One App to Rule Them All Collaborative Injection of Situations in an Adaptable Context-Aware Application

Riadh Karchoud, Philippe Roose, Marc Dalmau, Arantza Illarramendi, Sergio Ilarri

Noname manuscript No.
(will be inserted by the editor)

One App to Rule Them All

Collaborative Injection of Situations in an Adaptable Context-Aware Application

Riadh Karchoud · Philippe Roose · Marc Dalmau · Arantza Illarramendi · Sergio Ilarri

Received: date / Accepted: date

Abstract Currently, we are living in the era of ubiquitous computing, that introduces the possibility to have an increasing number of mobile applications on different types of devices with ever-growing capabilities. Consequently, the continuous rise of mobile applications opens the door for an unmatched number of diverse possibilities of what users can do and expect to do. Due to the high demand for apps and the unstoppable growth of app stores, the computing world is slowly shifting towards an interconnected, distributed, and context-aware digital ecosystem. With so many possible use cases and such diverse user needs, is it desirable to have one single application that does it all? Has it become a necessity to have one application able to understand users and eliminate the need for other applications?

Our vision of this single application is a context-aware distributed mobile application dedicated to everyday users. This app needs to offer to the users a high level of comfort and a better-customized user experience by replying both re-actively and pro-actively to the users' needs without confusing them with the large diversity of apps and devices available. Nonetheless, no predefined application can predict or autonomously handle all the possible situations that could happen to the user in all different areas (shopping, work, travel, etc.), due to the infinite possibilities. Therefore, our proposal allows to dynamically

Riadh Karchoud
E-mail: karchoud.riadh@gmail.com

Philippe Roose
E-mail: Philippe.Roose@iutbayonne.univ-pau.fr

Marc Dalmau
E-mail: dalmau@iutbayonne.univ-pau.fr

Arantza Illarramendi
E-mail: a.illarramendi@ehu.eus

Sergio Ilarri
E-mail: silarri@unizar.es

add new use cases, by both non-experts (e.g., everyday users) and domain experts (e.g., a travel agent), into the user's application.

Keywords Context-aware mobile applications, pervasive applications, situation awareness, reactive systems.

1 Introduction

Nowadays, devices are packed with a large amount of applications for multiple purposes. These applications try, in their own categories (social, work, entertainment, etc.), to fulfill the needs of the users. However, the diversity of those needs makes it a challenge for mobile applications to accurately understand their users and respond to them by managing adequately their daily situations regardless of their nature or categories.

The interest for this kind of applications comes from the fact that users have repetitive habits, that they perform on a regular basis, and evolving needs, that continuously change and shift according to their physical and social environment. Those reasons raise the issue of the relevance of services that can be automatically offered by context-aware applications in order to respond to certain situations.

In the specialized literature, we can find different proposals in the area of context-aware applications. However, in general, they only partially cover the needs of users, due to three main reasons:

- Firstly, they focus their work on understanding the context of the user under a specific limited area of expertise (museum guided tours (Chen and Huang, 2012), context-aware health-care, tourism (Basiri et al., 2017), smart transportation, etc.) in which users and developers, with no expertise in those domains, could not either improve or adapt the application to behave according to the actual requirements of the user.
- Secondly, they cannot be either extended to cover wider use cases or enriched to handle more precisely specific personal situations, due to the lack of dynamicity in the predefined rule-based models that they support.
- Finally, most of the context-aware applications focus either solely on one device (on which they are installed) or on a complex network of sensors (Sang-Seok et al., 2017) that should be pre-installed all around the user in order for them to run correctly.

Therefore, in the considered scenario, with a large variety of use cases, diversity of needs, and multiplicity of devices, it becomes a necessity to have a solution that handles them properly. In this sense, we propose a Long Life Application (LLA) or Eternal Application, as it runs ubiquitously and evolves constantly by changing its behavior and offering a variety of services according to the user's needs.

One main novelty of the proposed LLA is an injection mechanism (Karchoud et al., 2017b) that enables users and other external sources to continuously and proactively introduce new potential situations into the user's LLA

application, in order to improve its understanding about the user's situations and overcome the lack of dynamicity in current existing applications.

The objective of this paper is first to present briefly how our LLA proposal is able to create/inject/detect everyday situations and react to them dynamically, thus providing the appropriate services for the users. Next, to show the details of the main contribution of the paper, that is the injection mechanism that enables users and other external sources to continuously and proactively introduce new scenarios and services into the user's application, in order to overcome the fact that only 25% of users return to any given application after the first use (Grennan, 2016). This paper is an extended version of (Karchoud et al., 2017b); specifically, we provide a more detailed description of both the situation injection mechanism and the prototype developed, besides including a description of the injected situation model.

This paper starts in Section 2 by presenting related work on mobile applications focused on context awareness dedicated to mobile end users. Then, Section 3 presents the global idea behind our LLA proposal and Section 4 summarizes the basics of the situation-based contextual modeling proposed. After that, Section 5 is focused on the context injection mechanism used to enrich the application dynamically. The way in which the matching among services and situations is handled is presented in Section 6. Then, Section 7 shows a real-life scenario and we compare our proposal with a classical one based on the use of app stores. The paper finishes with some conclusions about the proposal and some prospective lines of future work.

2 Related work

The final goal of our work is proposing an end-user dynamic context-aware application. After studying all the areas that lead to building and running this kind of applications, we collected information about both commercial and research applications providing the wanted features for common usage on mobile devices. So, in the following, we present a set of existing applications that provide similar features to what we want to achieve. First, we describe some commercial apps created by companies; afterwards, our focus turns to research works tackling this area.

Compared to the works described in this section, our novel contribution is illustrated by our detailed contextual situation model which is able to be enriched continuously from a variety of sources. Although some works propose injecting context information (by using rules, natural language, or voice commands), they are usually restricted to predefined rules or simple commands.

2.1 Commercial apps

Google is one of the biggest companies putting an effort to aim their work at the area of dynamic applications able to detect the context and offer customized services to their users. Its *Google Now* application (Martin, 2017) is

triggered by contextual changes or voice commands. It detects/gathers relevant data (home location, work location, calendar events, Google+, etc.) about users and saves their Google searches using the Google Knowledge Graph (Google, 2017). It uses other Google services (Google Maps, Google Images, Google Alert, etc.) to provide features like getting updates on sports, movies, and events.

Google also took another interesting initiative with context-aware applications, but in a different format, with *Google Instant Apps* (Lardinois, 2017). They offer the possibility for developers to decompose their applications into smaller apps (activities) dedicated to specific needs and triggered by clicking on a web link or by using NFC (Near Field Communication). For example, to buy an item on Amazon, the user reaches the item by using Google and then, when he/she is about to pay, the Amazon payment instant app is launched with no installation required. Another use case that Google presents is a parking micro-app that is only launched when the user faces a parking meter machine and places his/her phone on it to trigger the parking app. It uses an approach similar to micro-apps (Syer et al., 2011). The strength of these solutions comes from the fact that they are built on the multiple services of Google. Nonetheless, Google focuses their applications on predictive recommendations that may be interesting to the user only in a limited range of situations.

EasilyDo (Edison, 2017), *24me* (24me, 2016) and *Tempo* (Rodgers, 2013) are all personal assistants solutions for productivity planning and scheduling. Tempo is based on an AI (Artificial Intelligence) approach able to pull user contextual information and predict meeting places and attendees. 24me integrates the user's calendar, tasks, notes and personal accounts together, in order to customize the application and have automated reminders about paying bills, sending gifts, going to doctor appointments, etc. EasilyDo offers some predefined smart features like auto-dialing conference calls and viewing attendees LinkedIn profiles, getting bad weather alerts, accessing boarding passes, and getting flight status. The problem of these applications comes from their limitations (redundancy of recommendations/features, lack of dynamicity, control out of the user's hands, etc.) and the lack of diversity of the services they offer (limited to notifications, recommendations, etc.).

IFTTT (IF This Then That) (Ovadia, 2014) is another interesting solution. It considers the user domain (devices) and exploits it by using web services able to communicate, link and control remote devices (such as connected lamps). Although its contextual model is poor, it offers pertinent useful scenarios (e.g., turn on the lights automatically at 7 pm). The issue is that IFTTT is dependent on those services, which makes it repetitive and not convenient for the dynamic nature of context-awareness.

2.2 Research works

SECE (Sense Everything, Control Everything) (Boyaci et al., 2010) is an application for context-aware service composition offering a rule-based approach

in a more user-friendly way, enabling users to define the expected behavior of a set of web-based services under certain situations. These situations are used to trigger the composed service execution. They are based on basic contextual information like the location and time. Even though SECE is user-friendly, inserting rules in a written manner is always considered a boring task by mobile end users.

DoTT (Do This on That) (Chihani et al., 2013) is another rule-based system using context providers in order to be aware of the user's situations instantly. It is built over a reasoning engine able to comprehend the adaptation rules and react to the user's context changes by following those rules. As output, it has an interactive interface that can host the services provided by the system. The difference between this approach and other rule-based systems is that the rules in this work are presented in natural language, following a specific grammar (sentences). The drawback is the limitations of its basic services (messaging, calls, calendar, and social aspects). Moreover, Natural Language Processing (NLP) has limitations and issues when it comes to languages like Arabic (Farghaly and Shaalan, 2009).

Dig-Event (Zhao et al., 2011) is a mashup service that allows the users to define and sort activities such as trips and meetings (considering an optimized order according to the time and location), obtaining recommendations of a diversity of relevant services for performing those activities. Recommendation systems rely on context-based selection criteria including the time, type of activity, budget, etc. These technologies are equipped with applications that can help their owners to navigate in the surrounding areas by proposing different activities, such as restaurants where to eat, shopping centers to buy from, or museums to visit, based on the preferences and the profile of the owner. Nevertheless, this system lacks flexibility in context acquisition (Basiri et al., 2017). In Dig-Event, the user's context is the result of the manually-entered information when declaring an activity, but it does not consider context changes that occur when doing that activity.

The last type of context-aware applications are the so-called *trail-based applications* (Clarke and Driver, 2004). A *trail* is a contextually-scheduled collection of activities and represents a generic model that can be used to satisfy the activity management requirements of a wide range of context-based time management applications. Combining the trails concept with mobile devices, context-aware technology creates opportunities for innovative activity-based application development. As an example, *Hermes* (Driver and Clarke, 2004) is a software framework for mobile, context-aware trails-based applications, which supports developers by providing generic components containing structure and behavior common to all trails-based applications. It organizes, using a mathematical model, the activities of the user according to his/her previous trails. The problem with this approach is that it is too much focused on travel scenarios, rather than in everyday situations that can happen also indoors.

3 Proposed approach

From the perspective of the user, our front-end application is the result of the combination of software components deployed on the user's device (or devices) when needed. Nevertheless, behind what the user sees, there is a combination of processes and modules working simultaneously to offer the wanted results to him/her. The approach starts by collecting contextual information that helps the framework (back-end) to deploy the adequate components through the middleware. These components will form the visible part of the application (front-end) in the *user domain* (list of available connected devices owned by the user).

LLA differs from the classical approach where the user needs to look for the applications; instead, it makes the services come to the user when needed. The application is based on a modular architecture that is able to detect, understand, and react to the user by offering services related to his/her needs. Both the deployed services and the main architecture are built based on the Kalimucho middleware (Da et al., 2014), in order to manage the distribution aspect of the proposal.

The general process of LLA (see Figure 1) starts by extracting contextual information from the user domain and then inputting it into the core architecture in order to animate it (i.e., active its reaction). Inside the core of our proposal, we implement a process respecting the *Event Condition Action (ECA)* approach (Nakagawa et al., 2012).

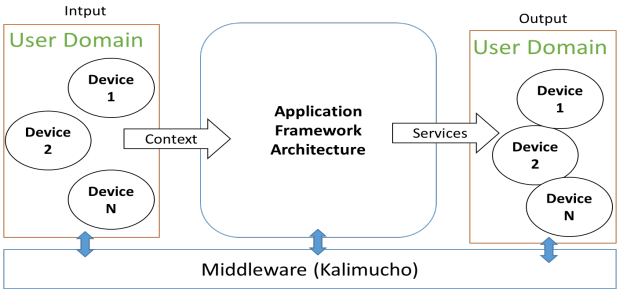

Fig. 1 LLA's overall workflow

When fed to the core, this contextual information helps to construct an *event*, to verify if *conditions* surrounding that event are met and, finally, to deploy services as an *action* for that event. Through this process, the contextual information respects a defined situation-based contextual model that structures and represents the data extracted from the devices in order to build an entity (*situation*), which can be understood by both the application and the users.

After defining the main components of our proposal and describing the workflow process, we evaluated the proposal by comparing it to existing solu-

tions (see Section 2). This comparison shows that our solution improves the limitations of existing applications by providing a number of novel contributions on the theoretical and technical level. In order to build this proposal, the first step was to formulate the situation-based contextual model, which is the focus of the next section.

4 Situation model

Situation awareness is commonly defined as the perception of environmental elements with respect to time or space, the comprehension of their meaning, and the projection of their future behavior (e.g., see (Endsley, 1995)).

The *situation* is the key component in our system (Karchoud et al., 2016). According to the Cambridge dictionary, a situation is *"The set of things that are happening and the conditions that exist at a particular time and place"*:

- The set of things = Activity – What are you doing?
- Time – When?
- Location – Where?
- Conditions = Exceptions – What are the exceptions?

Besides, for the context-awareness to be effective, we need to answer another question, in order to know what the user expects when a specific situation happens:

- Service – What do you expect to happen?

Figure 2 represents the building blocks of the situation representation model in our application.

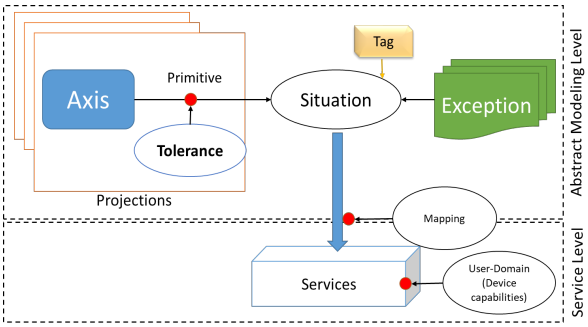

Fig. 2 Situation representation model

In our proposal, a situation is represented on two different levels. The first level is the abstract modeling level where the information that builds any situation is defined. The second level is the service level, where the reaction strategy to the situation is defined. This representation model is the core of

our proposal. In (Karchoud et al., 2017a), we presented a clear representation of this model and we described the way LLA is able to use the model in order to detect the context across multiple devices, which was the more novel contribution of that part of our solution. In this paper, we focus on the problem of dynamic situation injection into LLA.

5 The situation injector

A major issue with mobile applications in general, and context-aware applications in particular, is their repetitiveness and their usual limitation to restricted application domains or closed spaces equipped with sensors (Harter et al., 2002).

With the aim of overcoming that issue, LLA handles a semi-automatic, collaborative, open injection mechanism that continuously provides the user with richer contextual awareness and more suitable services. In this way, we ensure the continuous growth of LLA by making it able to enhance and enrich the user experience by managing (adding, deleting, or modifying) new situations introduced through the injection mechanism. This mechanism is based on a user-friendly situation model in order to make the injection an easy and understandable procedure that can be performed even by end users with no understanding of technical requirements. The injection mechanism has multiple sources of injection and communicates directly with the user's application.

5.1 The injector's workflow

The injector acts as an input (inserter) dedicated to enriching the context awareness considered for the user, based on diverse sources. It can be used to inject/modify/delete/update situations and/or services without having to access directly the devices of the user. Various sources can inject the situation's description into the user's app. The data of this layer is stored in cloud storage and is shared among all the user's devices. This solution avoids redundancies and inconsistencies among the user devices regarding the considered situations, in order to keep the coherence of LLA.

One benefit of using this high-level contextual injection mechanism, which extracts context data from multiple sources, is its capability to provide a way of increasing continuously the repository of monitored situations, which allows LLA to be more customized to the user along time.

LLA needs to be aware of the user's habits, needs, and social environment. Three main categories of context injection sources are considered by the application: the user, social media, and external providers, as described in the next subsections. Each main source of injection uses a different process to collaboratively update the application of the user in a transparent way that does not require any unnecessary downloads.

The advantage of this mechanism resides in eliminating the time and network usage usually needed by current existing applications to do regular heavy

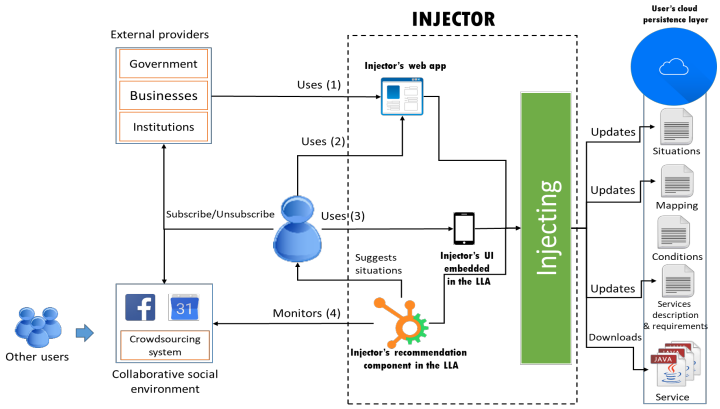

Fig. 3 The injector's workflow

updates, for even the simpler changes in their code base. Using our approach, the updates will be occasional and probably affect only the situation file or the mapping file by using already-stored components. Even at the service level, developers will be able to modify components separately and therefore have fewer and lighter updates.

5.2 User's injection process

The first source of situations is the user himself/herself, and more precisely his/her needs and habits. What motivated us to propose this is the repetitiveness that users feel while they set their alarm every night or while they open their emails every morning. These habits can be automated in order to help the user to have the same experiences without having to always perform those same tasks again and again.

To do so, the user has the possibility to use either the injector's UI, which is accessible from his/her LLA's Manager UI (mobile version) or the Web Injector. In the first case (see Figure 3 "Uses (2)"), the mobile injector is dedicated to building simple and fast situations by working on only one *projection* (a projection is a specific representation of the contextual information) and three *axes* (conceptual directions, like the time and location, that represent the evolution of context) per situation. In the second case (see Figure 3 "Uses (3)"), a web application allows a more precise and rich description of situations and services by using multiple projections and six axes.

In both cases, the user can either create, delete or modify a situation by inserting/modifying the required values (like the time, location, and activity) and then update or create a matching by selecting a service or a set of services to be deployed when that situation is detected. The user can also decide to share these situations with other users.

After selecting the required service/s, the user verifies if the components composing the selected service/s exist/s in his/her repository. A possible extension of this could be to have a store for services instead of applications, where users can select services to download into their local or cloud storage in order to use them willingly when needed. Nonetheless, if the user wishes for the required service that it does not stay in the storage space of his/her device, he/she could specify to trigger the download only when the situation is detected and ask to be deleted when that situation finishes.

Finally, the user has the possibility to subscribe to, or unsubscribe from, other injection sources freely, giving those sources the possibility to provide him/her with new situations, mappings, and services.

5.3 Collaborative social environment's injection process

Social media has become the largest source of information about users' activities, preferences, etc. Moreover, we live nowadays in the age of media sharing. In this scope, the injector incorporates into the LLA's architecture a component running transparently in the background, which monitors the user's social media (if the user subscribes to it and allows this monitoring) in order to suggest/recommend new situations.

In order to build these suggestions, the injector follows a specific process (see Figure 3 "Monitors (4)"). After extracting events and birthdays from Facebook, tasks from Google Calendar, etc., the injector mechanism asks the user to specify the tolerance and the expected reaction to these potential situations. The user can select services from the Services Repository to be deployed automatically when the situation is detected.

If the monitor detects a new shared situation coming from another user, it suggests it to the user and allows him/her to choose whether to download the services that other users recommend for that specific situation or simply select his/her own services. Otherwise, if the user does not specify any reaction strategy, the application creates a situation and assigns a reminder service by default. This widens the applicability of contextual engagement by giving the user the possibility to create/share his/her own situations or extract them automatically from his/her social environment (using data from Facebook, Google Calendar, Twitter, etc.).

For example, if the user received an invitation to a concert event on Facebook and he/she chooses to participate, the injector's recommendation component, which is monitoring the user's social media, detects this event, builds a situation, and proposes adding it to the user's own situations; then, the user accepts this situation; finally, he/she also selects a ticket provider service and a video streaming service that he/she wishes to use when the concert situation happens.

5.4 External providers' injection process

External providers are a very important source of injection in this proposal, as it completes the mechanism with an open system that can provide endless possibilities to the user by continuously proposing new situations and services. Whereas the other sources of injection (the user and social media) handle private and social situations, this source covers a wider range of situations related to a variety of domains that otherwise could not be considered. The providers are external sources that the user can subscribe to in order to allow them to inject his/her application with situations and provide him/her with services.

For these sources, the process of injecting situations (see Figure 3 "Uses (1)") is done with the help of a web application. Providers can create situations (like users), and then select among their subscribers the ones that are concerned by that specific situation. Nonetheless, they should implement their own services and components using the Kalimucho framework (Da et al., 2014), to enable their easy management in the proposed framework. Finally, the injector broadcasts those situations, mappings, and services, to the concerned users.

The situations and mappings are described via a web application and then injected to the subscribers of the provider along with the appropriate services (software components). The external providers are classified into three main categories, as described in the next subsections.

5.4.1 Government providers

The first category is formed by government organisms and services (universities, hospitals, the police, etc.). If the user subscribes to these organisms, he/she allows them to inject situations.

For example, if the police decide to close some borders due to an emergency, it injects that situation to all the users in the vicinity, in order to notify them and propose another road when the user gets close to the borders on that specific date.

5.4.2 Businesses and private companies

Any business (McDonalds, Carrefour, a gas station, etc.) can offer its situations and allow users to subscribe and use its services. Considering other applications that use context awareness to aim advertisements at users, our proposal provides a new way to engage those users, not limited to texts and notifications but enhanced by the possibility to offer dedicated customizable services instead.

For example, a parking company can construct and inject a situation into the users' situation repository that, upon detecting that the user entered one of its parking areas, deploys a parking service to help the user to find the

closest empty space and remind him/her where he/she parked when he/she comes back to leave.

5.4.3 Institutions/organizations/associations

These providers represent non-governmental entities but that, as opposed to business and private companies, do not necessarily obtain a financial gain from the user.

An example of this would be a football club that injects training sessions situations to its players' applications. When the defined primitives are verified and service hardware requirements are met, LLA can deploy a Biometric Monitor Service to track the performance of the players.

6 Matching the context to services

LLA uses a cloud layer as a medium between users and providers, where it acts as an application store. This means that this layer synchronizes the situations, services, and mappings for the user, and allows providers to enrich them. The injected data reflect our situation model and allows LLA to understand what is happening and how to react.

6.1 Abstract level

At the abstract modeling level, the situation is represented by a combination of data and concepts. A situation, in LLA, is a combination of multiple projections on different axes. Each projection is a combination of times, locations, and activities projected using binary describers (*primitives*). Besides, there may be *exceptions*, based on the same axes and primitives (for more details, see (Karchoud et al., 2017a)). The situation's abstract model is represented as follows:

- Graphically: Using operations and variables, a situation is represented through the defined concepts. The graphical representation is used to have a clearer understanding of the situations.
- Textually: A situation S is presented textually in the following format, where P is a projection:

  S : Tag ; Pn [Axis(Primitive(values, tolerance)...); ...] [...] ... EXCEPT [Situation/Axis(...)]

In LLA, these textual representations are translated into XML and injected into the user's application (*situation repository*) in order to define his/her situations.

For example, we present a situation S1 (House Alarm Situation) that happens when the time is between 10pm and 6am with a 20 minutes tolerance (Projection1) and inside his/her house, or, when the user is outside his/her home with a 20 meters tolerance value (Projection2). The exception to this is being on Sunday. For our example, the graphical representation is shown in Figure 4. Textually it is presented as indicated in the following:

S1: Projection1[Time(A(22,20) B(6,20)); Location(I(Home,5))] OR Projection2 [Location(O(Home,20))] EXCEPT [(Time(Sunday))]

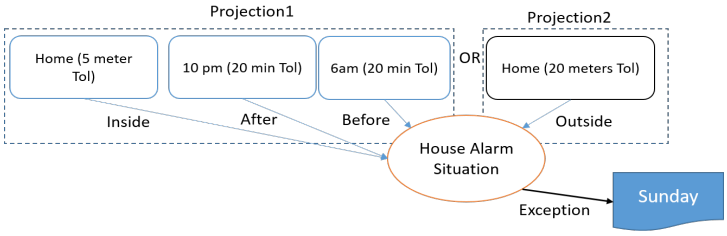

Fig. 4 House alarm situation representation

6.2 Service level

At this level, our model allows describing the behavior expected when LLA detects situations. In order to react properly to these situations, the application needs to find the appropriate services. In LLA, a service is a composition of mobile software components controlled by Kalimucho middleware to ensure their communication and sustainability. Each component is defined by its specific requirements (needs of hardware).

The service composition and requirements are specified by the developer of the service and added in the description of the service in a 'Services description & requirements' file in LLA. The description is presented, both textually and graphically, as follows:

- Graphically: The graphical representation is used to have a clearer understanding of the composition strategy for the service.
- Textually: A description for a service is presented textually in the following format:

  Service: [Input [Component (Component requirements &,||...); ...]; Core [...]; Output [...] |Links (ComponentN ->ComponentN+1;...)]

In the textual representation, the S! prefix is used to represent a static requirement, that cannot be changed, whereas the D! prefix represents a requirement over a dynamic resource. For example, "S!RAM > 4 GB" means

that the RAM should have more than 4 GB of physical memory, whereas "D!RAM > 4 GB" means that there should be more than 4 GB of RAM available at that moment. The use of a D! or S! prefix is optional. In our context, the "!" character means the need of a resource and not the opposite (e.g., "!Display" means that the device needs to have a screen).

For example, a video chat service that uses our modular composition would contain five components: Picture feed component, Sound feed component, Text feed component, Video chat core component, and Video chat UI component. Textually, this service is presented in Figure 5 and graphically in Figure 6

Video chat service: [Input [Picture feed component (!Camera); Sound feed component (!Microphone); Text feed component ((!Keyboard)||(!Display))]; Core [Video chat core component (D!RAM>1GB)]; Output [Video chat UI component (S!Display>10inches);] |Links (Picture feed component -> Video chat core component; Sound feed component -> Video chat core component; Text feed component -> Video chat core component; Video chat component -> Video chat UI component;)]

Fig. 5 Video chat service description

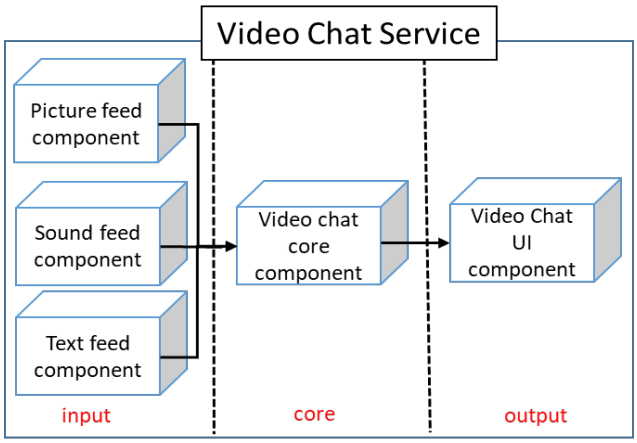

Fig. 6 Video chat service composition using the graphical representation

These components are linked to each other via Kalimucho connectors (Da et al., 2014), yet work separately on their specific tasks in order to define the overall behavior of the service. Naturally, these components have hardware requirements that the devices need to provide in order for the service to be executed in a functional state. LLA orchestrates these components inside the user domain by considering their requirements in order to ensure the sustainability and correct functioning of the service.

6.3 Matching services to situations

In LLA, for each situation, a mapping is considered. The mapping matches situations to services in order to respond to contextual changes when needed. Using the service composition and component requirement concepts, the matching defines clearly the needs of the service in terms of the optimal distribution. A mapping M is presented textually in the following format, where P is a projection and Sn is a situation:

$$M : Sn\ [\ P(1..n)\ [\ Service;\ \ldots\ ]]$$

These textual representations are translated into XML and injected into the user's application in order to define the required responses to the detected situations. Having defined the service layer, we have the full description of the situation (context and service).

For example, in Figure 7, a full graphical description is shown. This representation illustrates an example of an important monthly work meeting situation done remotely by using a video chat service.

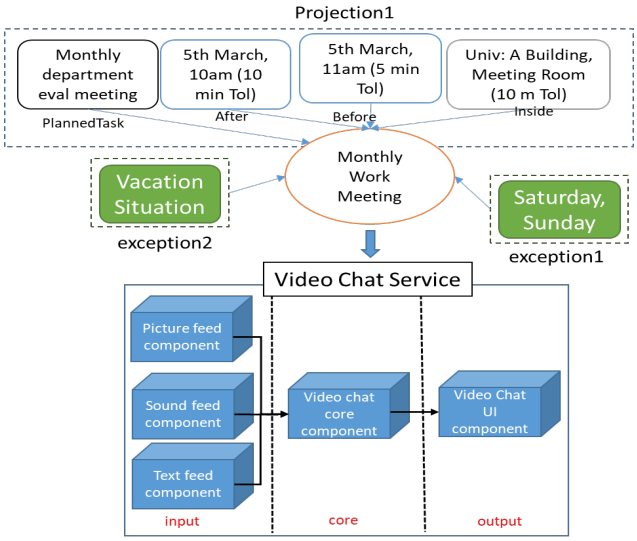

Fig. 7 Monthly work meeting – full representation

Using the same video chat service example (see Figure 6), the textual representation of the mapping is as follows:

M1: Monthly work meeting [Projection1 [Video chat service]]

This indicates that, when the monthly work meeting situation is detected and is about to start, LLA will deploy the video chat service. If the situation is ending, LLA stops the service by stopping all its components.

7 Use case and prototype

In this section, we show the interest of the proposed solution from two different perspectives. The first one implies assessing the benefits of the situation model; for that purpose, we apply the solution to a real-life use case, in order to prove the richness and dynamicity of the proposed situation model. The second one implies comparing it with the currently-used approaches based on app stores (downloading multiple apps from an application store manually, on demand), to show its robustness and the improvements that it provides over those traditional alternatives.

7.1 Real-life use case

As an example, five situations that could occur daily to any user are considered. For that, the first step is to inject these situations into his/her situation base. In Figure 8, we show how the user's situations are represented textually. A, B, and W represent primitives on the Time axis meaning *After*, *Before*, and *While*, respectively. So, for example, *A (22,20)* means *After* the hour 22 (10 pm) with a 20 minutes tolerance. O and I represent the Location's axis primitives *Outside* and *Inside*, respectively. Finally, PT means *Planned Task*. Pn refers to the name/number of a projection.

*__Home Security Alarm situation (S0)__: P0[Time(A(22,20)B(6,20)); Location(I(Home,5))] OR P1[Location(O(Home,20))] EXCEPT (Time(Sunday))
__At Home situation (S1)__: P0[Location(I(Home,5))]
__At lunch situation (S2)__: P0[Time(A(12,0)B(14,0)); Location(I(Home,10))] EXCEPT (Location(Home))
__Meeting situation (S3)__: P0[Time(A(15,0)B(17,0)); Location(I(MeetingRoom,2))]
__Work Situation (S4)__: P0[Location(I(Office,5); Activity(PT("OfficeWork"))] EXCEPT (Situation(holidays))

Fig. 8 Five situations in a real-life use case

The next step consists of injecting mappings in order to link the situations to the services (see Figure 9).

*__M0__: Home Security Alarm Situation (S0) [P0 [Security Alarm Service]; P1 [Distant Security Alarm Service]]
*__M1__: At Home Situation (S1) [P0 [Home Control Service]
*__M2__: At lunch Situation (S2) [P0 [Restaurant Service, Social Feed Service]]
*__M3__: Meeting Situation (S3) [P0 [Meeting Service]]
*__M4__: Work Situation (S4) [P0 [Work Service]]

Fig. 9 Mappings injected to link the situations with services in the real-life use case

Then, the injected services must be described. Those descriptions (see Figure 10) enable LLA to scan the user's devices in order to find which one/s

has/have the required capabilities to host the components of the service. The executables (Java jar files) of the services are downloaded into the LLA of the user. Finally, in Figure 11, we show the generated situations, mappings and the provided services for the examples described in Figure 8.

*__Meeting Service__: [(Input [Picture feed component (!Camera); Sound feed component (!Microphone); Text feed component ((!Keyboard)||(!Display))]; Core [Meeting core component (D!RAM>2.5GB)]; Output [Image output component (S!Display>16inches); Text UI component (S!Display>10inches); Sound output component (S!Audio<40dB)] |Links (Picture feed component ->Meeting core component; Sound feed component ->Meeting core component; Text feed component ->Meeting core component; Meeting core component ->Image output component; Meeting core component ->Sound output component; Meeting core component ->Text UI component;)]
*__Security Alarm Service__: [Input [Video Surveillance component ((!Camera)& (S!Network=“Wifi”); Fire detector component(!Smoke-sensor)]; Core [Security monitoring component (D!RAM>1GB)]; Output [Security status component (S!Display<5inches)] |Links (Video Surveillance component ->Security monitoring component; Fire detector component ->Security monitoring component; Security monitoring component ->Security status component;)]
*__Restaurant Service__: [Output [Menu component (!Display)]]
*__Social Feed Service__: [Output [Feed Wall component (!Display)]]
*__Work Service__: [Output [Work component (!Display)]]
*__Home Control Service__: [Input [Remote Control component ((S!Display>=5inches)& (S!Network=“Infrared”||“Bluetooth”||“Wifi”))]; Output [Status component (S!Display>28inches)) |Links [Remote Control component ->Status component]

Fig. 10 Description of the services in the real-life use case

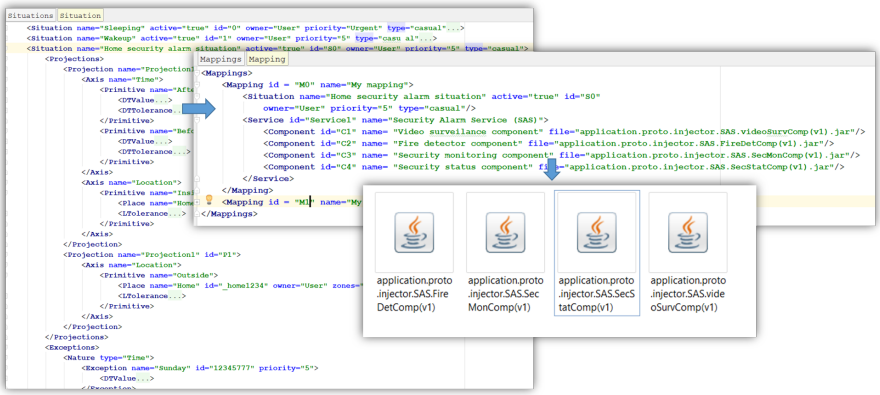

Fig. 11 Generated files and provided services for the real-life use case

The management of the described scenario involves many different mechanisms. Thus, we present in this section the results obtained after implementing the proposed architecture (Karchoud et al., 2017a) and testing it on an Android device (Samsung Galaxy S4 running Android 5.0.1 on an ARMv7

processor and 2 GB RAM). As a proof of concept, we present a video of the LLA's functioning prototype available at `https://www.youtube.com/watch?v=GRVv_bsiB_s&t=15`.

For scalability evaluation purposes, in the developed prototype, we have included also a generator that can create situations for testing. We injected the considered scenario described above into the situation generator, and we ran the application while simulating the location, time and activity of the user. Specifically, for this experiment, we set up a time-frequency for contextual data extraction of 20 seconds and a location frequency of 10 meters (i.e., the data extraction is triggered when 20 seconds have passed or when the user has moved at least 10 meters). On the user's device, all the other apps (except the mock location app) were disabled, the GPS was activated, the Wi-Fi was activated, and the battery was plugged.

Then, we performed a scalability experiment for different numbers of situations (using the generator). The results obtained show that our solution (LLA) is able to handle and monitor between 1000 at 10000 situations simultaneously. The device on which we performed this experiment logged the following metrics: the average CPU usage when the app was launched and while it was running later, the average memory usage while the app was running, and the response time value until the app crashed (for more details, see (Karchoud et al., 2017a)). The results obtained showed that our proposal is stable overall. When the number of considered situations increases, a critical point is not reached until having a very large number of situations to detect (in our experiments, above 10^3), which is highly-improbable for everyday normal usage.

7.2 Comparison with a classical approach

Using LLA, users do not have to download applications on all the devices that they own. Instead, the injection mechanism allows the application to download or synchronize (between the cloud and the user's devices) only the required functionalities, by automatically installing, updating, deleting, and organizing the injected files and the required services. This implies a considerable amount of data and memory gain compared to the classic approach of downloading applications on all the devices, just in case they are needed. This means that, for example, without LLA, if the user wants to use Google Maps on any of his/her devices at any given time, then he/she has to download/install it on all his/her devices. With this old approach (based on the use of app stores), the apps must be updated also on all the devices, in order to keep the different instances of the apps running the same up-to-date version, even for the slightest change in code. On the contrary, in LLA, component-based services require updates to be done regardless of the device. Moreover, an update targets only one specific component separately (the one that changed), without touching the rest of the components composing the service. This is performed in a

centralized way, inside the user's cloud persistence layer (as it was illustrated in Figure 3).

To prove this, we considered two users: User 1 and User 2. User 1 made use of LLA and User 2 followed the current approach of mobile stores and standalone applications. We considered that both users possessed 10 devices each. Even though 10 devices per user may seem a large number nowadays, the number of personal mobile devices that a user owns (mobile phone, laptop computer, tablet, smart-watch, Smart TV, etc.) is expected to increase considerably in the next few years. Indeed, according to (Evans, 2011), with the growth of IoT, it is estimated that by 2020 the average will be already 6.5 devices per user.

We compared the behaviour of both users by using an estimation of the storage size and time required for the different operations that they both had to perform (install/update/delete). We considered the situations of Figure 10 for both users. User 1 followed the LLA's workflow (shown previously in Figure 1) to handle those situations; therefore, this user was actually using the LLA's services (see Section 6.2). User 2 relied on the current traditional approach and therefore used standalone applications (downloaded from a store) that provide similar functionalities to the services described in Figure 10 (e.g., Meeting service = Skype). This means that, in order to perform a fair comparison, the LLA's services will be considered equal in storage size to the similar standalone applications that were selected. In Table 1, we show the list of selected apps and their sizes. We considered the average sizes of the apps across different operating systems, since the size of an app varies according to the operating system on which it is installed; for example, Skype on iOS requires an estimated size of 103 MB but it requires 70 MB for a Debian GNU/Linux distribution. The apps considered were: Skype (`https://www.skype.com`), the Reolink app (`https://reolink.com/software-and-manual/`), the McDonalds's app (`https://www.mcdonalds.com`), the Twitter app (`https://twitter.com/download`), Microsoft Excel (`https://products.office.com/en/excel`), and Nest (`https://nest.com/app/`).

Table 1 LLA's services and the equivalent apps, along with their average sizes

LLA Service	**Standalone app**	**Average size**
Meeting service	Skype	143.5 MB
House alarm service	Reolink app	4.5 MB
Restaurant service	McDonalds's app	110 MB
Social feed service	Twitter app	84 MB
Work service	Microsoft Excel	1037 MB
Home control service	Nest	202 MB
	Total	1581 MB

These calculations were done on March 3, 2018, by considering the latest versions of the selected standalone apps. For LLA, the total storage size really used was calculated by adding the size of the basic LLA application (in our current prototype, the size of the core components of LLA is 10 MB) to the

size of the services. This is needed because the core of LLA has to exist on all the devices at all times, in order to ensure the continuity and management of the services (stored in the cloud layer), that is, to support the required LLA's functionality. Therefore, this core is duplicated on all the devices of User 1 (10 devices, in total)

*Overall storage size using LLA = (Size of the LLA's core components * Number of devices) + Size of the services = (*10 *MB* * 10 *devices)* + 1581 *MB* = 1681 *MB*

For the *Classical Store Approach (CSA)*, User 2 has to download all the applications on all the devices to ensure the continuity of usage (e.g., to continue a video call after leaving his/her house, the user has to install Skype on his/her computer, portable TV receiver, smart car, etc.).

*Overall storage size using the classical approach = Number of devices * Average size of the applications* = 10 *devices* * 1581 *MB* = 15810 *MB*

Besides the storage size needed, we have also to consider the time invested by the user with both approaches. For the LLA approach, this is the time taken by User 1 to download and install LLA, configure it, update it, and delete it (in case the user decides in the future to stop using it). For the classical store approach, it represents the time that User 2 requires to download, install, update, and delete all the different applications on all the devices (in our example, 10 devices in total). For these calculations, the tests were done using the Samsung Galaxy S4 device, mentioned before, with an average Internet download speed of 4.29 MB/s. The results are shown in Table 2 and the basic computations performed are shown in the following:

- Total download time (LLA and CSA) = Overall size / Internet average speed
- Total installation time (LLA) = Installation time of LLA * Number of devices
- Total installation time (CSA) = Installation time of each standalone app * Number of standalone apps * Number of devices
- Total delete time (LLA) = Delete time of LLA * Number of devices
- Total delete time (CSA) = Delete time of each standalone app * Number of standalone apps * Number of devices
- Total update time (LLA and CSA) = Total delete time + Total download time + Total installation time

The update is always the heaviest process. Although we consider the same formula for the update time in LLA and CSA, in our approach this is the worst-case scenario because in LLA the updates could be performed separately in a centralized way (cloud) targeting only the affected components. To perform a fair comparison, we have not considering the potential cost of component migration with LLA (components can be transferred between devices when

needed). The reason is that no existing app offers this migrating feature. It should be noted that with the traditional approach (CSA) an app will not be able to offer a certain functionality (e.g., a functionality that requires access to a microphone) if it is not enabled (e.g., there is no microphone) on the device where the app is executing. This situation would be equivalent in LLA if we disable the possibility of migrations. In the worst case, with LLA, if all the components need to migrate to all the devices of the user, then the cost of migrations would approximate the total download time cost of the CSA approach, but this is highly unlikely; besides, a device has a local repository that stores components locally and can transfer them via local networks, which is most of the time considerably faster and more reliable than using the Internet.

Table 2 Comparison of LLA (Long-Life Application) and CSA (Classical Store Approach)

	User 1 (LLA)	User 2 (CSA)
Overall size	1.681 GB	15.81 GB
Total download time	06m:32s	28m:05s
Total installation time	03m:50s	33m12s
Total delete time	00m:20s	01m:24s
Total update time (delete+download+install)	10m:42s	1h:02m:41s

Notice that the LLA's services are downloaded into the user's local repository, but they are not installed. They are dynamically deployed and they join the main execution thread of LLA dynamically while it is running. Overall, the results described in this section show that using the LLA approach can lead to a considerable amount of time and storage gain. Our approach is able to provide these results by using the collaborative injection mechanism that centralizes and contextualizes the user's services across his/her devices.

8 Conclusions and future work

Proposing only ONE application per user, to perform all his/her tasks and manage transparently all the functionalities, is an ideal solution not available nowadays. Nevertheless, a future where mobile apps will know what we need, even before we interact with them, is approaching. In this sense, many research works are oriented to the development of those future apps and they found out that contextual awareness is essential to the survival of mobile applications.

Our proposal for this is called Long-life Application (LLA), a name that reflects the nature of the continuity and evolution that is needed. This application will evolve while running according to the users' needs (considering his/her personal and/or professional requirements), and will continuously provide relevant context-adapted services.

In this paper, we focused on the problem of dynamic situation injection. LLA incorporates a dynamic injection mechanism that allows users, developers, and other entities, to expand the application and customize it according

to the user's specific needs. This contribution represents one main novelty of our proposal in the current mobile context-aware world. Compared to the currently-used mobile store system of multiple standalone applications, our solution targets specific services towards the users only when they need them. These changes will push developers to think in terms of context awareness, and therefore to provide a more personalized user experience. From the point of view of the user, LLA gives the possibility to have a richer, open, and hands-free experience.

An aspect that could be considered to improve the dynamicity of the LLA approach would be to integrate an intelligent situation recommender (e.g., based on machine learning techniques, such as deep learning approaches trained with large amounts of data) that digs deeper into the personality of the user in order to recommend more customized situations.

Acknowledgment

This work was supported by the Embassy of France in Spain and by the projects TIN2013-46238-C(1/4)-4-R, FEDER/TIN2016-78011-C4-(2/3)-R (AEI/FEDER, UE), FEDER/TCVPYR, and DGA-FSE (COS2MOS group).

References

24me (2016) 24me. `https://www.twentyfour.me`, [Online; accessed 5-March-2018]

Basiri A, Amirian P, Winstanley A, Moore T (2017) Making tourist guidance systems more intelligent, adaptive and personalized using crowd sourced movement data. Journal of Ambient Intelligence and Humanized Computing pp 1–15

Boyaci O, Beltran V, Schulzrinne H (2010) Bridging communications and the physical world: Sense Everything, Control Everything. In: GLOBECOM Workshops, IEEE, pp 1735–1740

Chen CC, Huang TC (2012) Learning in a u-museum: Developing a context-aware ubiquitous learning environment. Computers & Education 59(3):873–883

Chihani B, Bertin E, Crespi N (2013) A user-centric context-aware mobile assistant. In: 17th International Conference on Intelligence in Next Generation Networks (ICIN), IEEE, pp 110–117

Clarke S, Driver C (2004) Context-aware trails [mobile computing]. Computer 37(8):97–99

Da K, Dalmau M, Roose P (2014) Kalimucho: middleware for mobile applications. In: 29th Annual ACM Symposium on Applied Computing (SAC), ACM, pp 413–419

Driver C, Clarke S (2004) Hermes: generic designs for mobile, context-aware trails-based applications. In: Workshop on Context Awareness at MobiSys

Edison (2017) Easilydo. https://play.google.com/store/apps/details?id=com.easilydo, [Online; accessed 5-March-2018]

Endsley MR (1995) Toward a theory of situation awareness in dynamic systems. Human factors 37(1):32–64

Evans D (2011) The Internet of Things: how the next evolution of the internet is changing everything. White Paper by Cisco Internet Business Solutions Group (IBSG)

Farghaly A, Shaalan K (2009) Arabic natural language processing: Challenges and solutions. ACM Transactions on Asian Language Information Processing (TALIP) 8(4):14

Google (2017) Google Inside Search. https://www.google.com/intl/es419/insidesearch/features/search/knowledge.html, [Online; accessed 5-March-2018]

Grennan T (2016) Spring 2016 mobile customer retention report an analysis of retention by day. Tech. rep., Appboy

Harter A, Hopper A, Steggles P, Ward A, Webster P (2002) The anatomy of a context-aware application. Wireless Networks 8(2/3):187–197

Karchoud R, Roose P, Dalmau M, Illarramendi A, Ilarri S (2016) Long Life Application: Approach for user context management and situation understanding. In: International Conference on Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS), IEEE, pp 45–53

Karchoud R, Illarramendi A, Ilarri S, Roose P, Dalmau M (2017a) Long-life application – situation detection in a context-aware all-in-one application. Personal and Ubiquitous Computing 21(6):1025–1037

Karchoud R, Roose P, Dalmau M, Illarramendi A, Ilarri S (2017b) All for one and one for all: Dynamic injection of situations in a generic context-aware application. Procedia Computer Science (IUCC-CSS) 113:17–24

Lardinois F (2017) Google starts testing Instant Apps in the wild. https://techcrunch.com/2017/01/23/google-starts-testing-instant-apps-in-the-wild, [Online; accessed 5-March-2018]

Martin C (2017) How to use Google Assistant. http://www.pcadvisor.co.uk/feature/google-android/how-use-google-assistant-google-now-3574727, [Online; accessed 5-March-2018]

Nakagawa T, Doi C, Ohta K, Inamura H (2012) Customizable context detection for ECA rule-based context-aware applications. In: Sixth International Conference on Mobile Computing and Ubiquitous Networking (ICMU), Information Processing Society of Japan, vol 30, pp 98–105

Ovadia S (2014) Automate the Internet with If This Then That (IFTTT). Behavioral & Social Sciences Librarian 33(4):208–211

Rodgers E (2013) Tempo for iPhone uses AI to fold maps, contacts, and files into your calendar. http://www.theverge.com/2013/2/13/3982656/tempo-intelligent-calendar-for-iphone, [Online; accessed 5-March-2018]

Sang-Seok Y, Quang N, JongSuk C (2017) Recognition of emergency situations using audiovisual perception sensor network for ambient assistive living. Journal of Ambient Intelligence and Humanized Computing pp 1–15

Syer MD, Adams B, Zou Y, Hassan AE (2011) Exploring the development of micro-apps: A case study on the Blackberry and Android platforms. In: 11th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE, pp 55–64

Zhao Z, Liu J, Crespi N (2011) The design of activity-oriented social networking: Dig-Event. In: 13th International Conference on Information Integration and Web-based Applications and Services (iiWAS), ACM, pp 420–425