Springer

Dear Author,

Here are the proofs of your article.

- You can submit your corrections **online**, via **e-mail** or by **fax**.
- For **online** submission please insert your corrections in the online correction form. Always indicate the line number to which the correction refers.
- You can also insert your corrections in the proof PDF and **email** the annotated PDF.
- For fax submission, please ensure that your corrections are clearly legible. Use a fine black pen and write the correction in the margin, not too close to the edge of the page.
- Remember to note the **journal title**, **article number**, and **your name** when sending your response via e-mail or fax.
- **Check** the metadata sheet to make sure that the header information, especially author names and the corresponding affiliations are correctly shown.
- **Check** the questions that may have arisen during copy editing and insert your answers/ corrections.
- **Check** that the text is complete and that all figures, tables and their legends are included. Also check the accuracy of special characters, equations, and electronic supplementary material if applicable. If necessary refer to the *Edited manuscript*.
- The publication of inaccurate data such as dosages and units can have serious consequences. Please take particular care that all such details are correct.
- Please **do not** make changes that involve only matters of style. We have generally introduced forms that follow the journal's style.
  Substantial changes in content, e.g., new results, corrected values, title and authorship are not allowed without the approval of the responsible editor. In such a case, please contact the Editorial Office and return his/her consent together with the proof.
- If we do not receive your corrections **within 48 hours**, we will send you a reminder.
- Your article will be published **Online First** approximately one week after receipt of your corrected proofs. This is the **official first publication** citable with the DOI. **Further changes are, therefore, not possible.**
- The **printed version** will follow in a forthcoming issue.

Please note

After online publication, subscribers (personal/institutional) to this journal will have access to the complete article via the DOI using the URL: http://dx.doi.org/[DOI].
If you would like to know when your article has been published online, take advantage of our free alert service. For registration and further information go to: http://www.link.springer.com.

Due to the electronic nature of the procedure, the manuscript and the original figures will only be returned to you on special request. When you return your corrections, please inform us if you would like to have these documents returned.

Metadata of the article that will be visualized in OnlineFirst

ArticleTitle	A model-based framework for mobile apps customization through context-dependent rules
Article Sub-Title	
Article CopyRight	Springer-Verlag GmbH Germany, part of Springer Nature (This will be the copyright line in the final PDF)
Journal Name	Universal Access in the Information Society

Corresponding Author	Family Name	**Paternò**
	Particle	
	Given Name	**Fabio**
	Suffix	
	Division	
	Organization	CNR-ISTI, HIIS Laboratory
	Address	Via Moruzzi 1, 56124, Pisa, Italy
	Phone	
	Fax	
	Email	fabio.paterno@isti.cnr.it
	URL	
	ORCID	http://orcid.org/0000-0001-8355-6909
Author	Family Name	**Manca**
	Particle	
	Given Name	**Marco**
	Suffix	
	Division	
	Organization	CNR-ISTI, HIIS Laboratory
	Address	Via Moruzzi 1, 56124, Pisa, Italy
	Phone	
	Fax	
	Email	marco.manca@isti.cnr.it
	URL	
	ORCID	
Author	Family Name	**Santoro**
	Particle	
	Given Name	**Carmen**
	Suffix	
	Division	
	Organization	CNR-ISTI, HIIS Laboratory
	Address	Via Moruzzi 1, 56124, Pisa, Italy
	Phone	
	Fax	
	Email	carmen.santoro@isti.cnr.it
	URL	

	ORCID

Schedule	Received Revised Accepted

Abstract	The advent of the Internet of Things and mobile applications has made the possible contexts of use more and more varied, and creates new challenges for user interface developers. Although model-based approaches aim to support the generation of applications for different implementation technologies, limited attention has been paid to how to exploit them for novel context-dependent applications. We present a model-based framework that allows developers to flexibly customize their mobile apps to react to events not foreseen in the initial versions. It is composed of an authoring environment supporting the definition of model-based descriptions and generating mobile apps from them. The authoring environment allows developers to enrich the dynamic behaviour of the generated applications through trigger-action rules. The resulting versions of the apps can provide customized behaviour according to the actual contexts of use. The authoring environment supports efficient development of such customizations. We show its potential by describing a trial application, and report it on a first test with developers.

Keywords (separated by '-')	Authoring context-dependent applications - Model-based mobile app generation - Event–condition–action rules
Footnote Information	

Universal Access in the Information Society
https://doi.org/10.1007/s10209-018-0620-x

LONG PAPER

A model-based framework for mobile apps customization through context-dependent rules

Marco Manca[1] · Fabio Paternò[1] · Carmen Santoro[1]

Abstract
The advent of the Internet of Things and mobile applications has made the possible contexts of use more and more varied, and creates new challenges for user interface developers. Although model-based approaches aim to support the generation of applications for different implementation technologies, limited attention has been paid to how to exploit them for novel context-dependent applications. We present a model-based framework that allows developers to flexibly customize their mobile apps to react to events not foreseen in the initial versions. It is composed of an authoring environment supporting the definition of model-based descriptions and generating mobile apps from them. The authoring environment allows developers to enrich the dynamic behaviour of the generated applications through trigger-action rules. The resulting versions of the apps can provide customized behaviour according to the actual contexts of use. The authoring environment supports efficient development of such customizations. We show its potential by describing a trial application, and report it on a first test with developers.

Keywords Authoring context-dependent applications · Model-based mobile app generation · Event–condition–action rules

1 Introduction

We are witnessing an increasing availability of environments characterized by the presence of a multitude of connected objects and devices. In such environments, a variety of events can occur, triggered by changes in the state of such objects and devices, and by human behaviour. However, it is usually very difficult to foresee at design time the relevant events to consider and how to manage their occurrence appropriately. Thus, it becomes important to design novel environments that allow developers to rapidly customize the context-dependent behaviour of their applications in terms of how they should react to dynamic events occurring in the surrounding environments.

✉ Fabio Paternò
fabio.paterno@isti.cnr.it

Marco Manca
marco.manca@isti.cnr.it

Carmen Santoro
carmen.santoro@isti.cnr.it

[1] CNR-ISTI, HIIS Laboratory, Via Moruzzi 1, 56124 Pisa, Italy

Model-based approaches [1] aim to support user interface generation starting with some logical descriptions. In this way, it is possible to obtain implementations for various platforms with limited effort for developers. Some of such approaches exploit domain-specific languages [2, 3], namely languages designed to support a well-defined set of tasks. However, when addressing context-dependent applications, a model of the interactive application is not enough: a model of the possible contexts of use should also be provided along with ways to connect both models. In addition, there is a need for continuously updating such relationships to address how the application behaves when faced with new contexts or new situations that were not originally considered.

Nowadays, the need for more effective approaches for developing (and even evaluating and testing [4, 5]) mobile context-dependent Internet of Things (IoT)-based apps is pressing, especially in business environments, which are changing rapidly and where there are many organizations that need to deliver their products and services in a variety of business contexts. For instance, this is the case of companies managing chains of shops, where each shop presents specific aspects to consider in terms of size, products, users, or organizations managing several elderly assistance facilities, which have specific ways to monitor and support patients

Journal : Large 10209	Article No : 620	Pages : 17	MS Code : UAIS-D-17-00125	Dispatch : 1-6-2018

depending on their conditions. Developers of such applications must rapidly provide apps for different platforms (otherwise a good fraction of potential users could be lost) in a continuously growing device landscape [6]. Moreover, they also need to define, produce, and maintain different variants of the same application, with each version implementing a behaviour adjustment of the same application to the specific requirements, functionalities and software quality factors required by their clients operating in varying contexts. Typically, this often results in a laborious, costly and inefficient process where manual coding is the predominant approach, low-level configuration of heterogeneous hardware/sensors is needed, and cross-platform portability remains difficult. To cope with such issues, a number of model-based approaches and frameworks for cross-platform development of mobile applications have been put forward [3, 7], some of them addressing specific domains: for instance, MD^2 [8] and its enhancements [9–11] is aimed at business applications, while [12] is more oriented to be used in classrooms, for teaching application development on multiple platforms. However, current tools do not support context-based definition and configuration of such variants in an adequate manner, since generally each variant is defined from scratch and its development is kept separate. In addition, context-dependent adaptation decisions are typically hard-coded in applications, whereas a separation of concerns should be pursued.

The work presented in this paper aims to fill this gap, and consists of a method and a set of tools (including an authoring environment) to obtain mobile apps that can be rapidly updated by developers. One novel aspect of the proposed approach is that it allows developers to dynamically obtain context-dependent versions of mobile apps by specifying and executing rules that can be added any time without generating again the application. In particular, the contributions of this paper are:

- Model-based generator: A model-based environment, able to generate native mobile apps;
- Language for rules: A language for defining relevant event–condition–action rules, for specifying context-dependent adaptation rules;
- Support for personalisation: An authoring environment able to support developers to specify model-based descriptions of interactive applications, and associated rules that customize their context-aware behaviour.

In the paper, following the Introduction, we analyse relevant work in the state of the art. Then, we describe the presented approach by specifying a relevant scenario, highlighting the original aspects of the presented solution and describing its relevant architectural modules. The next section provides information about the adaptation rules considered in the approach. Then, the proposed Context-dependent applications Authoring Tool (CAT) is described, as well as the approach followed for generating android-based applications from model-based descriptions. Then, we describe how the approach proposed manages contextual rules, and afterwards present an example application that shows its potential. Finally, we report on a test with some developers, and conclude the paper by providing closing remarks.

2 Related work

The increasing availability of devices and sensors calls for authoring tools and frameworks able to support developers in exploiting such opportunities. For example, Nebeling has proposed authoring environments [13] able to facilitate the development of cross-device user interfaces taking into account relevant patterns of interactions. GlueTK [14] was a C++ framework for developing cross-device user interfaces able to exploit various interaction modalities. However, neither of these contributions addressed how to enable such applications to react and adapt to dynamic events detected by sensors associated with connected objects or environmental aspects.

The success of recent services such as IFTTT (http://ifttt.com/) demonstrates the emerging need for ways to customize the available applications to react to various contextual situations. However, IFTTT is limited in terms of expressiveness, because it allows users to create rules containing only one trigger and one associated action, whereas recent studies [15] have shown that even end users can manipulate more structured rules. Moreover, IFTTT only allows end users to connect events and actions that are already supported in a set of predefined applications, while our approach seeks to enable developers to efficiently modify their applications to better react to various combinations of contextual events and conditions. AppsGate [16] aims to support non-expert users to program their smart environments. In our case, we aim to provide an authoring environment intended for use by developers, since it requires some knowledge in modelling and programming (thus it is not for end user development environment). In addition, compared to [16], our solution can be exploited in various applications domains (not only the home). Another work [17] has proposed a meta-design approach for personalization of Internet of Things applications in various domains, though it targets end users without programming experience, thus narrowing the complexity of the customizations that can be achieved. For instance, in [17], only a limited number of types of UI modifications are supported, whereas in our approach every attribute of the elements belonging to a UI specification can be referred by a rule.

Some previous work has addressed the design of context-dependent applications. For example, iCAP [18] is a system that allows designers to specify applications in terms of a set of rule-based conditions. Unfortunately, such rules are often insufficient to obtain complete application descriptions, and iCAP did not address user interface modifications, since it mainly supported controlling surrounding appliances. One approach to obtain customization through trigger-action rules was put forward in [19], but it was an authoring environment limited to web applications, did not use a model-based language and, since it was based on web browser extensions, it was not able to address native apps (which are often preferred because they perform more efficiently). In addition, that approach was able to address a limited set of contextual aspects.

Model-based approaches aim to support designers and developers to concentrate on the main aspects of application design by hiding implementation details through conceptual descriptions. One important contribution in this area was Supple [20], which provided a tool able to consider aspects related to the user and the device at hand for generating a personalised version of the interactive application. Other researchers [21] showed how to generate versions optimized for mobile devices. However, the combined explosion of mobile technologies and the Internet of Things has limited the impact of such contributions, since they are not able to address the variety of contextual aspects to consider for customizing interactive applications. More recently, a generative approach with semantic interaction descriptions to obtain UIs for smart things was proposed [22], though again it assumed static configurations of the smart things to address and thus was unable to support customization in dynamic contexts of use.

In [23], the authors address the problem of developing and maintaining multiple native variants of mobile applications. To this goal, they present a model-driven engineering approach for automated generation of feature-based application variants for multiple mobile platforms. The approach allows developers to generate native mobile application variants covering: (1) variations due to operation systems and their versions; (2) variations due to software and hardware capabilities of mobile devices; and (3) variations based on the functional requirements of a mobile application. However, compared to our solution, this approach has less explicit consideration of the development/customisation of cross-platform applications leveraging the opportunities offered by the IoT world (in terms of modelling behaviour that can dynamically change according to the occurrence of events and conditions in the context).

Other authors have focused more on how model-based UI development approaches can be exploited to address certain aspects related to the experience of users interacting with ubiquitous applications. Recent contributions [24] presented a UI development framework for ambient applications integrated with a user modelling system, to provide usability predictions during early development stages. Other authors [25] addressed the issue of ensuring consistency of user experience across different and heterogeneous devices and platforms. To this aim, they introduce a generative design pattern-based approach for cross-device services. However, as the authors themselves acknowledge, the specified approach is able to address only limited contextual variability. In our work, we aim to overcome such limitations, still using a model-based approach. We aim to obtain a more general solution able to also address native apps, cover a broad set of contextual aspects, support the composition of events, conditions, and actions, and access the state of appliances.

In other model-based approaches [26], the focus was mainly on the reusability of UI components/fragments, based on the idea that the work of developers could be highly simplified with automatic UI composer tools able to build new applications by reusing parts of existing ones. However, that approach mainly focuses on constructing standalone Java-based UIs, and provides limited support of context-dependent aspects relevant for customizing applications by developers. To this regard, a more relevant contribution is [27], which presents so-called "self-adaptive UIs", allowing run-time UI adaptation obtained by an automatic reaction to context of use changes. In that approach, a model-driven development of UIs (based on IFML) is coupled with a separate model-driven development of UI adaptation rules and context of use called AdaptUI. However, in that work the opportunities for adaptation are mainly limited to UI-related changes, whereas in our approach a more comprehensive set of actions is provided, including those controlling devices, objects and appliances available in the current context of use.

3 Approach

3.1 Motivating scenario

The need for an application to operate in different contexts of use, and thus require different versions to better address them is important in various application domains. An example can be found in the retail domain, in which the same retail application can be deployed for radically different contexts of use ranging, for example, from large supermarkets in big shopping malls to smaller stores in town quarters. The latter ones are generally located in older buildings and have a limited capability of the underlying technological infrastructure, whereas large stores are in new buildings, with highly technological infrastructures that allow easier integration of new sensors and devices. Thus, the same application associated with the two types of shops has to handle radically

Journal : Large 10209 | Article No : 620 | Pages : 17 | MS Code : UAIS-D-17-00125 | Dispatch : 1-6-2018

different requirements. For instance, due to the small number of sensors available in the shop, the owner of the small shop is not able to offer a lot of functionalities in the associated customers' application. On the contrary, owners of big markets would aim to offer a high quality mobile shopping experience exploiting advanced technological infrastructure and customized services (e.g., dynamic personalisation of prices and offers based on customer profiles and their actual behaviour while moving in the shop), also to make the shopping experience more efficient (e.g., real-time directions for locating hard-to-find products, mechanisms to avoid queues). In addition, in both cases shop managers may like to introduce some more specific customization rules to the app, based on observation of actual clients' behaviour. From this small example, it is clear how the same application can differ in terms of events to recognise and associated context-based functionalities offered, which can vary depending on resources available, types of users, and other relevant contextual factors.

However, handling different yet closely related implementations could be difficult and time-consuming. What typically happens currently is that developers that have to build an application for a retail brand that has both big stores and small ones would end up developing and maintaining two different versions, having a common set of core functionalities typically associated with the considered domain (i.e., retail), and further characterised by specific functionalities requested by the particular shop. For instance, on the one hand, within the version targeting the big supermarket, there will be functionalities devoted to smart management of discounts and promotions to offer to users, and other functionalities for supporting customers in the mall (thanks to the technological infrastructure available in the big supermarket). On the other hand, the version targeting the small shop would not provide any specific support, for example, assisting customers while they are in the shop, since no particular sensing technology is likely to be available. Then, it would only include, e.g., personalised lists of daily and weekly offers (possibly customized taking into account customer profiles, their latest purchases, seasonal aspects, etc.).

The traditional development path on its own would not be sufficient to support adaptive capabilities, that is to say, support functionalities that need to change their behaviour over time, in a context-dependent manner. Thus, we need mechanisms not only allowing for easily managing different application versions in a consistent manner, but also supporting developers to enable the application to easily evolve and adapt depending on changing requirements and needs of users, by dynamically adding new rules or modifying the existing ones, anytime. This is different from traditional approaches, which tend to handle context-dependent behaviour by hard-coding it in the implementation code from the beginning.

3.2 Novel aspects of the solution

For describing interactive applications, we have used the MARIA language [28], which supports various abstraction levels for specifying a user interface. MARIA has one language for the abstract description, the so-called "Abstract User Interface" (AUI) level, in which the UI is described in a platform-independent manner, and multiple Concrete User Interface (CUI) languages, which support specifications of a User Interface for a specific platform independent of implementation languages. Such CUIs are obtained as refinements of the abstract one depending on the interaction resources of the target platform. Examples of platforms include the graphical desktop, the graphical mobile, and the vocal. For this work, we chose to use MARIA because its specification is publicly available, and there are already generators for web-based implementations. Furthermore, a tool (MARIAE [29]) supporting MARIA language is publicly available, which facilitates the specification of MARIA-based UIs.

The MARIA environment also provides a tool supporting AQ2 a reverse engineering process: it takes in input of an existing web application and returns the correspondent Abstract and Desktop Concrete UI description. Thus, through an automatic generation it is possible to obtain a native mobile version of an existing web application. We have developed a new generator for native android apps (from this work a generator for other types of native apps, such as iOS, can be obtained).

To have a clear separation of concerns (which facilitates the development of customized versions), in the presented approach the management of contextual aspects is defined in trigger-action rules, specified separately from the application description, even if the effects on the applications refer to MARIA elements. The only component related to the rules that the generator should integrate in the final application is the action interpreter, which is a component able to receive the actions, and apply them. The authoring environment has been designed to facilitate this integration. The novelty of this environment is not only the model-based generation of android apps (which has not been trivial since the android application model is quite different from the web application model). Another novel aspect is the integration of model-based descriptions of user interfaces with rules specified in terms of event–conditions–actions. In such rules, events and conditions refer to the context model, whereas actions can affect devices, objects, and appliances deployed in the context and also user interface elements defined in the model-based description. In this way we obtain an original solution for customizing the behaviour of context-dependent apps, which is more general than approaches such as iCAP [18] and more flexible than previous model-based approaches. The authoring environment supports compositions of triggers; thus, it is possible to specify expressions

Journal : Large 10209 | Article No : 620 | Pages : 17 | MS Code : UAIS-D-17-00125 | Dispatch : 1-6-2018

able to combine different context aspects, which cause the execution of a customization rule. This solution enables the implementation of applications personalised according to indications that can be provided by domain experts (e.g., shop managers, caregivers, etc.).

3.3 Method

In the proposed solution (Fig. 1), designers use the CAT to build first the MARIA specification of the application, and then the relevant adaptation rules that will manage the context-dependent customization aspects. Such rules are expressed in event–condition–action format. The MARIA specification is a model-based description of the application from which various generators can derive an implementation of the application for different platforms. The rules are defined by first indicating relevant events and/or conditions referring to the contextual entities included in the context model (which provides a logical specification of the entities that characterise the context) shown by the tool, then indicating the actions to perform on the elements of the relevant MARIA specification. Such actions can change the user interface of the generated application or generate specific effects (e.g., reminders or alarms) or change the state of the appliances managed by the application.

At run-time the generated mobile application first subscribes to the server-side module called adaptation engine (AE) so that the latter can get from the adaptation rule repository the rules (defined through the authoring environment) associated with the current application and a specific user. The code responsible for the subscription is also produced during the app generation. Such rules are then used by the AE to subscribe to the context manager to be notified whenever any event/condition involved in such rules is verified in the current context. As soon as an event occurs or a condition is verified, the context manager notifies the AE. The AE then checks whether there is any adaptation rule associated with the current application and user containing such event or condition, which could be triggered. In this case, the AE extracts the list of actions specified in such set of rules and sends them (in an XML-based format) to the mobile application, so that they can be interpreted and executed for customization goals. Indeed, applications include an interpreter (which is automatically generated when the context-dependent app is created) that is able to interpret and apply actions written in such format.

Thus, the AE and the context manager are pre-existing software modules, which compose the personalization platform. The AE acts as a repository of the rules and activates adaptations rather than directly carrying them out. Moreover, it is able to select which rule activate when multiple rules are triggered through a priority mechanism. It is also connected with the context manager, a software module (customizable for different environments) able to gather information from various sensors and devices and to notify other architectural components about the occurrence of relevant events or conditions (e.g., the user enters home). When an event occurs or a condition is verified, the context manager informs the AE, which identifies relevant adaptation actions. Such actions can determine updates in the interactive application (e.g.,

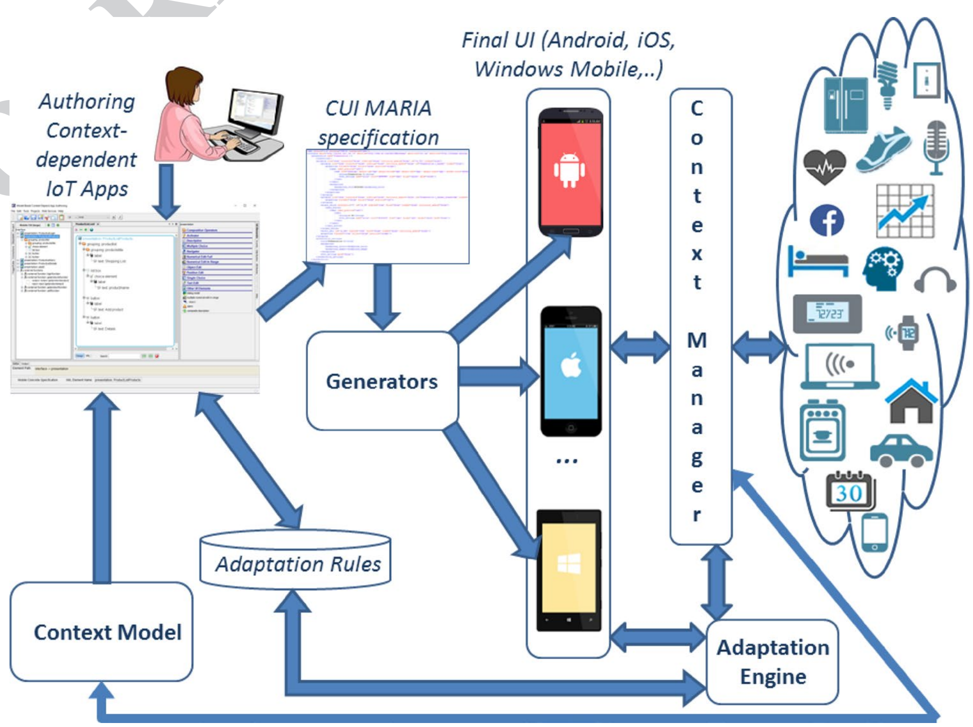


Fig. 1 Overview of the approach (design and run-time)

reorganize the layout, activate functionalities, etc.) as well as change the state of appliances available in the current context of use (e.g., switch on the radio, change light intensity). Both the AE and the context manager are deployed in the same host for the sake of efficiency. The reason why the AE is in a server and not in the mobile device is that it frequently has to interact with the context manager, and has to manage rules that can refer to different applications or application versions, or even to different users of the same application. Indeed, the rules created through the authoring tool can be associated to a specific application and a specific user.

The context manager provides functionalities to collect data from external contextual sources, such as sensors (temperature, noise, light, doors/windows closure, power absorption, etc.) or external services (e.g., weather forecast). It uniformly integrates such heterogeneous data in a common format, stores them, and makes them available to other architectural modules. The context manager is composed of one centralised module (context server) and a set of modules (context delegates), which can be distributed on various devices. The context delegates are software modules developed for handling data coming from associated sensors. In particular, they are tiny pieces of software able to connect to the concerned sensor, read the detected raw data and translate such data according to a homogeneous format able to abstract out from the heterogeneity of the specifications produced by the various sensors. Such data are sent to the context server which will update relevant data structures accordingly, to keep the state of the context updated. The context delegates can be deployed in various manners. For instance, the context delegate for monitoring the state of a smart lighting system can be deployed in the bridge that manages the associated lights. A delegate that collects temperature, humidity and motion values and sends them to the context server can be executed in an Arduino board. A smartphone can host a delegate that detects environmental noise by interfacing with the device's microphone.

Figure 1 provides an abstract description of the overall approach. Figures 5 and 6 provide more detail for the design and run-time phases, respectively. The context model is structured in a hierarchical manner, which makes it well-suited to easily define customization rules based on relevant contextual events/conditions. At the highest level, the context model includes four main contextual dimensions: user, environment, technology and social aspects. Each dimension is further refined into a number of sub-categories. For instance, the environment dimension (which provides general information on the surroundings) includes information concerning its type (e.g., indoor/outdoor), name, spatial aspects (e.g., size, shape, and position), ambient conditions (further refined into temperature, light level, noise, humidity, motion, presence, time, and weather), and things/appliances available in the considered environment. As the context model is described by a XML schema (XSD), the context manager can directly manipulate JAVA classes that are automatically derived from such XSD.

By traversing the hierarchy of the context model (from the highest level to the lowest level), it is possible to define a 'semantic' path to the specific contextual aspect developers aim to refer to. A new sensor can be easily integrated in the platform by simply defining a new context delegate associated with it. If a new contextual attribute is not defined in the context model, the latter can be modified to include and describe the data sensed by the new sensor. Then, the associated context delegate can send to the context server the sensed data value and the path in the context model hierarchy corresponding to the attribute, so that the context server can update the relevant data structures accordingly. For instance, in the complex condition IF < living room lamp is off > AND < light level in the living room is low>, two attributes associated with entities defined in the context model are referred. The corresponding paths used to identify the contextual resources referred by the rule are, respectively: technology/physicalObject/hue-lamp-livingroom/@state and environment/livingroom/@light_level. As it can be seen from Fig. 1, the context manager shares the context model with the authoring tool. As it will be further described in the following sections, the context model can be imported into the authoring tool so that developers, while defining a certain trigger and its properties, can easily select the possible events and conditions referring to the imported context model.

4 Adaptation rules

Adaptation rules are expressed through an ECA-based (event, condition, action) language, where events are instantaneous changes of the context state, conditions are Boolean predicates referring to states of contextual entities, and actions are changes to be applied either to the interactive application or to the state of appliances managed by the target application, or they involve the activation of external functionalities. It is worth noting that actions are application-dependent: since it is the application that interprets and executes them, the actions should be tailored to the application. An example rule is "IF presence is detected in the living room AND the light level of the living room is low, THEN switch on the living room light that is on the table". The meta-model of the language for specifying adaptation rules is shown in Fig. 2. A rule model is composed of one or more rules. Each rule can have one event part, one condition part, and one or more actions.

The event part can be either an elementary event, or a composition of events obtained by applying a 1-ary or an *n*-ary operator to event elements. Events compositions

Journal : Large 10209 | Article No : 620 | Pages : 17 | MS Code : UAIS-D-17-00125 | Dispatch : 1-6-2018

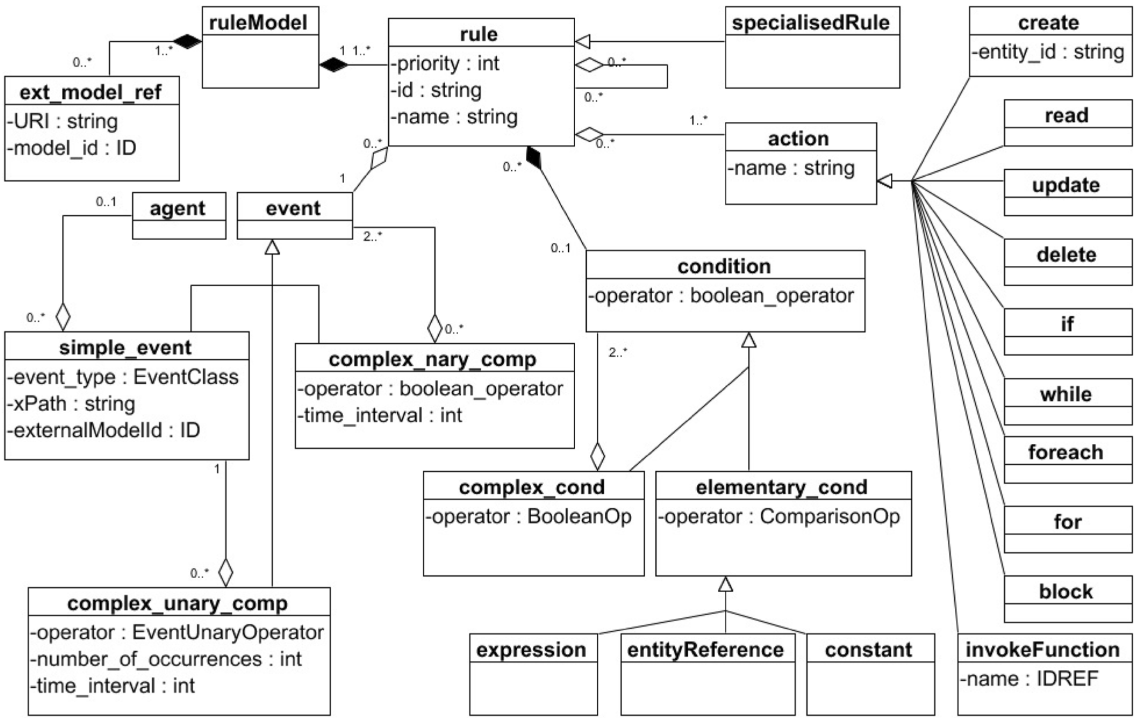


Fig. 2 The meta-model of the language for the rules

could also (optionally) specify a time interval in which the involved events should occur. A simple event is characterised by a name, and a string containing the path identifying the corresponding entity within the context model. Conditions can be elementary Boolean expressions (e.g., noise < 50), or complex ones [e.g., (noise < 50) AND (light = high)]. The specification of elementary conditions implies the specification of one of the following elements: a context entity reference, a constant, an expression (to be used when the referred entity is not directly available but is the result of a calculation). The specification of complex conditions involves multiple elementary conditions and the use of Boolean operators. A rule can specify one or more actions. There are various action types: some elementary actions (create, read, update, delete) and a number of well-known constructs to combine them such as if, while, for each, for, block (a sequence of actions that can be named for further reference). The invoke function is used to access an existing service which might be connected to available appliances. Other actions (update, create, delete) are aimed at customizing the UI (in terms of presentation aspect or content), while other actions (update, invoke function) are directed to change the state of appliances managed by the application. To obtain UI modifications, rules refer to the MARIA Concrete UI (CUI) specification, to indicate the actual elements to modify in the UI. A rule can also activate other rules and it is also possible to define specialized rules that are defined for one specific application. In this approach, conflicting rules are those that would require some appliances to be in different states at the same time (e.g., one rule asking to switch on a lamp in the morning and another one asking to turn it off), or rules that refer to the same UI element and, while the first one hides the element the other one renders it. Our environment is able to analyse the rules, detect whether they have conflicting actions, check if such actions can be executed at the same time, and, in the positive case, highlight the involved issues. Rules can also have priorities, useful when multiple, conflicting rules occur simultaneously.

In addition, the system is able to handle dynamic and rather unexpected situations that can occur during the development and could potentially lead to inconsistent states. For instance, if a developer modifies the MARIA specification and a rule refers to a UI element that no longer exists as a consequence of that previous change, the authoring environment is able to detect this and inform the developer about this inconsistency so that she can act upon it.

Another case is when a rule refers to a device or to an appliance which is no longer available at run-time in the current context. In this case, the rule will not be activated and, as a consequence, the associated action will not be applied and executed. Such an unexpected situation is identified by the context manager which has the overview of all the sensors, appliances and devices which are actually active in the current context, by means of receiving regular updates from

Journal : Large 10209 | Article No : 620 | Pages : 17 | MS Code : UAIS-D-17-00125 | Dispatch : 1-6-2018

the associated context delegate. If no update associated with a particular context entity is received by the context server within a specific threshold, that element will be considered no more active. The structure of the adaptation rule is specified in a XSD schema. Using the CAT tool (described in the next section) the adaptation rules can be easily edited and then saved in XML language for facilitating their sharing.

5 The CAT authoring environment

The CAT authoring environment has been developed to support the editing of the user interface specification and the associated event–condition–action rules. The part concerning the authoring of the MARIA specification of the interactive application is similar to [29]. We focus on the novel part that allows developers to specify event–condition–action rules with references to a context model (for indicating events and conditions), and to the application specification (for indicating possible actions which modify the UI aspects). Figure 3 shows the authoring environment, which is organized in various panels. In the left panel, there is the list of the rules defined for the current application. In the central panel (the rule editor), there is the presentation of the rule which is currently edited. The right panel presents some lateral tabs that allow the designer to change dynamically its content. In particular, the tabs associated with the right panel are: rule elements, which shows the allowed children of the rule element currently selected in the central panel, following the structure and the constraints defined in the meta-model described in the previous section. Thus, it is possible to define a rule by dragging an element from the right panel and dropping it in the central panel; a double-click on an element currently visualised in the central panel will open a dialog box with the list of editable attributes; CUI (the tab currently selected in Fig. 3, which stands for “Concrete User Interface”), which shows the interactive tree-like representation of the MARIA specification of the interactive application that the current rule refers to, so that developers can easily refer to elements of the associated UI while editing a rule; rule element attributes, for defining the attributes of various rule elements; context model, showing the elements of the context model visualised according to its hierarchical structure.

The “CUI” tab content is useful when developers need to create a rule in which the included actions affect an element of the user interface. In such cases the developer can select a user interface element from this tree-like specification and, using drag & drop, include a reference to the concerned CUI element into the specification of a rule action.

A similar approach has also been used to enable designers to easily specify rule events and conditions. Since events and conditions both refer to contextual aspects, their definition will refer to context entities contained in the context model, which can be imported into the Authoring tool. In particular, the user can select the “context model” tab of the right panel of the tool (which shows the context model), and then drag & drop the reference to a specific contextual entity or attribute of interest into the specification of the event/condition part of the rule currently shown in the central panel. A video showing how to interact with the authoring tool is available at http://youtu.be/UbGnfmA4MlA.

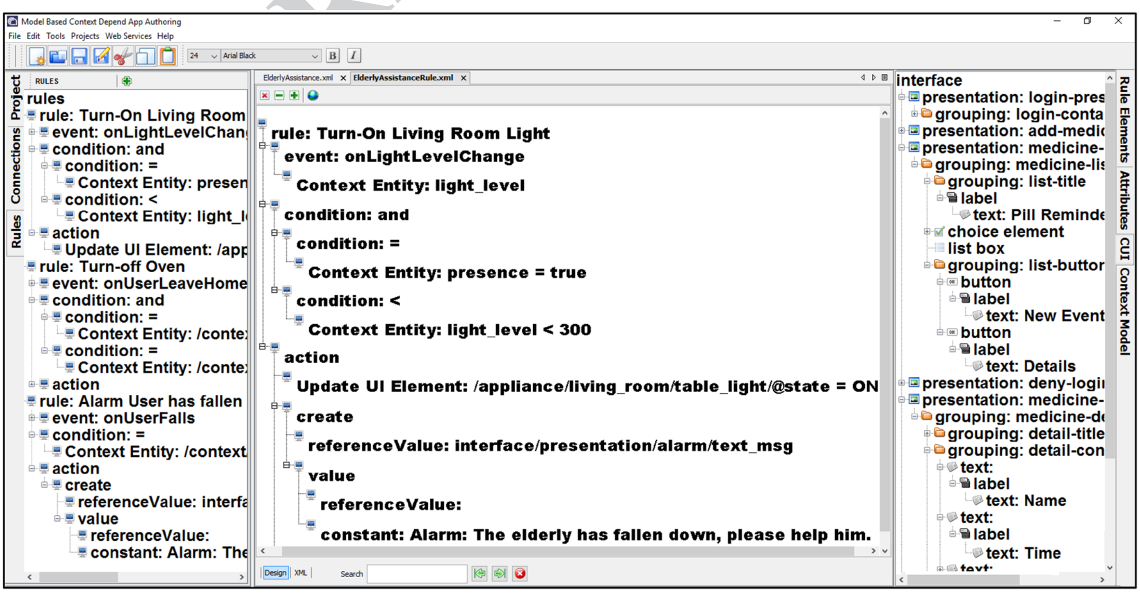


Fig. 3 The CAT authoring environment

6 From MARIA to android implementation

The MARIAE editor already provided generators for a number of implementation languages (HTML5 + JavaScript, VoiceXML, X + V). However, the generation of android native mobile apps from MARIA specifications has not been addressed before. Here we report on how a generator for android apps has been designed and implemented. To obtain a native app for another platform, there is no need to re-define the MARIA specification; it would simply require another generator which takes as input the same description and produces code for the target platform (e.g., iOS).

The environment in which android apps are executed is rather different from the one used by web ones. Indeed, android apps maintain their state even if they are no longer visible, while in web applications this has to be managed explicitly by programmers. Developing an android application is based on two main concepts: activities and services. Activities are elements that require direct interaction with the user. Services are applications that work independently and can be started by activities if they are needed. In android apps, there are no links because the navigation is managed through dynamic activation and termination of the activities. Activities contain a GroupView object that defines the user interface, and which can contain several elements of the same type (view elements), which represent the graphical widgets. The GroupView is commonly called layout. An activity can also contain one or more fragments. A fragment represents a portion of the user interface in an activity and it can be used by one or more activities, while an activity can use more than one fragment.

The generation process recursively analyses the MARIA user interface elements and creates the android implementation. For this purpose, the generator uses intermediate representations (wrappers), which are classes that implement a specific Java interface, which in our case consists of a method to handle the generation of Java code. Each wrapper also contains the information needed to represent the UI element (e.g., the button wrapper will contain the text, the size, the text colour, the background colour, etc.). The wrappers provide an intermediate representation of the android elements aimed to maintain the generator code more readable and modular. The generator takes a MARIA file as input, interprets it, and transforms it to an android-based counterpart, based on activities, layouts and views. The generator analyses each MARIA element starting from the root. If it is elementary, the result is the corresponding android wrapper; otherwise (i.e., it is a complex element) the generator is called recursively on the elements contained within it. In particular, the generator starts by analysing the initial element of the MARIA file ('interface' element). From this element, it obtains information about the name of the project, the external functions and the data model (a data structure storing the values of all UI elements). Then, it creates a wrapper containing such information, as well as the manifest file and the files needed by an android compatible IDE. Then, the generator continues to analyse the interface children, which are the presentation elements. Each of them represents a view for the android interface, and the presentation is translated into a concrete android activity equipped with one or more fragments (this number depends on the number of presentation children). After having handled the presentation element, the generator begins to analyse recursively the child elements starting from the first grouping element. The generator applies the recursion only to complex elements (groupings, repeaters and relations). In these cases, it generates a Layout wrapper and reiterates over the children to generate the respective wrappers and put them inside the previously generated Layout. The generator analyses the properties of the MARIA elements and adds them to the wrapper: for example, a grouping can have a title, a background colour, a width, etc. Simple interface elements typically do not need a layout but are objects supported by android.

In UI specifications, it may happen that one element appears multiple times in different parts of the specification: this can introduce errors and can be hard to maintain. To avoid such errors, we decided to allow developers to reuse portions of user interfaces defined in other presentations. For this reason, we added the "ref" (reference) attribute to the composition of elements: this attribute informs the generator that the considered composition refers to another one, and the generator should refer to that definition. The example in Fig. 4 shows a UI containing two different presentations. Presentation 1 is a login view with a grouping element (grouping 1), which includes three different children elements: a text box (TextEdit 1) where the user can type

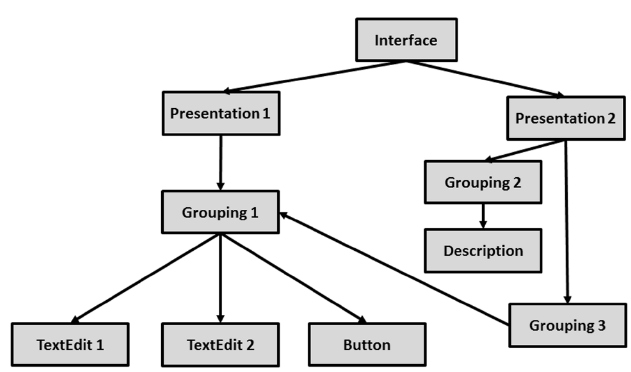


Fig. 4 The logical structure of an example user interface

Journal : Large 10209 | Article No : 620 | Pages : 17 | MS Code : UAIS-D-17-00125 | Dispatch : 1-6-2018

the username information, a text box (TextEdit 2) where the user can type the password, and a button that, when selected, makes the application check the login and the password by sending this information to an external function. When the function returns the login response and the credentials are not valid, presentation 2 is shown.

This second presentation has two different grouping elements: the first one (grouping 2) contains an error message (description) to show that the login was not successful, and the second (grouping 3) should contain the same text boxes and button as presentation 1 to re-insert the login credentials. Thus, grouping 3 refers to grouping 1. In the structure of the android application corresponding to this MARIA example the presentations are mapped into two activities, and even if there are three groupings, only two fragments will be generated, because grouping 3 refers to grouping 1 and the fragment generated from grouping 1 is used (through a reference) for both activities. As already mentioned, an application can call functions defined externally (e.g., the login function of the previous example). Such functions can return values representing the results of their execution, and these values can be used in different presentations of the same interface. For this reason, the MARIA specification includes the definition of a data model used to store the user interface state, and defined through a specific XML schema definition (XSD). An external function includes input and output parameters. The input parameters values can be taken either from a user interface element or from the data model. Once the external function has been executed, the generated values are stored in the data model so that one or more UI elements can present them.

The generated implementation manages the data model through a Java Class shared amongst all the activities of the application. To implement external functions, the generator creates a class called AsyncFunctionHandler to call external functions without locking the interface. This class uses asynchronous threads, and the functions are called from a thread different from the main thread that handles the UI. When the code generation is completed, the generator creates the scripts associated with the MARIA elements (activators) that perform the activation of the external functions. To do so, the generator creates a handler for a specific event (e.g., the click of a button), which performs an asynchronous call to the web server that contains the function. The asynchronous thread waits for the response from the web server and inserts the result inside the data model (as specified in the MARIA description). Finally, the generator creates the connections between the activities. Connections can be elementary or conditional: an elementary connection is activated when the user interacts with a navigator element, and starts a new activity. Conditional connections model cases when switching from one presentation to another one is triggered only if a specified condition is verified; the condition may refer to a value of either a UI element or an element stored in the data model. Conditional connections are only executed after applying changes to the data model made by external functions to have it updated before making decisions on which presentation to load. The generated code of a conditional connection starts a new activity on the screen if the condition is verified, while a simple connection starts the activity without checking anything.

7 Contextual rules management

The generated applications are context-dependent, which means that they can change their dynamic behaviour depending on contextual events or conditions described by rules defined through the CAT authoring tool. Figure 5 shows what happens at design time, mainly focusing on the generation process, where the authoring environment supports the editing of both the MARIA specification and the rules for the application. The MARIA specification is handled by the android generator to generate the corresponding android application, while the event–condition–action rules are then sent to the AE, which stores them for use at run-time. To manage the contextual customization, the generator includes in the application a class called AsyncTaskUpdater. This class extends the AsyncTask interface that defines asynchronous threads for android applications.

At run-time, this class will open a WebSocket channel to the AE and will subscribe to it sending its credentials (which include the app name and the user name), to receive the actions to support the application customisation needed for the specified user when a rule is triggered (see Fig. 6, which better details what happens at run-time).

The AE loads the rules associated with the application, and subscribes to the context server to be informed when an event occurs or when a condition is satisfied. When a sensor changes its value, the correspondent context delegate sends it to the context server, which updates its internal representation of the context, and checks whether such update could trigger the execution of a rule. If this happens, the context server communicates the verified events/conditions to the AE which retrieves the actions described in the correspondent triggered rules and sends them to the application through the web socket channel opened before. Indeed, the AE provides applications with the actions to carry out specified through a XML-based rule language, which can refer to MARIA elements (e.g., an action may require to update an UI element and hence its id should be indicated). The generator also includes an action interpreter in the application. Such interpreter is able to receive the action specifications, transform the MARIA identifiers included in them into references to the corresponding android objects (the correspondences are indicated in a table created during the generation

Journal : Large 10209 | Article No : 620 | Pages : 17 | MS Code : UAIS-D-17-00125 | Dispatch : 1-6-2018

Fig. 5 The generation of context-dependent apps

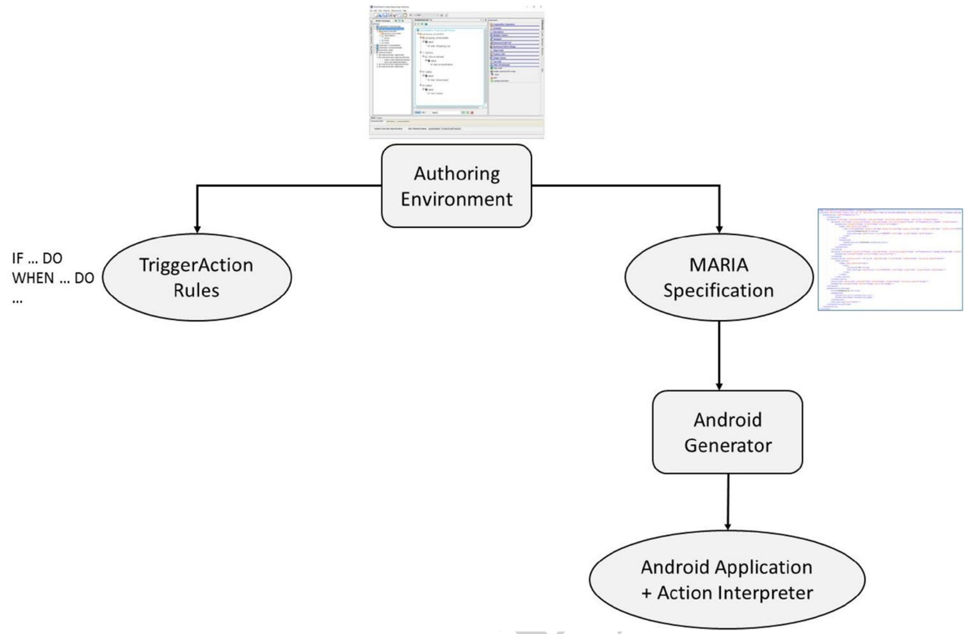
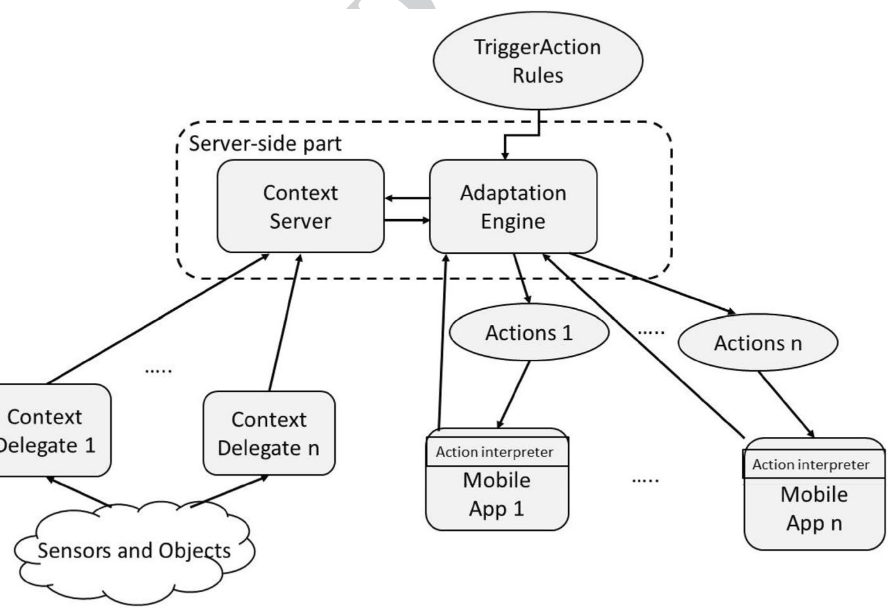


Fig. 6 The run-time communication



phase), and execute them. This computation may also be applied on UI parts not currently visible, since they belong to a different activity. In this case, all the actions received are saved, and then analysed when a new activity is started.

8 A trial application

In this section, we consider a trial application. The case study considers an organisation managing different types of residential facilities for persons 60 years old or more. In particular, the organisation manages various types of supportive housing options. Among all, it supports both "assisted living" facilities providing independent living opportunities such as self-contained private apartments, and a medium-scale residential centre in which several elderly people live. Thus, the organisation needs an application allowing them to manage two rather different settings, since the two options greatly differ in terms of services offered, technological support available, environmental conditions, needs and profiles of target users.

Indeed, on the one hand, the residential centre provides 24-h care by skilled professional staff, as well as meals, housekeeping, supervision, storage and distribution of medication, and even personal care assistance with basic activities such as hygiene, dressing, eating, bathing and transferring. The facility is composed of several private bedrooms (whose occupancy is limited to a maximum of two residents per bedroom) and some shared areas: residents can share a bathroom with other residents, and common spaces where residents can gather with each other or with visitors to socialize and recreate. Access to the facility (entrance/exit) is controlled by the handling organisation.

On the other hand, private apartments (ranging in size from small studio units to two bedroom units) have private bathrooms, kitchenettes and locking door, and offer easy access to outdoor areas and gardens. Differently from residents in the residential care centre, residents of private apartments can set their own schedules for when they want to wake up, eat meals and go to bed. They are also able to enjoy customized, cooked-on-premises meals instead of being restricted to a fixed menu.

The two contexts also differ in terms of technological support offered for both monitoring conditions of residents (e.g., the set of sensors installed) and improving their life/facilitating their living (e.g., the actuators available). On the one hand, some people living in residential facilities wear the Plux Bitalino [30] chest band, which gathers from various sensors (e.g., ECG, accelerometer, temperature) data relevant for monitoring their health conditions. In addition, still within the residential facility, the organisation is also experimenting light therapy to investigate its effects on cognition, sleep, circadian rhythms and depression on their residents. In this setting, the organisation would like to have an application effectively managing all such technological support, monitoring and assisting the elderly, and even informing the staff when some anomalous situation is detected.

On the other hand, people living in private apartments are considered more autonomous than the ones living in the shared facility. As such, in private flats, the managing organisation is more focused on having an application which facilitates user themselves in managing their daily routines. Thus, in the setting of private apartments, the application should rather support users' monitoring and controlling their home environments through a (typically small) set of basic sensors/actuators (e.g., lights, thermostats, air conditioning, motion), help users in managing typical routine activities (e.g., reminders for taking medicines), even persuading them in doing specific activities (e.g., doing physical exercise).

Figure 7 shows an example rule (R1) related to a customization involving a dynamic change of the user interface: "If the light level is less than 200 lx, then increase the fonts (by 10%) and set to black the colour of elements representing the medicines the user has to take within the next 10 min".

It shows an excerpt of the underlying specification of this rule obtained through the authoring tool. If the light level is under a specified threshold, the action iterates over the

```
<xml version="1.0" encoding="UTF-8">
<ruleModel>
  <rule name="Rule 1">
    <condition operator="lt">
      <entityReference xPath="environment/@light_level"/>
      <constant type="int" value="200"/>
    </condition>
    <action>
      <foreach>
        <in><entityReference xPath="dataModel:medicineList"/></in>
        <do>
          <if>
            <condition operator="lt">
              <entityReference xPath="./@intake_time"/>
              <contant value="current_time() + 10"/>
            </condition>
            <then>
              <update>
                  <entityReference
              xPath="uiElement:[id='./@medicine_name']/@font-size"/>
                  <value>
                    <contant value="+10%"/>
                  </value>
              </update>
              <update>
                <entityReference
              xPath="uiElement:[id='./@medicine_name']/@font-color"/>
                <value><contant value="black"/></value>
              </update>
            </then>
          </if>
        </do>
      </foreach>
    </action>
  </rule>
</ruleModel>
```

Fig. 7 Excerpt of an example rule for elderly assistance

medicine list (stored in the data model): for each medicine whose intake time is expected in less than 10 min from the current time, the medicine name in the user interface will be modified by increasing its font size, and setting its font colour to black. It is worth noting that there is an iteration in which two updates actions are applied to each user interface element identified by current medicine name. The android generator then transforms the references to the MARIA specification into references to the corresponding elements in the android implementation.

However, it may happen that, after a certain time, the user would not suffer anymore from some of the problems AQ5 he had when the rule visualised in Fig. 7 was created. For AQ6 instance, imagine that after a while the user no longer needs to take aspirin, Maalox and collyrium (visualised in Fig. 8) because the related acute diseases have disappeared. However, the user has still to take insulin on a regular basis. In this situation, the customisation rule R1 could even be deleted, and replaced by just a cellphone's vibration signalling the right time for taking insulin. In addition, after a while, since the user frequently tends to waking up during the night, there might be the need of creating a rule that automatically switches on the light in the living room. Thus, a suitable customization rule (R2) could take into account

Journal : Large 10209 | Article No : 620 | Pages : 17 | MS Code : UAIS-D-17-00125 | Dispatch : 1-6-2018

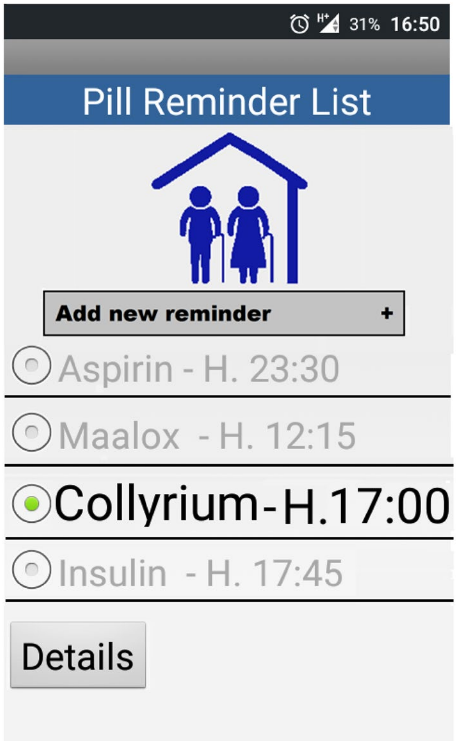


Fig. 8 Reminding pills in a trial application for elderly assistance

the motion and the light level sensors installed in the home in such a way that when the motion sensor detects a movement and if the surrounding light level is less than 200 lx (low light), then the living room light is turned on. The rule is composed of one event and one condition: the event is related to the motion sensor, while the condition is related to the light level.

To obtain the desired change of state (from off to on in the light installed in the living room), when editing the action part, the developer has to access the list of external functions in the MARIA specification (see the right panel in Fig. 9), select the relevant one, and indicate the parameters' values that should be used by the function when it is called. The rule specification obtained will contain an invoke function action, indicating the relevant external function to activate and the associated parameters values. In this case, the updateAppliance function takes as input four parameters: the room where the appliance is placed, the appliance name, the name of attribute that should be updated and the value that the attribute should take. The authoring tool facilitates the definition of these actions by supporting drag-and-drop of the external function definition from the right panel into the central one, where the rules are edited.

In the case of residential facility, the home environment and the vital signs of its occupant are thoroughly monitored, and the technological infrastructure available is more sophisticated, with actions referring to the IoT

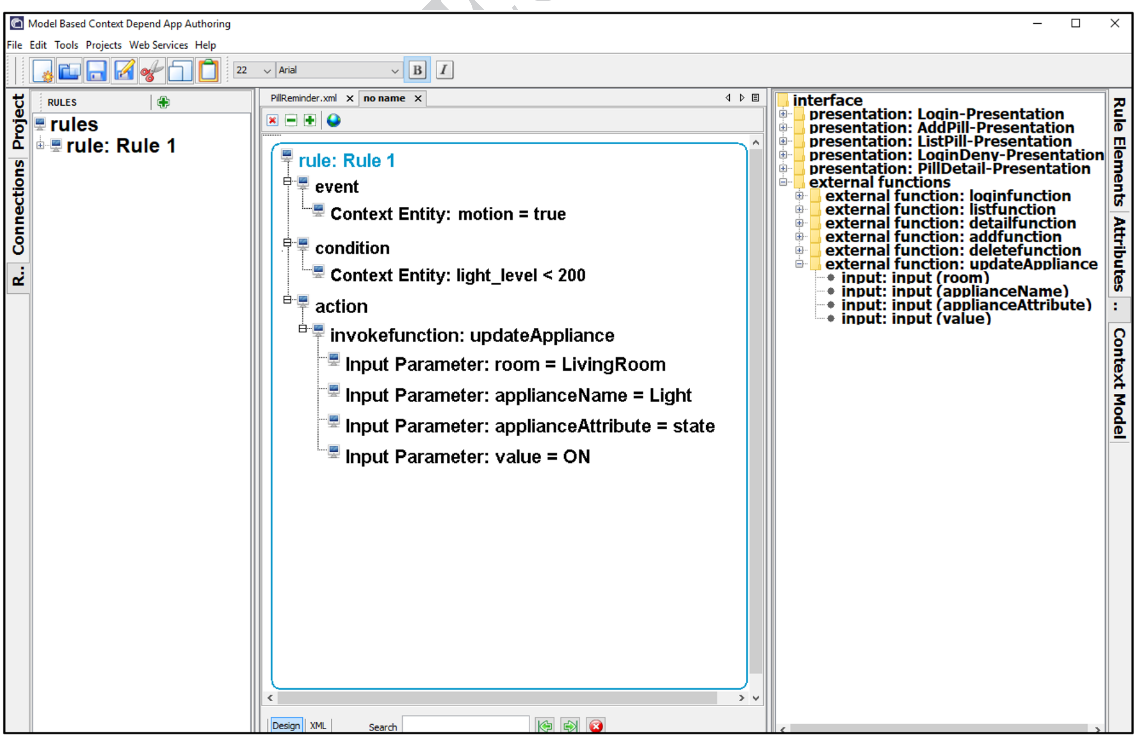


Fig. 9 Editing a rule with invoke function action

Journal : Large 10209	Article No : 620	Pages : 17	MS Code : UAIS-D-17-00125	Dispatch : 1-6-2018

appliances and devices available (e.g., allowing to control lights for the light therapy, reacting to information detected from the chest band). Thus, the kind of functionalities that the application should offer is very different from the one offered by the version of the application used in the apartments. For instance, in the case of residential centre, the application functionality and adaptation rule for pill reminders presented before would not be useful anymore as in the residential centre the professional staff takes care of this aspect. Rule examples that are relevant for the residential care setting are provided below.

R3: When two or more seniors are in the same shared living room, display on the TV the next planned recreational activities of the day. This can be done to stimulate the elderly to do some physical exercise as well as to increase their socialization with other peers.

R4: When the elderly is detected to be agitated (e.g., heart rate above 100 beats/min and respiration rate above 30 breaths/min), change the colours and intensity of the room lights to gradually move the user to a more relaxed and calm feeling and call the caregiver for further help. In situations of elderly becoming upset, a rule like this can be used to provide the elderly with an immediate help coming from the smart environment surrounding them, before professional staff arrive and directly offer help. In this case, the smart ambient would provide some light therapy-based support.

R5: When the elderly is likely to have fallen in her private bedroom, send a vocal alarm to the referring staff persons. This rule can be used for ensuring safety also when the elderly is alone in her private bedroom (in some situations they cannot even be able to call for help). In this case, accelerometer-based fall detector included (e.g., that in the Plux Bitalino chestband) can be used to recognise the situation.

However, similar to the previous rules these three rules can also be subject to dynamic adjustments and adaptations by designers (even after users already started using the application) to cope with changed and evolving requirements by the elderly in the residential facility. For instance, considering the R4 rule, designers can be asked by the care personnel to decrease the concerned thresholds for detecting elderly restlessness state (associated with a combination of respiration rate and heart rate), to be able to intervene timely, before the elderly becomes too agitated.

Likewise, after a while, the designer of the application can be asked to slightly change rule R5, since the involved vocal alarm was found not effective in noisy environments, thus running the risk that the alarm went unnoticed by the staff. In such a case, the new rule can be changed into one able to generate a multimodal message one (e.g., combining vocal, vibration, and graphical modalities).

9 Developers test

A test was conducted to collect feedback on the authoring tool. Users were recruited in the local department, and received a small gift as compensation. Six people (1 female) aged 25–48 ($M = 33.5$, SD = 8.1) participated in the test. As for their education, two users have a Master Degree (Digital Humanities and Computer Science), two have Bachelor, one user has a PhD in Information Engineering, one only secondary school. All of them have some programming experience: one has 1–2 year programming experience, four users have 6–10 year experience, one user has experience of more than 10 years. Four users had some limited experience in using MARIAE. Before the test, four users had not used any tool for creating context-dependent interactive applications, while two mentioned IFTTT. Five users had already heard about rule-based approaches for context-dependent customisation of interactive applications. In particular, one mentioned a generic trigger-action approach, three mentioned IFTTT. Users were asked to specify three rules referring to a shopping application whose MARIA specification was provided to them. Using the tool, they had to create rules so that the resulting application would exhibit more dynamic context-dependent behaviour. The rules were: R1: "When the user is near a public display, show recipes that can be made using the products in the list"; R2: "If the user is celiac, add a promotion on gluten-free products in the presentation showing the list of products"; R3: "If the user is going to exit the shop and it is raining, load a presentation showing a map with the best path to reach the user's home and avoid traffic".

One evaluator observed the users' interactions during the experiment. Before the test, users were given some slides providing a general introduction, instructions on how to access the tool, explanations of its main features and tasks to carry out. After the test, users filled in a questionnaire, which included a demographic section (about e.g., education, experience/familiarity with programming and modelling), and a section more related to the tool. In the questionnaire, a 1–5 Likert scale (1 = not usable; 5 = very usable) was used for ratings. The feedback was positive, and the ratings were on average always on the positive side.

Allocation of screen space between the presentation of the model-based UI description and rules (Median = 4). One user said that the way this combination was supported is confusing and needs more detailed information.

Rule presentation and editing in the tool (Median = 4). One user declared that the creation of rules improves as soon the user understands the underlying mechanism. Another user said that he would have preferred more

Journal : Large 10209 | Article No : 620 | Pages : 17 | MS Code : UAIS-D-17-00125 | Dispatch : 1-6-2018

control within the central panel, without the need to move between the central and the right panel of the tool. Another user suggested adding some hints for better supporting users during the editing phase (e.g., if a mandatory value has not been set in the attributes tab yet, this should be highlighted to the user).

Presentation of the context hierarchy within the tool (Median = 4). One user said that when the underlying mechanism is understood, rule creation improves, though at first it is not very intuitive. Another user said that the context model elements are visualised appropriately although a 'search' functionality could help to more directly access contextual model elements.

Clarity of specifying the behaviour of a context-dependent application through this ECA-based approach (Median = 3.5). One user suggested using some presentation techniques (e.g., different colours) to better identify events/conditions/actions in the central panel.

In the stacked bar chart below, it is possible to see an overview of the main aspects rated by users in the questionnaire (Fig. 10).

Aspects that users liked the most in the tool One user liked the context hierarchy, another user liked the idea of context-dependent customization of interactive applications. Another user appreciated the classification of the context model (which he judged complete and suitable for modelling many situations), the tree-like visualisation of the rule, and the possibility to act directly on the various elements of the UI specification.

Aspects that users disliked the most One user did not like the supported drag & drop, he suggested simplifying the tree-like structure to create actions, events and conditions, by making more steps automatic (when possible). One user said that the features of the tool become clearer as soon as users acquire familiarity with the tool. From the test, it also came out that frequent mouse movements between the central panel and the right panel could make the interactions a bit tiring: as a solution, one user suggested adding more controls directly in the central panel.

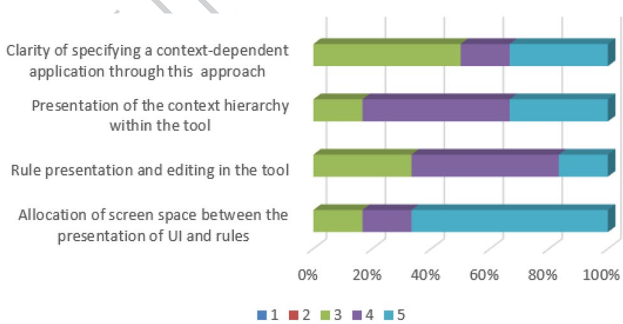


Fig. 10 Stacked bar chart providing an overview of users' ratings on key aspects of the tool

In terms of useful application areas, users mentioned AAL scenarios and domotic applications. Although the test was a first study, its results show that CAT was usable by people different from its developers, and it was appreciated by the test subjects (e.g., the hierarchy for defining events/conditions), although some aspects (e.g., interacting with some panels) could be further improved. Participants liked the flexibility provided by the tool in supporting different types of rules and scenarios.

As for the task success, all the users successfully specified the 1st rule, both in terms of events/conditions and actions. In addition, there was just one user who wrongly specified the 2nd rule (he forgot to specify the condition involving the user's celiac disease) and the 3rd rule (he forgot to specify the condition involving the rainy weather), while the remaining ones successfully specified them. Thus, the results of the test were encouraging in terms of the capability of users in specifying context-dependent behaviour using the proposed approach. Moreover, users especially appreciated the context hierarchy, which is one of the key features of our approach, which was judged overall suitable for modelling many real world situations. Finally, as evaluators, we also observed progressively increased efficiency in the time needed for creating the rules during each test.

10 Discussion and conclusions

The proposed approach can provide useful contributions in various aspects concerning development of context-dependent mobile apps.

Rule-based approach for creating context-dependent versions of mobile apps effective for handling dynamic customisations Traditional approaches tend to handle the behaviour resulting from context-dependent adaptation decisions by hard-coding it in the implementation code. Using the proposed rule-based approach, there is a separation of concerns between the application logic and its adaptive, context-dependent behaviour, as it is possible to easily compose rules expressing context-based adaptation decisions for versions targeting different contexts of use.

Facilitating reuse of design artefacts among different configurations of the same software application Current practises do not adequately support context-based definition and configuration of application variants, since the development of each variant is generally kept separated. This results in considerable redundancies between versions, and it is also a time-consuming and error-prone way of proceeding. With the proposed approach, every context-dependent variant of the same application shares a common set of core functionalities (modelled in a specific high-level language), thereby facilitating reuse of design/development artefacts.

Journal : Large 10209 | Article No : 620 | Pages : 17 | MS Code : UAIS-D-17-00125 | Dispatch : 1-6-2018

Abstraction level for handling heterogeneity of IoT hardware/sensors/events In current approaches, designers often need to access low-level libraries to configure and exploit various hardware and sensors. In this approach, events are modelled and made available to designers at a logical level and they are uniformly handled by the context manager, which also deals with the dynamic appearance/disappearance of associated resources. In addition, the context management support is not hard-coded in a single application but handled by a specific module, which can be easily accessed by various applications.

Support for maintenance and further evolution of the apps From the first tests conducted, we gathered encouraging feedback regarding the potential of our approach to help in more easily and effectively making applications evolve in a context-dependent manner. However, to confirm and improve on such results, further tests will be carried out in the future.

To conclude, in this paper we present a method and a set of tools for supporting developers in customizing context-aware apps by extending the original behaviour through event–condition–actions rules. The method exploits model-based descriptions to facilitate obtaining implementations for various environments (e.g., android, web). We describe a trial application and report on a first test with developers.

Future work will be dedicated to systematic empirical evaluation of the approach with software developers to further validate its initial—yet promising—potentialities in offering significant benefits to reduce development efforts and costs needed for the customisation, maintenance and evolution of mobile applications.

Acknowledgements We thank Luca Pardini for his help in the implementation of the android generator for MARIA specifications.

References

1. https://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/
2. AQ8 Kurtev, I., Bézivin, J., Jouault, F., Valduriez, P.: Model-based DSL frameworks. In: Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '06), pp. 602–616. ACM, New York. (2006). https://doi.org/10.1145/1176617.1176632
3. Le Goaer, O., Waltham, S.: Yet another DSL for cross-platforms mobile development. In: Proceedings of the First Workshop on the Globalization of Domain Specific Languages (GlobalDSL '13), 28–33. ACM, New York. (2013). https://doi.org/10.1145/2489812.2489819
4. Majchrzak, T.A., Schulte, M.: Context-dependent testing of applications for mobile devices. Open J. Web. Technol. (OJWT) **2**(1), 3–14 (2015) (**RonPub**)
5. Rieger, C., Majchrzak, T.A.: Weighted evaluation framework for cross-platform app development approaches. In: Wrycza, S. (ed.) Proceedings of the 9th SIGSAND/PLAIS EuroSymposium on Systems Analysis and Design Information Systems, Lecture Notes in Business Information Processing (LNBIP), pp. 18–39. Springer, Berlin (2016)
6. Rieger, C., Majchrzak, T.A.: Conquering the mobile device jungle: towards a taxonomy for app-enabled devices. In: Proceedings of 13th International Conference on Web Information Systems and Technologies (WEBIST), pp. 332–339. SciTePress (2017)
7. Jia, X., Jones, C.A.: AXIOM: a model-driven approach to cross-platform application development. In: Proceedings of 7th ICSOFT (2012)
8. Heitkötter, H., Majchrzak, T.A., Kuchen, H.: Cross-platform model-driven development of mobile applications with md2. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13), pp. 526–533. ACM, New York. (2013). https://doi.org/10.1145/2480362.2480464
9. Majchrzak, T.A., Ernsting, J.: Achieving business practicability of model-driven cross-platform apps. Open. J. Inf. Syst. (OJIS) **2**(2), 3–14 (2015) (**RonPub**)
10. Ernsting, J., Rieger, C., Wrede, F., Majchrzak, T.A.: Refining a reference architecture for model-driven business apps. In: Proceedings of 12th International Conference on Web Information Systems and Technologies (WEBIST), 307–316. SciTePress (2016)
11. Heitkötter, H., Kuchen, H., Majchrzak, T.A.: Extending a model-driven cross-platform development approach for business apps. Sci. Comput. Program **97**(1), 31–36 (2015)
12. Dickson, P.E.: Cabana: a cross-platform mobile development system. In: Proceedings of 43rd SIGCSE. ACM, 2012, pp. 529–534 (2012)
13. Nebeling, M., Dey, A.K.: XDBrowser: user-defined cross-device AQ9 web page designs. CHI 5494–5505 (2016)
14. Van de Camp, F., Stiefelhagen, R.: GlueTK: a framework for multi-modal, multi-display human–machine-interaction. In: Proceedings IUI'13, pp. 329–338. ACM Press
15. Ur, B., McManus, E., Ho, M.P.Y., Littman, M.L.: Practical trigger-action programming in the smart home. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14), pp. 803–812. ACM, New York. (2014). https://doi.org/10.1145/2556288.2557420
16. Coutaz, J., Crowley, J.L.: A first person experience with end-user development for smart home. IEEE Pervasive Comput. **15**(2), 26–39 (2016)
17. Ghiani, G., Manca, M., Paternò, F., Santoro, C.: Personalization of context-dependent applications through trigger-action rules. In: ACM Transactions on Computer–Human Interaction (ACM TOCHI), (2017)
18. Dey, A., Sohn, T., Streng, S., Kodama, J.: iCAP: Interactive prototyping of context-aware applications. Pervasive 254–271 (2006)
19. Ghiani, G., Manca, M., Paternò, F.: Authoring context-dependent AQ10 cross-device user interfaces based on trigger/action rules. In: The 14th International Conference on Mobile and Ubiquitous Multimedia (MUM2015), pp. 313–322. ACM Press
20. Gajos, K., Weld, D.S.: SUPPLE: automatically generating user interfaces. In: IUI '04: Proceedings of the 9th International Conference on Intelligent user interface, pp. 93–100. ACM Press, New York (2004)
21. Eisenstein, J., Vanderdonckt, J., Puerta, A.: Applying model-based techniques to the development of UIs for mobile computers. In: Proceedings of the 6th International Conference on Intelligent user interfaces (IUI '01), pp. 69–76. ACM, New York. (2001). https://doi.org/10.1145/359784.360122
22. Mayer, S., Tschofen, A., Dey, A.K., Mattern, F.: User interfaces for smart things-A generative approach with semantic interaction descriptions. ACM Trans. Comput. Hum. Interact. (TOCHI) **21**(2), 12 (2014)
23. Usman, M., Iqbal, M.Z., Khan, M.U.: A product-line model-driven engineering approach for generating feature-based mobile

Journal : **Large 10209**	Article No : **620**	Pages : **17**	MS Code : **UAIS-D-17-00125**	Dispatch : **1-6-2018**

applications. J. Syst. Softw. **123**, 1–32. (2017). https://doi.org/10.1016/j.jss.2016.09.049(**ISSN 0164–1212**)

24. Halbrügge, M., Quade, M., Engelbrecht, K., Möller, S., Albayrak, S.: Predicting user error for ambient systems by integrating model-based UI development and cognitive modelling. In: Proceedings Ubicomp'16, pp. 1028–1039. ACM Press
25. Nguyen, T., Vanderdonckt, J., Seffah, A.: Generative patterns for designing multiple user interfaces. In: Proceedings of the International Conference on Mobile Software Engineering and Systems (MOBILESoft '16), pp. 151–159. ACM, New York. (2016). https://doi.org/10.1145/2897073.2897084
26. Karuzaki, E., Savidis, A.: Yeti: yet another automatic interface composer. In: Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '15), pp. 12–21. ACM, New York. (2015). https://doi.org/10.1145/2774225.2774843
27. Yigitbas, E., Sauer, S., Engels, S.: G: self-adaptive UIs: integrated model-driven development of UIs and their adaptations. ECMFA 126–141 (2017)
28. Paternò, F., Santoro, C., Spano, L.D.: MARIA: a universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput. Hum. Interact. **16**(4), 19:1–19:30 (2009)
29. Paternò, F., Santoro, S., Spano, L.D.: Engineering the authoring of usable service front ends. J. Syst. Softw. **84**(10), 1806–1822 (2011)
30. http://bitalino.com/en/

Journal : **Large 10209**	Article No : **620**	Pages : **17**	MS Code : **UAIS-D-17-00125**	Dispatch : **1-6-2018**

Journal:	**10209**
Article:	**620**

Springer
the language of science

Author Query Form

Please ensure you fill out your response to the queries raised below and return this form along with your corrections

Dear Author

During the process of typesetting your article, the following queries have arisen. Please check your typeset proof carefully against the queries listed below and mark the necessary changes either directly on the proof/online grid or in the 'Author's response' area provided below

Query	Details Required	Author's Response
AQ1	Author: This sentence" We show its potential by describing ...." has been slightly modified for clarity. Please check that the meaning is still correct, and amend if necessary.	
AQ2	Author: This sentence" The MARIA environment also provides a tool supporting ...." has been slightly modified for clarity. Please check that the meaning is still correct, and amend if necessary.	
AQ3	Author: This sentence" In this section we consider ....." has been slightly modified for clarity. Please check that the meaning is still correct, and amend if necessary.	
AQ4	Author: This subheading " ...application" has been slightly modified for clarity. Please check that the meaning is still correct, and amend if necessary.	
AQ5	Author: Please check Figures 7 and 8 are interchanged to maintain sequential order.	
AQ6	Author: The caption of figure 8 has been slightly modified for clarity. Please check that the meaning is still correct, and amend if necessary.	
AQ7	Author: Please check and confirm the inserted citation of Figure 10.	
AQ8	Author: Please update references [2], [6], [7], [8], [10], [12], [14], [15], [17], [19], [20], [21], [24], [25] with full details.	
AQ9	Author: Please update references [13], [18], [27] with volume id.	
AQ10	Author: Please update reference [19] with year.	
AQ11	Author: Please provide the access date for references [1] and [30].	