users to go beyond consuming services and become producers of services.

This work is developed within the context of SITAC (Social Internet of Things, Apps by and for the Crowd), an ITEA 2 project aimed to improve the interaction among humans and services. A preliminary version of the DSVL was presented in [Mansanet et al. 2014]; the complete architecture can be found in [Mansanet et al. 2015].

The rest of the paper is organized as follows: Section 2 introduces the related work. Section 3 presents a motivation example. Section 4 introduces some rationale of the proposed solution. Sections 5 and 6 present the

DSVL created for end-users and EUCalipTool, its supporting authoring mobile tool. Section 7 introduce an overview of the software architecture that supports EUCalipTool. Section 8 discusses the validation of this work. Finally, conclusions and further work are presented in Section 9.

2 Related Work

In this paper, we present a research work to support end-users in the composition of services from mobile devices. The main research field where we can find works that also face this challenge is the End-User Development (EUD) field, more specifically the mobile EUD (m-EUD) branch. Next, we discuss the most important works developed in this field.

iCAP [Dey et al. 2006], is a visual, rule-based system that allows end-users building, prototyping, testing, and deploying interactive context-aware applications without writing any code. To do so, end-users specify graphically the behaviour of the application by defining objects (i.e., people and artefacts) and the rules that govern them. These rules are categorized in three groups: (1) simple if-then rules where actions are triggered when a condition is satisfied (e.g., if it is night-time, turn the television off), (2) relationship-based actions where personal, spatial and temporal relationships can be specified (e.g., when I wake up – for temporal, the next room – for spatial, and family – for personal), and (3) environment personalization where user preferences are taken into account (e.g., a user may prefer classical music).

Simple implementations of a similar rule-based system are Atooma (2015), Tasker (2015), Locale (2015) or Ifttt (20015). These tools allow customizing mobile functionalities following the "if x then y" structure. In this case, end-users can build the same type of applications where "x" and "y" can vary according to possible events that can be captured by the mobile phone (x) and actions that can be performed by the device, natively or by means of a specific application. Lucci and Patternò (2014) presented a comparison between Atooma, Tasker, and Locale with the objective of analysing the expressiveness and usability of this type of tools.

MircroApp [Cuccurullo et al. 2011] allows end-users to graphically create applications by combining (in sequence or parallel) actions (e.g., take a picture, send an email) that are offered by a mobile phone. In this case, the tool builds the dataflow between these actions automatically. This approach uses a colour code to differentiate between different types of parameters required by each action. In addition, the user can also specify whether a specific parameter is going to be set up at design time (being the same value used each time the composition is executed) or whether this is going to be provided during execution time.

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 to users with none programming skills and the advanced view,

which is targeted to more advanced users. While in the beginner's view users are assisted during the application creation, in the advanced view users have more freedom and control for the definition of the behaviour of the application. In fact, in the beginner's view users just need to configure the provided templates to build a specific application. However, in the advanced view users create a tree of blocks where input for a parent block is fed by the outputs of its child blocks.

Puzzle [Danado & Paternò 2014] is a framework that allows end-user creating mobile applications directly from a mobile device. It allows combining functionality provided by the own device, smart objects, and web services as if they were jigsaw pieces. The main benefit of using such metaphor is that users can start working with the given tool without a previous training.

Context Studio [Häkkilä et al. 2005] allows creating applications that activate mobile functions when a defined context-action rule is satisfied. The UI provided to define such rules is based on lists where the end-user first selects the action to perform (e.g., switching to mute or launching the camera), and then the trigger (e.g., shaking the device or drawing a circle in the screen of the device).

TouchDevelop [Athreya et al. 2012] is a mobile programming environment targeted to end-user programmers, i.e., users with programming knowledge, to create mobile applications. Applications are built in this case as scripts written in the TouchDevelop language. In addition to programming skills, in order to build applications with this environment, end-users need to know the features and constraints of the provided language.

Discussion. First, not all that these works provide abstractions that helps end-users without programming background to compose services. Even more, a visual environment is not provided for all of them since some are textual approaches. In addition, most of the above presented approaches only allow end-users to define an action (or set of actions) that must be executed when a condition (some of them defined only over values available at design time) is satisfied.

We differ from the above presented approaches in the fact that we allow end-users to go a step further in the definition of activity sequences in a visual way. As we introduce next, our solution provides end-users with mechanisms to create complex compositions that can include loops, parallel executions, or multiple conditional activities. Conditions can be defined either over activity outputs or over context conditions available at run time such time, weather or user position. In addition, we pay a special attention to the definition of input and output of activities, and how they must be linked to create sequences. It is not clear how the analysed works face this problem from an end-user perspective.

3. Motivating example

This section introduces a scenario based on a day in John's life, a 22 years old university student. The university where John is studying offers some services, which provide the university community with information about university facilities (i.e. library availability, available parking places, events celebrated at the campus, etc.).

John is using the university facilities as well as other existing services as follows: From Monday to Friday, every morning before leaving home, John first books a seat at the library of the university. Then, he checks the weather forecast for that day to decide how to reach the university. On rainy days he prefers to reach the university by bus, so he checks the schedule of the line bus that makes the route from his place to the university. On the other hand, on sunny days he prefers to go to the university by bike. Since the university offers a bike parking service, he books a place close to the library to park his bike. He also checks the state of the traffic flow along his route to know if there is some jam. In any case, John notifies his classmates about the library where his is going to stay by publishing this information via Facebook and Twitter.

Figure 1 shows the graphical representation of John's scenario in the BPMN notation, which is one of the most used notations to compose services. Note that John needs to use concepts such as events, sequence flows, or gateways in order to create the composition he needs with this notation. These concepts can be extremely difficult for end-users without notions about modelling and programming. In addition, this type of notations and languages are not conceived to be used in mobile devices.

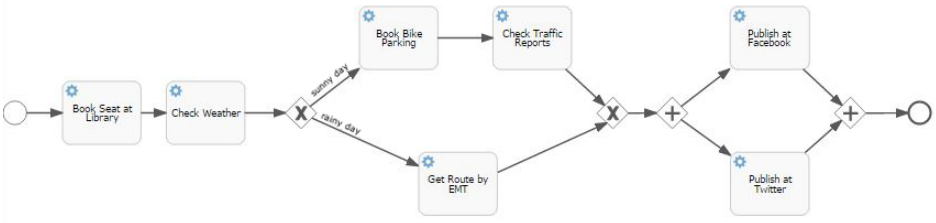


Figure 1. BPMN representation of John's scenario

John can use current mobile solutions targeted to end-users such as the ones presented above. However, as we have explained, they do not give support to create automations more complex than condition-action rules. This is not enough to support John's needs.

4 Rationale of the proposed solution

We need to provide end-users with a tool capable of composing services intuitively, not based on traditional composition languages such as BPMN. This tool must provide end-users with concepts that allow them to focus on

domain aspects instead of technical issues, but at the same time, provide them with enough expressivity to satisfy their needs.

According to [Van Deursen et al. 2000], the use of a DSVL is recommended to help end-users in the definition of domain concepts since they have proven to be more intuitive and easy to use than other options like general-purpose languages or textual languages. On the one hand, they are focused on a particular domain and provide specialized features to describe it, which helps end-users to understand the language. On the other hand, they provide visual elements that are better-suited to target end-user cognitive processes and understanding [Couper et al. 2004, Galitz 2002].

In addition, a DSVL should be semantically rich enough to obtain complex service compositions. A solution to this is that the proposed DSVL provides concepts semantically equivalent to the ones defined by the business process metamodel [BPDM, 2014], since it includes all the constructs required to create such compositions. However, to make sure that the DSVL provides more intuitive concepts than the ones provided in the mentioned metamodel, it is necessary to perform an abstraction effort to get closer such concepts to end-users. Such an effort was already performed in [Weber et al. 2008] with their proposed set of *change patterns*, a set of high level abstractions aimed at achieving flexible and easy adaptations of business processes. These abstractions are defined in terms of high level change operations (e.g. the creation of a parallel branch) which are based on the execution of a set of change primitives (e.g. add/delete activity). As opposed to change primitives, change pattern implementations typically guarantee model correctness after each transformation [Casati 1998] by associating pre-/post-conditions with high-level change operations, supporting the correctness-by-construction principle [Dadam & Reichert 2009]. For this purpose, structural restrictions on process models (e.g. block-structuredness) are imposed. This is a valuable characteristic to be considered on the definition of a DSVL for end-users, since it reduces the possibility that end-users make mistakes while composing services. In addition, a correct usage of change patterns allows speeding up the creation of the composition [Weber et al. 2008].

Therefore, in this work we take as basis a subset from the change patterns proposed by Weber at al. to allow end-users to create all main control-flow constructs (i.e. sequences, parallel branches, conditional branches, and loops). This subset includes AP1 (Insert Process Fragment) with its two variants, i.e., Serial and Parallel Insert, AP2 (Delete Process Fragment), AP8 (Embed Process Fragment in Loop), and AP10 (Embed Process Fragment in Conditional Branch). For illustration purposes, Figure 2 shows the changes introduced by the application of pattern AP8 into S to obtain S' where activity B is embedded in a loop. The result is a loop fragment that contains the selected activity.

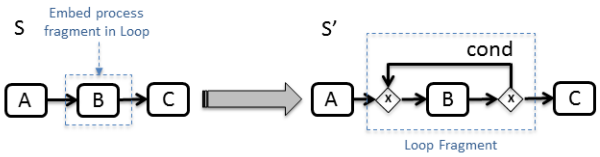


Figure 2. Example of the "Embed process fragment in loop" pattern

By using the mentioned subset of change patterns, we have defined a language semantically as rich as any other composition language such as BPMN but more intuitive and easy-to-use for end-users.

Finally, visual elements should be defined considering the device in which the DSVL is used. In our case, end-users must create service composition with a mobile device. Thus, the proposed visual elements have been designed in order to be used with this type of devices. In particular, the proposed DSVL is supported by EUCalipTool, which is an authoring mobile tool for end-users that has been designed by using patterns and guidelines for mobile development.

4.1 Example of usage

Let us provide the reader with a first contact with the proposed solution in order to see how end-users can create service compositions. Next sections explain its design and development in detail.

In order to create John's composition, he just need to use the EUCalipTool tool. Figure 3 shows some screenshots that illustrate the process of defining the service composition of John's scenario. A composition is created by using always the metaphor of "adding an element" to a container. The composition is the main container and can include activities or fragments. While activities are high level representations of services, fragments are structures inspired by change patterns (i.e., loop, parallel, and conditional structures) which may contain in turn activities or other fragments. However, end-users do not need to worry about the creation of such structures (i.e., create and link all the required modelling elements), instead they just need to add elements to a specific container. 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 & Ioannidou 2006].

Thus, after creating an empty composition, John accesses its graphical representation, which is shown in Screen 1. Initially, it is represented by an empty list. From this screen, John clicks the button with the "+" symbol, and accesses Screen 2 where the available services are shown as activities.

Figure 3. Example of usage of the proposed end-user mobile tool

From the list of activities, John adds the activity “Book Seat at Library” (see result in Screen 3). Next, he clicks again the “+” button and select a *Weather Predefined Item*. Predefined items are fragments with a predefined condition, which facilitates end-users the definition of actions that depends on conditions such as weather, location, time, etc. (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). John adds the activities "Book Bike Parking" and "Check Traffic Reports" to this branch by following the same steps 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).

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

5. A DSVL to support the composition of services by end-users

This section presents the DSVL proposed to support end-users in the creation of service compositions. DSVLs are defined by:

1. An *abstract syntax*, which indicate the concepts that define the language and the relationship among these concepts, independent of any particular representation or encoding.
2. A *visual concrete syntax*, which define the visual metaphors that end-users must use to create these concepts.

Furthermore, a proper *tool (or editor)* must allow end-users to use the concrete syntax’s visual metaphors in order to create descriptions based on the abstract syntax.

Next subsections present the abstract and concrete syntax of the DSVL. Note that one of the most important guidelines to be followed in the design of a language for end-users is their support to avoid errors [Repenning & Ioannidou 2006]. To achieve this, we focused on creating a DSVL and a tool that satisfy the principle of correctness-by-construction, i.e. whose simple use let avoiding errors. This aspect is highlighted as the elements of the DSVL are introduced.

5.1 Abstract Syntax

An abstract syntax for end-users should include concepts that can be easily understood and used by them, but at the same time, these concepts should be semantically rich enough to create the desired descriptions.

Thus, as we have explained above, we have defined an abstract syntax based on a set of change patterns. This abstract syntax is described by the metamodel shown in Figure 4. According to this figure:

- A *Composition* has a *name*, a *description* and a graphical *icon.* These properties are defined in an additional abstract class (*NamedElement*) because they are shared by other classes of the metamodel. They are used to semantically characterize compositions and activities.

  A composition is made up of a set of *Composition Elements*. Inspired by the set of change patterns presented above, a composition element can be of two basic types: activities and fragments. The fact of using only two types of elements facilitates the definition of a composition by end-users, since they only have to handle with these two concepts in contrast to languages such as BPMN where several concepts need to be understood (activities, events, gateways, connectors, swimlanes, etc.). The *previousElement* relationship between composition elements allows establishing the sequence order between activities and fragments.

- An *Activity* is a high level representation of a service. It has a *name*, a *description*, and a *graphical icon*, hiding end-users all the technological issues that are required to invoke the service. We use the term "activity" instead of service because it is closer to end-users' mental model [Engeström et al. 1999]. Thus, end-users focus on indicating the activities they want to perform in a composition, such as checking weather forecast, posting a message in Facebook, and so on.

  An activity may need one or more *inputs* (e.g. the location where the end-user wants to check the weather forecast or the message to be published in Facebook) and a *result* (e.g. the actual weather forecast for the given location). The input and the result of an activity correspond with the parameters and the return value of the associated service. Both have a *name* and a *description* in order to help end-users to understand their semantics. Note that compositions may have also an output and some inputs. However, this data is implicitly derived from the activities included in the composition. The output of the composition is defined from the output of the last activity included in the sequence that define the composition logic. For instance, if a composition finishes with an activity whose output is a weather forecast, this forecast constitutes also the output of the composition. Inputs of a composition correspond with activity's inputs that need to be provided at runtime. For instance, consider a composition that includes an activity that need to know the current location of the user. This input must be provided at runtime in order to perform the activity and therefore the composition. Consequently, this activity input constitutes an input of the composition.

  Regarding the introduction of input values, inputs have a *source*, which indicates how the value should be provided. According to the *InputSourceType* enumeration, it can be: *runtime*, which indicates that the input

must be introduced by end-users at runtime; *compositionTime*, which indicates that the input is defined at design time by the authoring end-user; and *ActivityOutput*, which indicates that the input is associated to the output of another activity (see *sourceResult* association). This last option is a key aspect in the creation of activity sequences, since an activity may need as input the result of a previous one.

- A *Fragment* is the result of applying one of the change patterns introduced above. They are composed by *Branches*, which contain a sequence of elements (activities or other fragments). Each fragment provides specific execution logic to the elements of their branches:
  - *Parallel* fragment: indicates that the elements of two or more branches must be executed at the same time. It is the resultant element of applying the "Insert Process Fragment in Parallel" change pattern.
  - *Conditional* fragment: indicates that the elements of a branch must be executed if a condition is fulfilled. It can have one or more branches, each of them with its own condition. It is the resultant element of applying the "Embed Process Fragment in Conditional Branch" change pattern.
  - *Loop* fragment: indicates that the elements of one branch must be repeated while a condition is satisfied. It is the resultant element of applying the "Embed Process Fragment in Loop" change pattern.

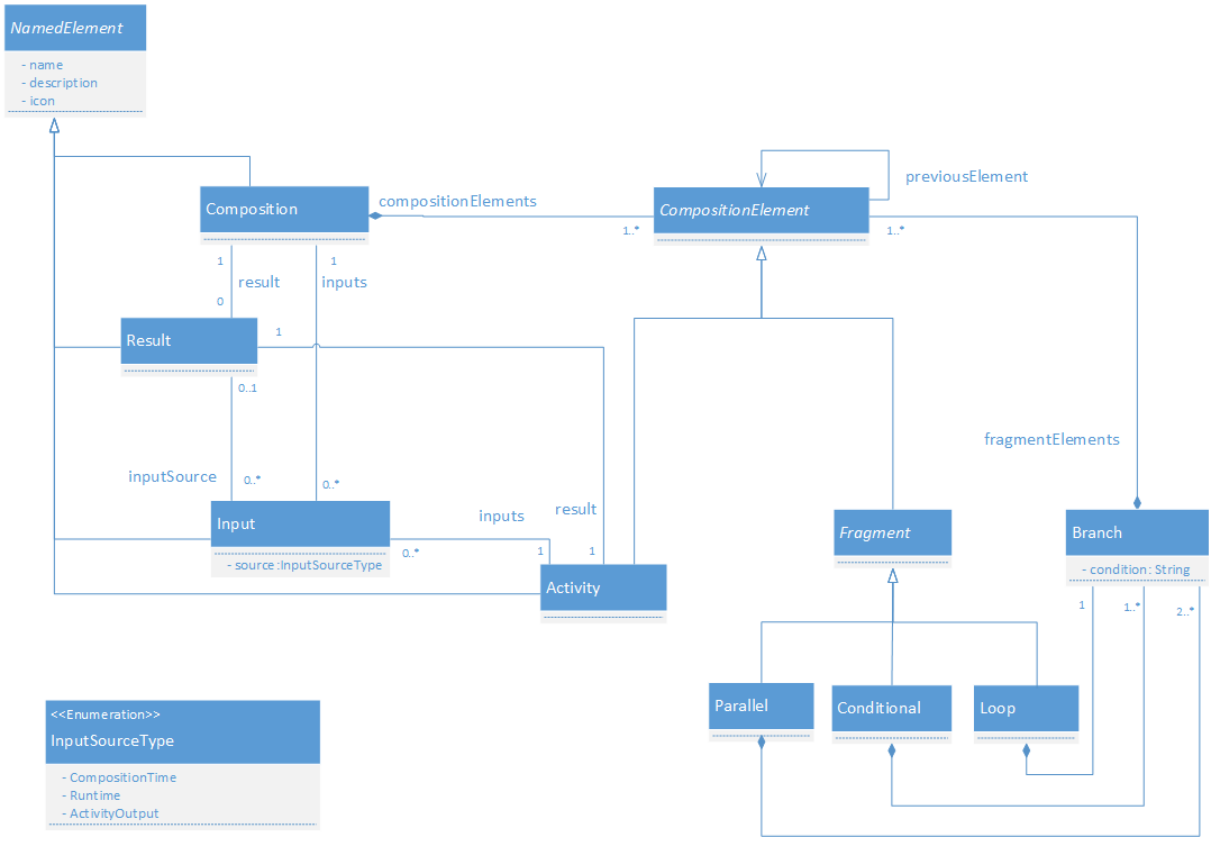


Figure 4. Abstract syntax metamodel

Note that there is not a resultant element for both "Insert Process Fragment in Serial" and "Delete Process Fragment" change patterns. The first pattern is used to create sequences of elements and we capture this information

by using the above introduce *previousElement* relationship. Regarding the second pattern, the result of its application is the removing of an element from a composition, and it is not reflected in the abstract syntax. It is supported directly by EUCalipTool.

In order to better understand the most important concepts of this metamodel, Figure 5 identifies them in the BPMN description of the running example. This process is composed of a sequence fragment of four composition elements: two activities that are followed by a conditional fragment and a parallel fragment.

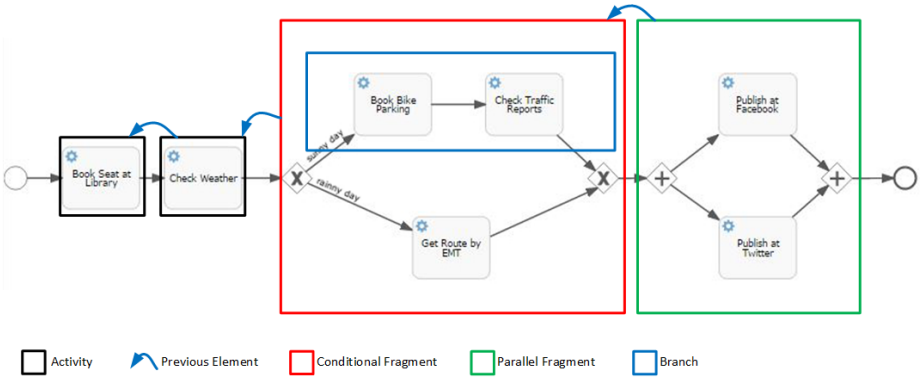


Figure 5. Identification of composition elements in the running example

5.2 Concrete Syntax

In this section, we introduce the visual representations that allow end-users to create descriptions based on the abstract concepts presented above. These representations have been defined by using patterns and graphical components specifically designed for mobile devices.

Compositions

A *Composition* is initially created by indicating a name, a description, and a graphical icon by means of a form created with this purpose. A snapshot of this form is further presented in Section 6.2 (see Figure 15A). Once a composition is created, end-users access its graphical representation to include the desired services.

In [Danado & Paternò 2014], different metaphors were evaluated by end-users in order to know which ones were most intuitive to connect components and compose various arrangements. The puzzle and workflow metaphors were the two most ranked. We have based 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 using the List layout (see Figure 6), 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. For each activity in the List, its name is shown. For each fragment, the type of the fragment

and the branches that it contains are shown. The elements of each branch are also displayed by using the List layout.

On the other hand, each element of a composition is connected graphically to the next one through an inverted little triangle. This aspect has been inspired by the jigsaw metaphor [Renger et al. 2008], which defines pieces inserted into other ones to reinforce the notion of connection or combination of elements. We have used a similar solution to evoke end-users the idea of connecting activities and/or fragments.

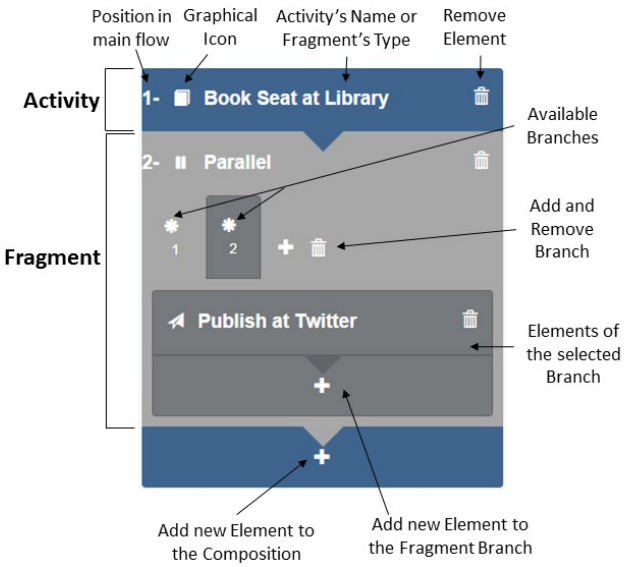


Figure 6. Graphical representation of a composition

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 the content of a fragment (see Figure 6). This button is placed at the location where the new element will be added in order to help end-users to create a mental representation of the result of the action before performing it.

Finally, a delete button complements composition's and fragment's elements in order to remove them. This is an icon-based button that shows the image of a trash, which is broadly accepted to represent the removing action. It is displayed at the right side of each element.

Composition elements

In order to add elements to a composition, a tabbed component has been defined. It is accessed when end-users click some of the available + *buttons*. It has three tabs (see Figure 7): the two first (Figures 7A and 7B) allow adding activities and fragments. The third one represents predefined items (Figure 7C), which provides support for the creation of conditional fragments in an easier way.

- The *Activity* tab (see Figure 7A) allows end-users to add an activity either to the composition or to a fragment. To do so, end-users just need to select the desired one. Since the number of activities displayed there can be

large, this tab implements the Continuous Filter pattern [Van Welie & Trætteberg 2000] in order to facilitate finding a specific one (see the implemented filter in Figure7A).

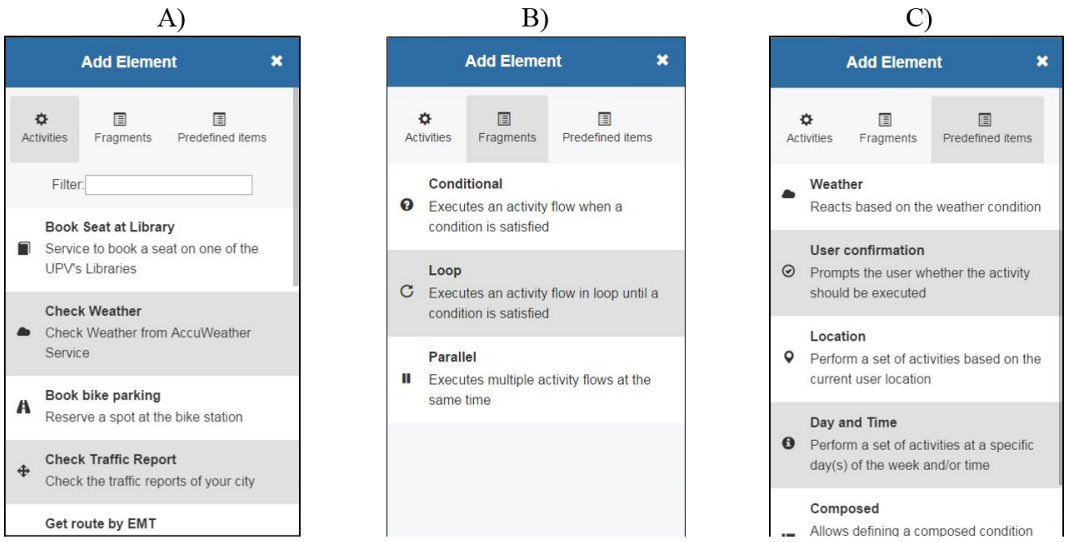


Figure 7. Tabs for adding elements

- The *Fragments* tab (see Figure 7B) provides end-users with the three basic fragments defined in the abstract syntax: conditional, loop and parallel.

When a fragment is added, it is created with some branches by default. The number of branches and the possibility of adding new ones depend on the type of fragment. According to the abstract syntax presented above:

– A parallel fragment must have at least two branches. Thus, when this fragment is added, two branches are automatically defined. In addition, end-users have the possibility of adding new ones. Branches can also be removed when more than two are defined. Figure 8A shows a newly added parallel fragment. At this point end-users can only add new branches. Removing branches is not allowed since the parallel fragment requires at least two branches to be a valid fragment.

– A conditional fragment must have at least one branch. Thus, when this fragment is added, a branch is automatically defined. In addition, end-users have the possibility of adding new ones. Branches can also be removed when more than one are defined. Figure 8B shows a conditional fragment with two branches. In this case, end-users can add new branches or remove existing ones.

– A loop fragment must have one and only one branch. Thus, when this fragment is added, a branch is automatically defined. In this case, end-users do not have the possibility of adding or removing branches. Figure 8C shows a newly added loop fragment. In this case, end-users are not able neither to add nor remove branches.

Note that the creation of fragments with default branches, and the constraints defined over the actions of adding and removing branches help satisfying the correctness-by-construction principle. Fragments always will have the

required branches by construction, and end-users cannot do actions that change this. Therefore, we reduce the possibility of creating incorrect descriptions to only two scenarios:

- When a new fragment is added its branches are initially empty, with no elements in it, which is not correct according to the abstract syntax presented above.
- When a composition is initially created it has no elements, which is neither correct according to the abstract syntax.

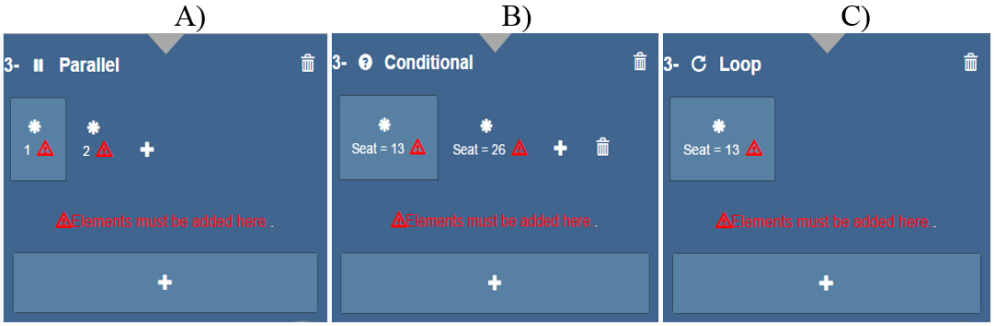


Figure 8. Branches automatically defined in each type of fragment

To solve these problems, we had two main options: (1) create a default content in order to guarantee correctness-by-construction, or (2) mark fragments and composition as incorrect elements to force end-users to complete them. The first option was discarded because it implied adding elements to the composition (e.g. a default activity) that may not fit with the needs of end-users, and then, they would have to modify or replace this default element. Thus, we chose the option of marking newly created fragments and compositions as uncompleted elements and, following the guidelines presented in [Haines et al. 2010], a message that alerts end-users how to fix the problem is shown. Fragments presented in Figure 8 show representative examples of these messages. In addition, end-users cannot add new elements until existing ones are correctly completed. This restriction as well as the constraint related to fragment braches are controlled by EUCalipTool.

Following with fragments, note that the conditional and loop fragments require the definition of conditions. In the first case, a condition for each branch should be defined in order to indicate when its elements must be executed. In the case of the loop, only one condition must be defined. While it is true, the elements of its unique branch must be executed iteratively. The conditions can be defined using either the output of previous activities or context properties (i.e. weather status, user location, etc.)

In order to define these conditions, the form in Figure 9 has been defined. It allows end-users to create a condition by comparing the output of previous activities or a context property with the output of an activity or a specific value. In the example below, a post on Facebook is published if the seat booked by the activity *Book Seat at Library* has the number 13.

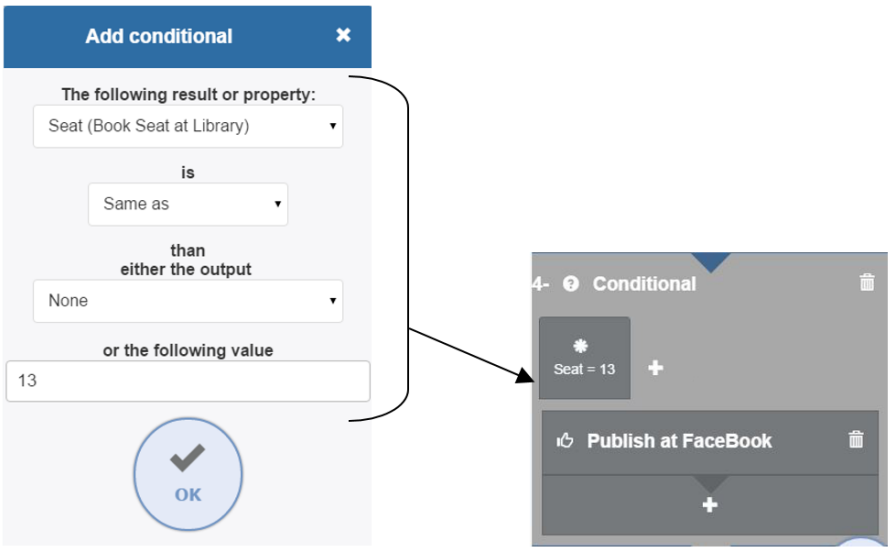


Figure 9. Configuration of a condition

- The *Predefined Items* tab (see Figure 7C) facilitates end-users the creation of conditional fragments. This tab includes a set of conditional fragments that have been created with predefined conditions. We have been inspired by end-user guidelines that promote the provision of reusable components [Segal 2005]. The conditions have been defined from the analysis of related work, and from our experience in the definition of context-aware systems [Serral et al. 2013], activity and task modelling [Uden 2008, Valderas 2006], and adaptive business process [Ayora 2013]. They are the following:
  - *Weather*: a conditional fragment that allows end-users to perform specific activities depending on weather, i.e. depending on whether it is a sunny, cloudy, rainy, or windy day.
  - *User confirmation*: a conditional fragment that allows end-users to perform specific activities previous confirmation. This fragment is predefined to ask end-users at runtime if they want to execute some activities.
  - *Location*: a conditional fragment that allows end-users to perform specific activities according to their location. This condition is checked at runtime by interacting with the geolocation capabilities of the end-user mobile device. It is possible to indicate a scope range in order to indicate how close to the location the end-user must be in order to perform the activities.
  - *Day and Time*: a conditional fragment that is predefined to perform activities at a specific day(s) of the week and/or time.
  - *Composed*: a conditional fragment that allows end-users to use the previous predefined conditions in order to create a composed one. For instance, this fragment allows end-users to define a set of activities that must be performed in a location (Location condition) at a specific time (Day and Time condition).

In order to configure the predefined conditions, specific graphical components have been designed. All of them have been defined taking into account the study presented in [Galitz 2002], which recommend the selection of data instead of typing it for avoiding end-users' errors. As representative examples, Figure 10 shows the screens that allow end-users to configure a Weather condition (A), a Location condition (B) and a Day and Time condition (C).

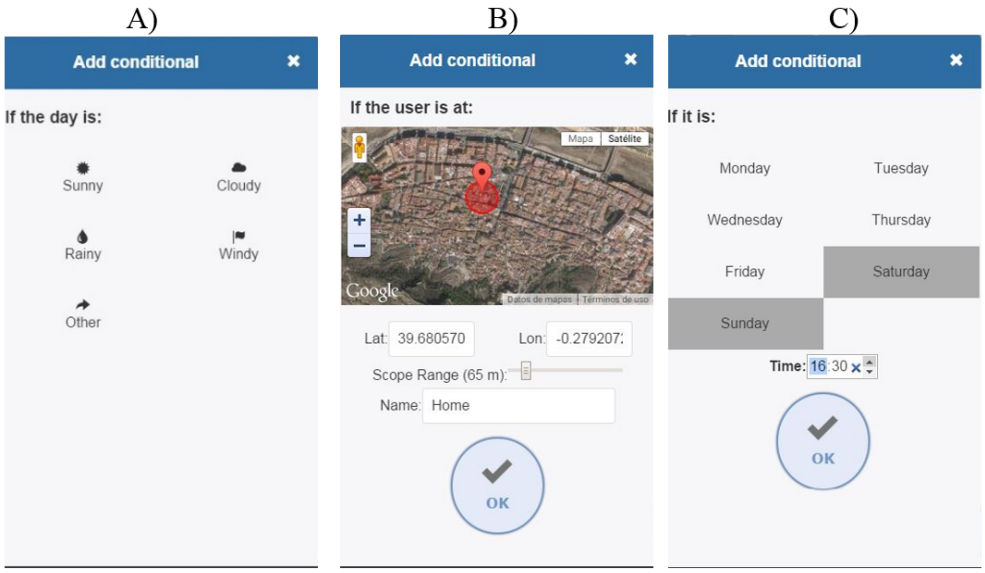


Figure 10. Configuration of the predefined items

Note that a predefined item may imply the execution of implicit actions in order to check the conditions. For instance, the Weather predefined item includes the action of knowing the forecast, although end-users do not define it explicitly. Thus, the use of this type of elements not only helps end-users by providing them with predefined conditions, but also implies implicit logics (e.g. the execution of the weather forecast service) that end-users do not need to define. In other languages such as BPMN these actions must be defined explicitly. As a representative example, Figure 11 shows the BPMN part of the running example that is equivalent to a Weather predefined item.

Note also that these fragments can be used either to involve a set of elements such as in Figure 11 example, or as a wrapper of a full composition. This last case allows end-users to easily indicate some conditions (e.g. a day and a time, a temperature, a location) in which a composition should be triggered. For instance, Figure 12 shows the *ProfAtUni* composition, which contains the activities that should be performed when a professor is close to the Universitat Politècnica de València (UPV) on Monday, Thursday and Wednesday at 8:00.

Finally, it is worth noting that predefined items also help to satisfy the correctness-by-construction principle since they constitute a mechanism for allowing end-users to create conditional fragments by configuring a condition instead of creating it from scratch.

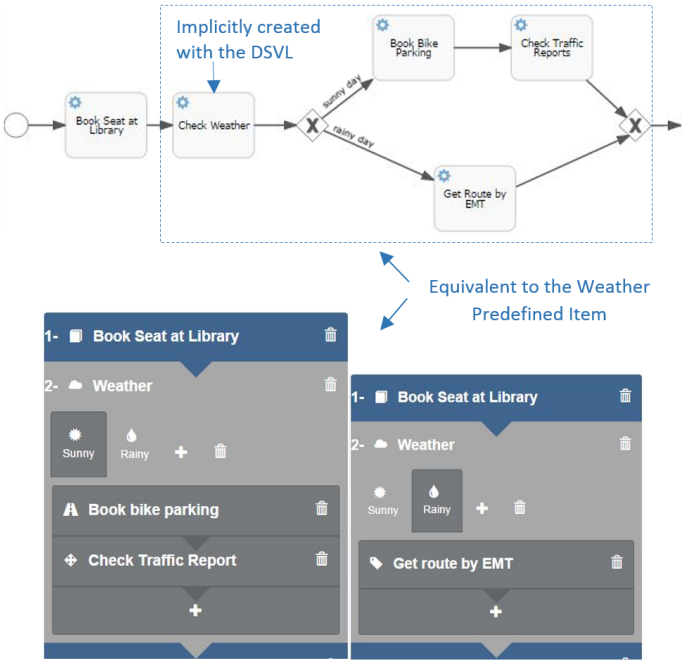


Figure 11. BPMN specification equivalent to a Weather predefined item

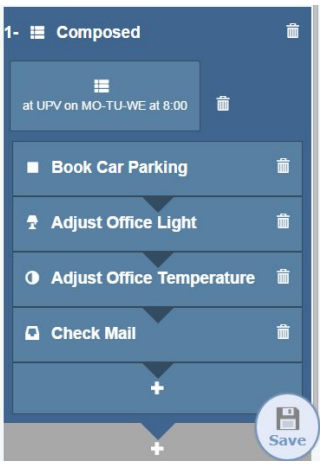


Figure 12. Using a Composed Fragment as Composition Wrapper

Activity inputs

Activity Inputs represent the data that an activity may need to be properly performed. According to the abstract syntax presented above, this data can be obtained from three different sources: (1) from the end-user at composition time, (2) from the end-user at runtime, or (3) from the output of a previous activity.

In order to define this aspect, we have designed a graphical component that allows end-users to configure the inputs of each activity one by one. It is shown in Figure 13A. Using the arrowed buttons placed at the top and bottom sides of the component, end-users can browse the activities of the composition. The central part shows the inputs of each activity and the value source assigned by default (this is explained in detail in Section 6.3). Tapping

on this central part end-users can access the form in Figure 13B, which allows them to define the source of the inputs of an activity. In this example, the end-user is defining the source of the input *Bike Station* of the activity *Book bike parking*. This input is a location, so the end-user has four options in this case:

- *Predefined Location*: a location defined at design time by using a graphical component similar to the one presented in Figure 10B.
- *Current Location*: a location defined automatically at runtime from the current user position.
- *Asked when needed*: the user gives the location at runtime.
- *Library*: this is the output of the previous activity *Book Seat at Library*. All outputs of previous activities that have the same type as the input are shown in this list of options. If this option is selected, the Bike Station input takes the value of the Library output.

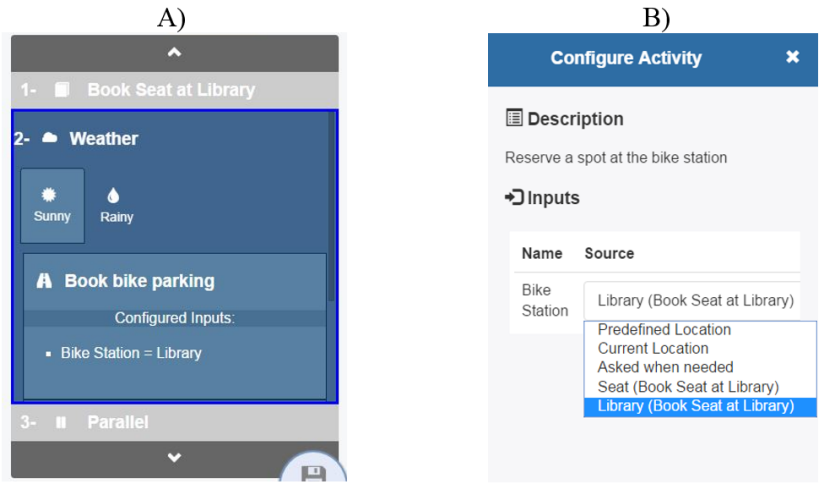


Figure 13. Input Configuration for the Book Bike Parking activity

6 End-user Authoring Mobile Tool

In this section, we present EUCalipTool. This tool implements the DSVL presented above in order to provide end-users with support to compose services by using their mobile devices. There are two versions of this tool[2]:

- WebApp version: it has been implemented by using web technology (HTML5, CSS3, Javascript) and can be accessed from any browser of a mobile device, without any type of installation.
- Android version: it has been implemented as a native app for the Android platform. It does not require any browser since the app is installed into an android device.

[2] Both versions can be accessed from the download section of the EUCalipTool web page at https://tatami.dsic.upv.es/eucaliptool/

This tool implements the DSVL presented above to allow end-users to compose services by using high-level descriptions of them. In addition, EUCalipTool incorporates design issues in order to help end-users to properly create a composition. Next, we analyse them in detail.

6.1 User Guidance

The most important aspect of the tool implementation is the application of the Wizard pattern [Van Welie & Trætteberg 2000]. This pattern is used to increase software usability. It promotes interfaces that guide end-users through the different steps that may involve a task. This improves the learning and memorization of the steps to be performed and keeps users from missing important things. We have used this pattern to guide end-users through the steps of creating a service composition. These steps are the following (see Figure 14):

1. **Creation**: a composition is created and initialized. It can be done from scratch or taking an existing one as base. This second option is explained in detail in the next subsection.
2. **Sequence Definition**: The sequence of activities and fragments must be defined in this step. End-users can add activities or fragments by using the defined concrete syntax.
3. **Input Configuration**: In this step, end-users must configure the inputs of each activity. The graphical component to do so has been presented above.

As we can see in Figure 14, the wizard pattern has been implemented as follows:

- The steps to be done in the creation of a composition are always shown in the upper side of each screen, and the current step is highlighted.
- Each time a new step is started end-users are shown with a screen that informs about this fact.
- End-users can use the upper side area where steps are shown to directly access the current and previous steps (step captions become into links after performing them), which helps them to go back in order to modify some aspect.

This guided process helps end-users to focus on each task separately: first they focus their efforts on create a composition (step 1); next they define the sequence of activities and fragments (Step 2); and finally they focus on managing inputs sources (step 3). This provides end-users with an incremental process, helping them to feel that what they are doing is not difficult but they are building up the necessary skills incrementally [Repenning & Ioannidou 2006].

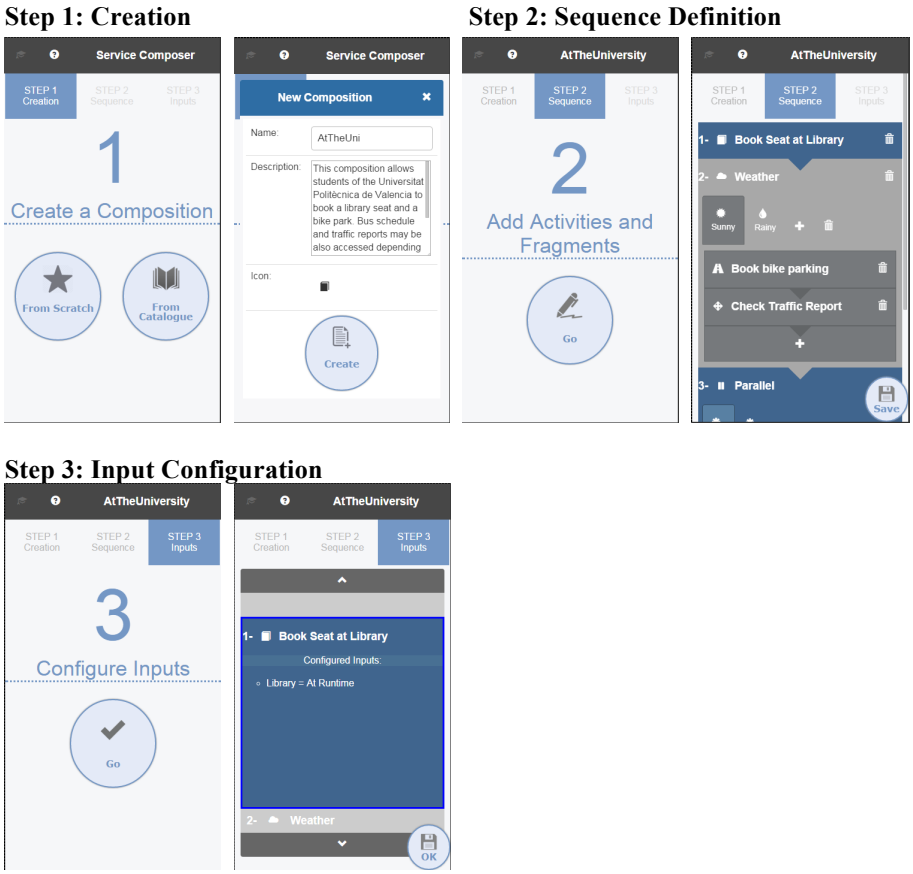


Figure 14. Screens of the steps required to create a composition

6.2 Two ways of creating a composition

There are two ways of creating a composition: from scratch and from a catalogue (see first step in Figure 14). With the first option, end-users need to initialize the composition by indicating a name, a description, and a graphical icon. The form to introduce this data is presented in Figure 15A.

With the second option, end-users can create a composition from an existing one. In the context of End-user Development, it is highly recommended that end-users can access a library of predefined components in order to select and use one of them as starting point [Segal 2005]. EUCalipTool allows end-users to create a service composition by selecting an existing one from a catalogue, and customizing it according to their needs. This catalogue includes: (1) examples that we have defined to be used by end-users, and (2) previous compositions created by the own end-user.

Figure 15B shows a snapshot of a catalogue with the list of composition examples that we have created. These examples cover the use of all the constructors provided by the DSVL in order to help end-users to learn them.

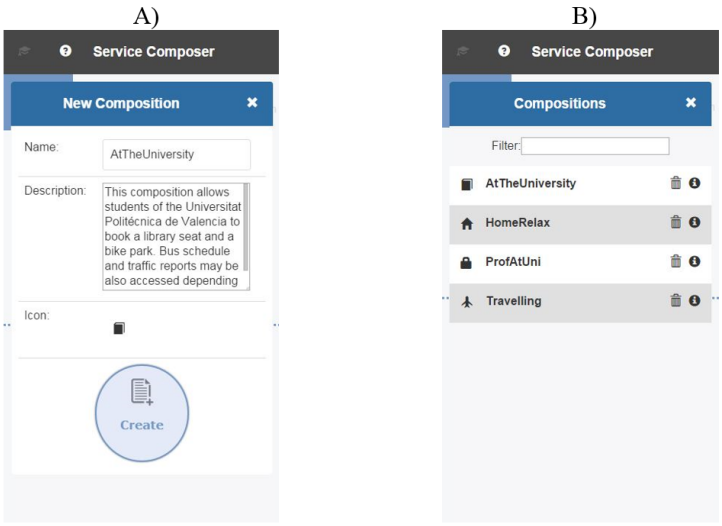


Figure 15. Creation from scratch (A) or from catalogue (B)

6.3 Automatic default configuration for activity inputs

In the third step of the composition process, end-users have to define the source for each activity input. To do so, a specific concrete syntax was defined and presented above (see Figure 13)**.**

Additionally, EUCalipTool provides end-users with automatic support to manage this aspect. By default, the tool links the inputs of each activity to the outputs of the preceding one, always when their types match. Information about input and output types can be obtained from the parameters of their associated services, which are described in a Service Registry [Mansanet et al. 2015]. If input and output types do not match, the input of an activity must be introduced at runtime.

Note that this default configuration allows satisfying the correctness-by-construction principle at syntactic level, since inputs have always associated a correct source. However, it is clear that it may be not semantically correct according to end-users' needs. In these cases, end-users can use the concrete syntax shown in Figure 13 to validate the default definition and change it if needed.

6.4 Additional Usability Issues

In this section, we introduce additional usability issues that have been considered in the design of EUCalipTool.

Selection rather than typing. The studies presented in [Galitz 2002] recommend the use of selection components as means to allow users to introduce data since information became less familiar, or subject to spelling or typing errors if entry fields are used. In addition, the study in [Couper 2004] states that end-users tend to select from the entire list of options that they are first presented with. They rarely make an effort to find additional options through scrolling.

Thus, we have designed user interfaces that avoid users to type data as much as possible and present available options in the same screen. As representative example, note how predefined conditions are configured without typing, and the different options that can be selected are shown in the same screen (see Figure 10).

Matching the real world. According to usability heuristics presented by Nielsen (2005), systems should provide a design that help matching items between system and real world. One of the most used techniques to achieve this is the inclusion of icons that graphically represent the actions that users can do by selecting items. In this case, figures presented along this paper show how the different available options are always complemented with a representative graphical icon.

Help. According to Nielsen (2005), it is recommended to provide help and documentation to end-users in order to facilitate the learning of use. To implement this in a mobile environment we have followed some of the patterns presented by Neil (2014). In particular, we have used the transparency pattern in order to provide end-users with indication of use. This pattern proposes the implementation of a see-through layer with a usage description positioned over the actual screen content. Figure 16A shows a screen with this type of help. This help appears automatically the first time end-users access a screen. It does not appear in later accesses to the same screen. It can be disabled if end-users do not need it in further steps, and can be enabled again when needed. To do so, end-users just need to use the button with the symbol , which is located at the left top corner of the screen.

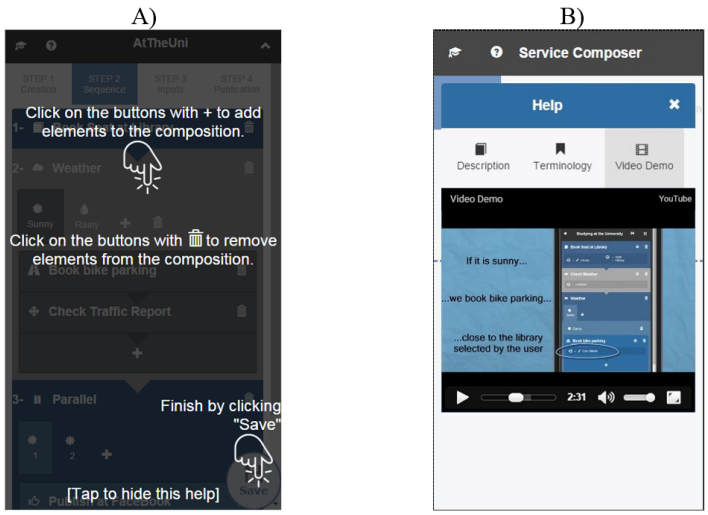


Figure 16. Layered Help and Video Demo

Another pattern that has been followed from [Neil 2014] is the use of a video demo, which is a way of invitation for applications that demonstrates the application in action. In particular, a tabbed panel help is always accessible by end-users from the button in the header with the symbol . This panel includes a general description of the application, a definition of the most important terminology, and the video demo. Figure 16B shows this screen with the video demo tab activated

7 Software Architecture

EUCalipTool allows end-users to focus on composing activities and avoid handling with technological issues of service implementations such as URLs, protocols, ports, and so on. To achieve this, a three layer architecture is proposed (see Figure 17). The *Service Layer* encompasses the services developed by professionals. Services are implemented by using the technology each professional considered convenient (e.g. SOAP or REST). The *Application Layer* provides end-users with EUCalipTool. Finally, the *Component Layer* hosts a Service Registry that plays the role of gateway between EUCalipTool and services' implementation.

The proposed registry maintains two facets of services: (1) invocation facet, which includes all the technological aspects of a service (e.g. protocol, url, port, parameters, etc.). This data is used to manage the invocation of a service at runtime, and it is hide to end-users at composition time. And (2) semantic facet, which describes the behaviour and goal of each service in such a way end-users can manage it. To make a service available for end-users, developers must register it into the registry by defining both facets (invocation and semantic). To do so, a web frontend is provided. End-users only need to interact with the high level representations provided by the second facet, which is accessed by EUCalipTool.

The Service Registry is implemented as a Java Web module, and publishes a REST API to interact with it through the HTTP protocol. JSON is used as interchange data format.

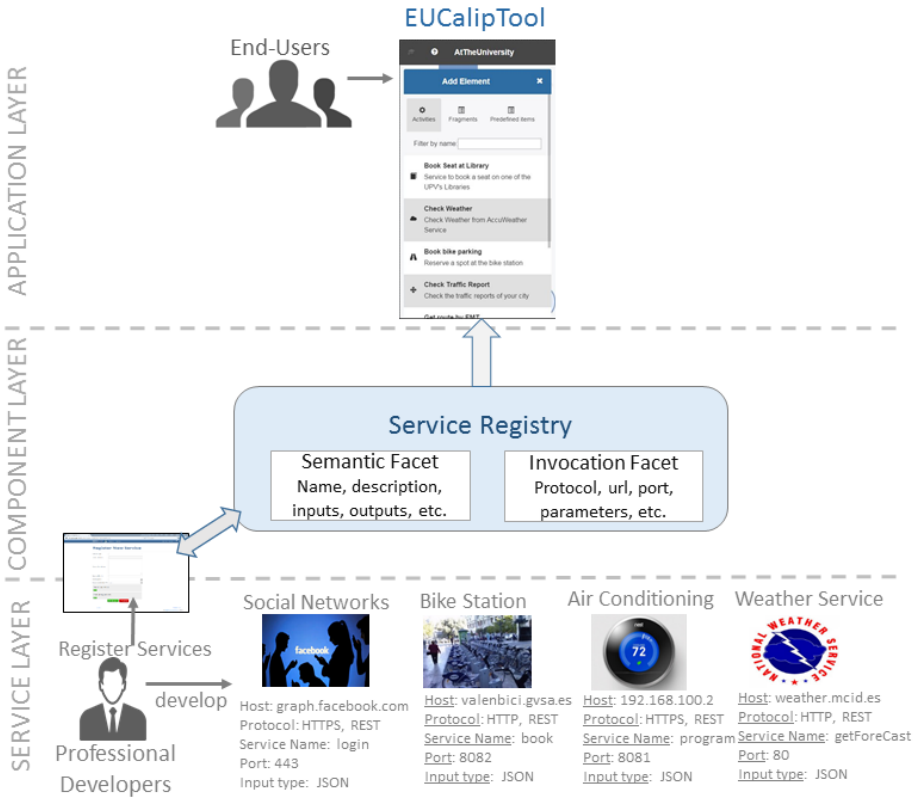


Figure 17. Software Architecture

8 Evaluation

This section presents the evaluation done to our work. First, we present a comparison between our DSVL and BPMN. Next, we present the usability evaluation test of EUCalipTool performed with real end-users.

8.1 DSVL vs BPMN

One of the motivations to create a DSVL for end-users was to provide a language for composing services that was simpler than existing ones, such as BPMN. Thus, we did a comparison between our DSVL and BPMN in order to determine if this goal was achieved.

The comparison was done by analysing the steps needed to model several scenarios with both solutions. We determined the main basic scenarios that users can face when composing activities, modelled them with both solutions, and measured the steps required in each of them. By steps we mean actions to create an element of the language.

As representative example, let us consider the definition of two activities, "post a message in Facebook" and "post a message in Twitter", which John wants to do at the same time, i.e., in parallel (see Figure 1):

- In order to model this scenario with BPMN, we have to (see Figure 17A): (1) create a split parallel gateway; (2) connect it to the previous element; (3) create one activity; (4) connect it to the gateway; (5) create the other activity; (6) connect it to the gateway; (7) create a join parallel gateway; (8) connect one activity to it; (9) connect the other activity to it; and (10) connect the join parallel gateway to the next element.
- With the proposed DSVL, end-users just need to (see Figure 17B): (1) add a parallel fragment, (2) add one activity, and (3) add the other activity. Although possible, no branches need to be created since two are added by default to the fragment; the fragment is connected to the previous element by default; it does not require the use of gateways to define the starting and ending point of the parallel behaviour; and it will be automatically connected to the next added element.

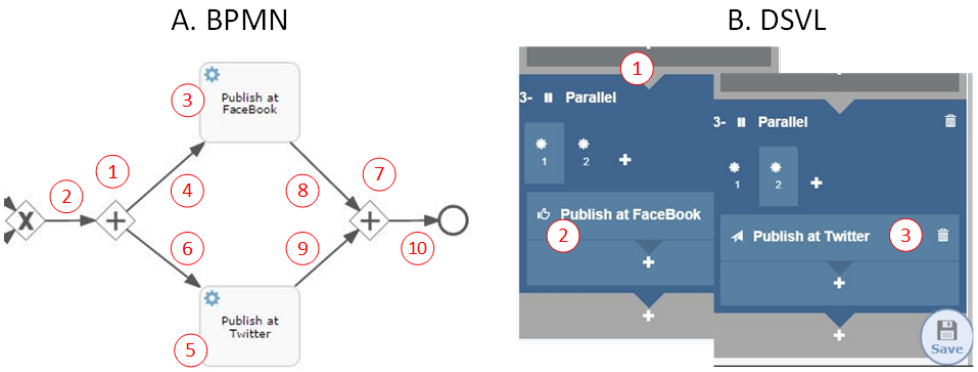


Figure 17. Steps to create parallel behaviour with BPMN and with the DSVL

Thus, by using BPMN we need 10 steps in order to define the proposed scenario. In contrast, with our DSVL we just need 3 steps to describe the same scenario. Table 1 presents a summary of all the evaluated scenarios. As we can see, the use of the proposed DSVL allows end-users to create service compositions with fewer steps than using BPMN. In particular, we can reduce the number of steps between a 60% and 80%.

Table 1. Summary of the results obtained with scenarios.

	Scenario	Steps in BPMN	Steps in DSVL
1	Create a composition with one activity	5 +1 Add start event +1 Add end event +1 Add activity +2 create links between elements	1 +1 Add activity
2	Add one activity in sequence	3 +1 Add activity +2 create links with previous and next elements	1 +1 Add activity
3	Add two activities in parallel	10 +2 Add parallel gateways +2 Add activity between gateways +4 create links between new elements +2 create links with previous and next elements	3 +1 Add parallel fragment +2 Add activities
4	Add an activity whose execution depends on a condition	8 +2 Add exclusive gateways +1 Define condition +1 Add activity between gateways +2 create links between new elements +2 create links with previous and next elements	3 +1 Add conditional fragment +1 Define condition +1 Add activity
5	Add an activity that must be executed iteratively while a condition is satisfied	9 +2 Add exclusive gateways +1 Define condition to return +1 Add activity between gateways +2 create links between new elements +2 create links with previous and next elements +1 create return link	3 +1 Add loop fragment +1 Define condition +1 Add activity
6	Add an activity whose execution depends on weather conditions, user confirmation, user location, or day and time	10 +1 Add activity to obtain weather conditions, user confirmation, user location, or day and time +2 Add exclusive gateways +1 Define condition +1 Add activity between gateways +3 create links between new elements +2 create links with previous and next elements	3 +1 Add predefined item +1 Define condition +1 Add activity

8.2 EUCalipTool Usability Evaluation

The design of EUCalipTool was performed by following an incremental and iterative design process [Larman & Basili 2003]. First, we created a set of user interfaces mock-ups that were validated by members of our research group and students of Computer Science at the Universitat Politècnica de València. After applying the corrections detected through this pre-test, mock-ups were implemented as a web front-end for mobile devices by using HTML, CSS, and Javascript. This web front-end was then evaluated by real end-users in an experiment done in a Summer School at the University of Coimbra [ICIS 2015]. We used this feedback to improve the web front-end and create the current version of EUCalipTool, which has been presented in this paper. For instance, the version used in the experiment forced users to define the sequence of activities at the same screen than input activities. We detected that this aspect overloaded end-users since they had to do two different tasks at the same time. Thus, we split these tasks into two different steps and screens. We also detected that additional mechanisms for guiding end-users in the composition process were needed. For this purpose, we implemented the wizard pattern that shows end-users the steps that need to be performed at the upper side of each screen.

In order to validate the current version of the tool, we applied a case study based evaluation by following the research methodology practices provided by Runeson and Höst (2009) (we also applied this methodology in the previous experiment introduced above). These practices describe how to conduct and report case studies and recommend how to design and plan the case studies before performing them.

Design of the case study. By using these practices, we designed and developed a case study that gives support to several scenarios that a student can perform in her daily life at the university, home or travelling. The running example based on John is part of them (see Section 3). We then conducted an experiment in which students of different areas and degrees of the Universitat Politècnica de València performed these scenarios adopting the role of John by using either the Webapp version of EUCaliptool or the Andriod native one.

A total of 17 subjects between 23 and 54 years old participated in the experiment (six female and eleven male). Most of them use mobile devices daily but only 7 of them had a strong background in computer science. We arranged several sessions in which the subjects first explored the tool to know its functionality and then carried out the scenarios under our supervision. To collect and analyse the results of the evaluated tool, we used a demographic questionnaire and an adapted IBM Post-Study Usability questionnaire[3] [Lewis 1995]. The last questionnaire is a 19-item instrument for assessing user satisfaction with system usability. In addition, the items of the

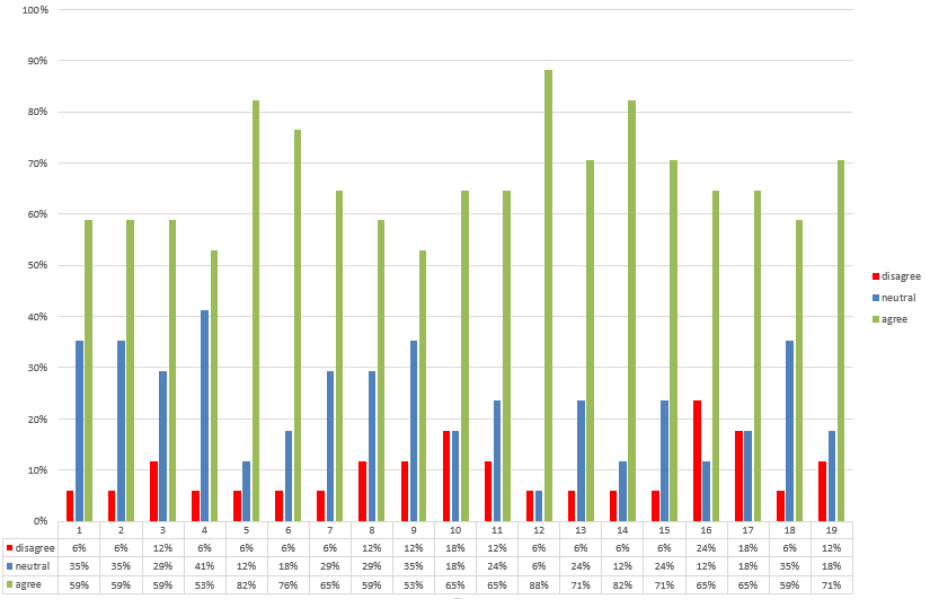
[3] Both questionnaires can be accessed at:
https://docs.google.com/forms/d/14fiIYym0fbRIi_v7ROxzSIetTYNQQYEhkMMNoyAhDNY/viewform?usp=send_form
https://docs.google.com/forms/d/18Y3vbDmq84J5BMPOqUTCjaDvUxicUtgmZQVx7u8k1e4/viewform

questionnaire ask subjects the following: if the tool was easy to learn to use, which allows us to measure the learnability of the tool; and if they were able to efficiently complete the tasks using this tool, which allows us to evaluate the applicability of the tool. We applied a Likert scale from 1 (lowest score) to 7 (highest score) points to evaluate the items of the questionnaire.

Evaluation results. Figure 27 shows a graphic with the obtained results. With regard to the learnability of the tool, the subjects of the experiment commented that the tool was easy to learn to use (question 7). Specifically, 65% of the subjects gave the tool a score between 5 and 7. They said that the organization of the interfaces and the guided steps helped them to easily learn how to compose activities. However, they found the language used in some parts of the tool (e.g. in the screen to define conditions) a little difficult to understand (question 10). The problem was the use of some terms such as 'conditional operator', 'branch', 'at runtime', and so on, which seems to be too much technical. We redesigned the full interface to change this terminology.

With regard to the applicability of the tool, 82% of the subjects involved in the evaluation gave the tool a score between 5 and 7, since they perceived that the tool allowed them to be more efficient (question 5). With regard to the functionality provided by the tool (question 18), 59% of the subjects gave the tool a score between 5 and 7. Most problems found regarding this aspect were related to the possibility of editing conditions, which was not allowed (i.e., they had to delete a conditional branch and add a new one). We improved this problem by allowing the edition of fragment conditions. Some of them propose the possibility of adding multiple activities at the same time. We are currently evaluating this option.

With regard to the usability of the tool, the results revealed that the tool was clear to use (question 15). Most subjects found the interface to be friendly and easy to use (question 16). The worst score obtained from the questionnaire was for the questions that determine the information quality of the message errors and help (questions 9 and 10). The subjects commented that error messages were not always clear enough to correct them (question 9), which was partially related with the problem of using technical concepts. Some subjects (most of them with basic computer knowledge) also explained that the tutorial should be easier to read (question 10). To improve this aspect, we are currently working to rewriting the tutorial in order to make it simpler.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
disagree	6%	6%	12%	6%	6%	6%	6%	12%	12%	18%	12%	6%	6%	6%	6%	24%	18%	6%	12%
neutral	35%	35%	29%	41%	12%	18%	29%	29%	35%	18%	24%	6%	24%	12%	24%	12%	18%	35%	18%
agree	59%	59%	59%	53%	82%	76%	65%	59%	53%	65%	65%	88%	71%	82%	71%	65%	65%	59%	71%

Figure 27. Usability Questionnaire results

Finally, 71% of the subjects stated, in overall, their satisfaction towards the tool (question 19), resulting in average a 5.05 from the 7 point liker scale, which is quite an acceptable score.

9. Conclusions and Further Work

In this paper, we have presented a research work to support end-users in the composition of services by using their mobile devices. To do so, we have presented a DSVL based on a specific set of change patterns, which provides a visual language for end-users simpler than existing ones for activity composition such as BPMN. This DSVL is supported by EUCalipTool, which is targeted to end-users without programming skills desiring to start creating complex service compositions by using a mobile device. From our results, EUCalipTool has addressed our initial goal and enables end-users to easily compose services, by using high-level service descriptions.

One important aim of this work is to contribute to the research on mobile end-user development in order to encourage end-users to become producers of services. In a world where people's environment is plenty of services that support their life style, this can be a crucial aspect to support their needs.

As further work, we plan to improve some of the problems detected in the usability evaluation such as providing new capabilities to the tool (e.g. adding multiple activities at the same time), or improving the provided help and messages. In addition, we want to do more usability experiments focused on other profiles of users and scenarios. Currently, we have focused on the tasks that a student can potentially do at university, home, or when traveling. We want to analyse other scenarios with professional end-users (those who use computers at work although they are not computer engineers) and pure end-users (those that do not usually use computers).

We are also working on providing a solution that transform end-user descriptions done with EUCalipTool into executable specifications. In particular, we are working on a generation module that transforms them into executable BPMN specifications. A preliminary version of the software architecture that support this solution can be found in [Mansaet et al. 2015]. In addition, we are investigating on applying a previous work focused on considerate computing. In [Gil 2013], we studied how to achieve a considerate interaction with users (i.e. disturbing them as less as possible) in the context of the IoT. We plan to apply this when executing composition created with EUCalipTool.

References

Athreya, B., Bahmani, F., Diede, A., & Scaffidi, C. (2012, September). End-user programmers on the loose: A study of programming on the phone for the phone. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on* (pp. 75-82). IEEE.

Atoma. (2015). Atoomam, a touch of magic. Accesible at: https://www.atooma.com/. Last time accessed: December 2015.

Ayora, C., Torres, V., Weber, B., Reichert, M., & Pelechano, V. (2013). Enhancing modeling and change support for process families through change patterns. *In Enterprise, Business-Process and Information Systems Modeling* (pp. 246-260). Springer Berlin Heidelberg.

BPDM (2014). Business Process Defintion Metamodel, volume ii: Process Definitions. http://www.omg.org/spec/BPDM/1.0/volume2/PDF

Casati, F. (1998). *Models, Semantics, and Formal Methods for the design of Workflows and their Exceptions*. PhD thesis, Milano.

Couper, M.P, Tourangeau, R., Conrad, F.G., Crawford, S.D. (2004). What they see is what we get: response options for web surveys. *Soc. Sci. Comput. Rev*., 22(1):111–127.

Cuccurullo, S., Francese, R., Risi, M., & Tortora, G. (2011). MicroApps development on mobile phones. In *End-User Development* (pp. 289-294). Springer Berlin Heidelberg.

Dadam, P., Reichert, M. (2009). The ADEPT project: a decade of research and development for robust and flexible process support. Computer Science - R&D, 23. 81-97

Danado, J., Davies, M., Ricca, P., & Fensel, A. (2010). An authoring tool for user generated mobile services. In *Future Internet-FIS 2010* (pp. 118-127). Springer Berlin Heidelberg.

Danado, J., & Paternò, F. (2014). Puzzle: A mobile application development environment using a jigsaw metaphor. Journal of Visual Languages & Computing, 25(4), 297-315.

Dey, A. K., Sohn, T., Streng, S., & Kodama, J. (2006). iCAP: Interactive prototyping of context-aware applications. In *Pervasive Computing* (pp. 254-271). Springer Berlin Heidelberg.

Engeström, Y., Miettinen, R., & Punamäki, R. L. (1999). *Perspectives on activity theory*. Cambridge University Press.

Galitz, W.O. (2002). The Essential Guide to User Interface Design: An Introduction to GUI. *Design Principles and Techniques*. *John Wiley Sons, Inc*. New York, NY, USA.

Gil, M., Serral, E., Valderas, P., & Pelechano, V. (2013). Designing for user attention: A method for supporting unobtrusive routine tasks. *Science of Computer Programming*, *78*(10), 1987-2008.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645-1660.

Häkkilä, J., Korpipää, P., Ronkainen, S., & Tuomela, U. (2005). Interaction and end-user programming with a context-aware mobile application. In *Human-Computer Interaction-INTERACT 2005* (pp. 927-937). Springer Berlin Heidelberg.

Haines, W., Gervasio, M., Spaulding, A., & Peintner, B. (2010). Recommendations for end-user development. In *ACM Workshop on User-Centric Evaluation of Recommender Systems and their Interfaces*.

ICIS. (2015). Internet Computing in the Internet of Services. Summer School. Department of Informatics Engineering of the University of Coimbra. Available at: http://icis.uc.pt/. Last time accessed: December 2015.

Ifttt. (2015). Ifttt, If This Then That. Accesible at: https://ifttt.com/. Last time accessed: December 2015.

Larman, C., & Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, (6), 47-56.

Lewis, J. R. (1995). IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, *7*(1), 57-78.

Liberman, H., Paternò, F., Klann, M., Wulf, V. (2006). End-user development: an emerging paradigm. *H. Liberman, F. Paternò, V. Wulf (Eds), End User Development*, Vol 9. 427-457.

Locale. (2015). Accesible at: http://www.twofortyfouram.com. Last time accessed: December 2015.

Lucci, G., & Paternò, F. (2014). Understanding end-user development of context-dependent applications in smartphones. In *Human-Centered Software Engineering* (pp. 182-198). Springer Berlin Heidelberg.

Mansanet, I., Torres, V., Valderas, P., Pelechano, V. (2014, september). A Mobile End-Use Tool for service Compositions. *X Jornadas de Ciencia e Ingeniería de Servicios (JCIS 2014)*, 25-35.

Mansanet, I., Torres, V., Valderas, P., Pelechano, V. (2015, september). IoT Compositions by and for the Crowd. *XI Jornadas de Ciencia e Ingeniería de Servicios (JCIS 2015)*.

Neil, T. (2014). *Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps*. "O'Reilly Media, Inc.".

Nielsen, J. (2005). Ten usability heuristics. https://www.nngroup.com/articles/ten-usability-heuristics. Last time accessed: February 2016.

Renger, M., Kolfschoten, G.L., & de Vreede, G.J. (2008). Challenges in collaborative modeling: A literature review. *In Advances in Enterprise Engineering I,* held at CAiSE 2008, Montpellier, France, Vol 10. 61–77.

Repenning, A., Ioannidou, A. (2006). What Makes End-User Development Tick? 13 Design Guidelines. End User Development. Human-Computer Interaction Series Vol 9. 51-85.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, *14*(2), 131-164.

Segal, J. (2005, May). Two principles of end-user software engineering research. In *ACM SIGSOFT Software Engineering Notes* (Vol. 30, No. 4, pp. 1-5). ACM.

Serral, E., Valderas, P., & Pelechano, V. (2013). Context-Adaptive Coordination of Pervasive Services by Interpreting Models during Runtime†.*The Computer Journal*, *56*(1), 87-114.

Tasker. (2015). Tasker, Total Automation for Android. Accesible at: https:// http://tasker.dinglisch.net/. Last time accessed: December 2015.

Uden, L., Valderas, P., & Pastor, O. (2008). An activity-theory-based model to analyse Web application requirements. *Information research*, *13*(2), 1.

Valderas, P., Pelechano, V., & Pastor, O. (2006). A transformational approach to produce web application prototypes from a web requirements model. *International Journal of Web Engineering and Technology*, 3(1), 4-42.

Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-Specific Languages: An Annotated Bibliography. Sigplan Notices, 35(6), 26-36

Van Welie, M., & Trætteberg, H. (2000, August). Interaction patterns in user interfaces. In *7th. Pattern Languages of Programs Conference* (pp. 13-16).

Weber, B., Reichert, M., Rinderle, S. (2008). Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. *Data and Knowledge Engineering*, 66. 438-466.

Yu, J., Sheng, Q. Z., Han, J., Wu, Y., & Liu, C. (2012). A semantically enhanced service repository for user-centric service discovery and management. *Data & Knowledge Engineering*, *72*, 202-218.