Foundations for a Platform to Develop Context-Aware Systems by Domain Experts

David Martín, Carlos Lamsfus, Aurkene Alzua
Human Mobility and Technology
CICtourGUNE
Donostia - San Sebastián, Spain
davidmartin@tourgune.org

Diego López de Ipiña
Deusto Institute of Technology - DeustoTech
University of Deusto
Bilbao, Spain
dipina@deusto.es

Abstract—There are several toolkits in order to develop context-aware systems. Most of them are for programmers, providing high-level APIs in order to manage context data. Therefore, people that do not have programming skills but have the necessary knowledge about the application domain where the system is going to be deployed cannot participate in the development process. One of the main tasks in the development of context-aware systems is the definition of user situations, which are best defined by people with expertise in the application domain. The involvement of domain experts can improve the final product and it can speed up the development process. This article presents the theoretical foundations for a toolkit in order to promote the involvement of domain experts in the development of context-aware systems. Based on these requirements, a platform to develop such systems has been implemented and it has been validated with domain experts of the tourism domain.

Context-Aware Middleware; End-User Programming; Web of Things; Mobile Services

I. INTRODUCTION

In the near future, all the everyday objects will have an embedded computer and they will be connected to the Internet, providing personalized services to users at any time and place [1]. This personalization is essential for an optimal user experience.

Nowadays, this vision is real in mobile environments. Our smartphones are connected to the Internet and they can provide us with information and services everywhere. But this mobile environment is very different from a desktop one. For instance, the screen is smaller, users are on the move and they usually require very specific information at a given time and place. This way, the customization of information becomes even more important in mobile scenarios.

One of the possible ways to provide a better user experience is to develop systems that can obtain data about the user’s context, process them in order to identify the user’s situation and finally adapt the functionality of the system to the identified situation.

However, the development of context-aware systems is difficult for programmers, because there are several technical challenges that have to be faced: context sources have to be identified, information has to be obtained from these heterogeneous and distributed sources, data has to be modeled in order to be processed by computers, the situation of the user has to be identified using reasoning engines, and finally, the service has to be adapted to the identified situations.

Apart from that, it can be difficult for a programmer to identify the needed situations and the desired behaviors of the system to be developed once a situation is detected, because they are usually dependent on the application domain (e.g. tourism, automotive industry, smart home). Domain experts, that is, people that are experts in the domain where the application is going to be deployed, can provide the needed expertise regarding the above requirements.

Over the last decade, several toolkits and programming frameworks have been proposed in order to simplify the development of context-aware systems [2]. However, there are still some gaps in the reviewed frameworks. On the one hand, not all of them are designed to support users’ mobility, and do not provide the user with a Geographic Information System (GIS) in order to perform geospatial operations. On the other hand, these toolkits are configured using high-level programming APIs for skilled programmers. This way, non-technical domain experts cannot be involved in the development life-cycle.

This paper presents the foundations for a platform in order to promote the involvement of domain experts in the development of context-aware systems. These foundations are based on the gaps identified in the state of the art. The implemented solution is also described, which is called Context Cloud. The platform provides a web front-end where all the features involved in the development of context-aware services can be easily configured without writing any line of code. For instance, the context data model, the context information gathering process and the rules to identify users’ situations can be configured using a web interface. In this manner, the involvement of domain experts in the development process is possible.

The paper is organized as follows. In section 2, the related work is presented. In section 3, the theoretical framework is explained. In section 4, the requirements analysis is described. In section 5 the implementation of the platform is detailed. Section 6 presents the evaluation of the platform. Finally, section 7 concludes the paper with brief concluding remarks.

II. RELATED WORK

The first implemented framework to ease the development of context-aware systems is the Context Toolkit [3]. This framework presents an architecture composed of different functional modules in order to acquire, aggregate
and interpret context information that is modelled using key/value pairs. Other approaches like CASS [4] propose a layered middleware architecture that uses a relational data model to represent context data. JCAF [5] is a framework and a runtime environment to develop and deploy contextual computing applications. These approaches use interpreters to convert acquired raw data into higher level context data, but these transformations cannot be very complex because there are no inference mechanisms.

SOCAM [6] and Semantic Spaces [7] are also frameworks based on three different layers, namely a sensing layer, a middleware layer and an application layer. The CoBra [8] middleware proposes a different approach where software agents are used in order to acquire and process context data in a smart meeting room environment. All these toolkits are based on ontologies in order to represent context data.

The main drawback of the above presented toolkits is that they can only be configured using high-level APIs. This way, only people with technical skills can use them in order to implement context-aware systems.

Some authors have proposed visual approaches where domain experts can be involved in the development lifecycle. The iCAP toolkit [9] is a visual editor where domain experts can prototype context-aware systems using rules. DiaSuite toolkit [10] comprises a domain-specific design language, a compiler for this language and an editor to define simulation scenarios. The OPEN framework [11] is an ontology-based programming environment for rapid prototyping of context-aware applications. It is based on the configuration of semantic rules in order to trigger predefined actions.

Although these toolkits have been designed in order to involve non-technical users in the development process, they still have some drawbacks. For instance, the iCAP toolkit does not have functionalities in order to define a context data model and it cannot be extended. Apart from that, the rules that can be configured are simple. DiaSuite toolkit has a desktop visual programming environment, where simulation scenarios can be graphically defined. However, the provided DiaSpec language, which is similar to Java, can be quite difficult for a non-technical user because programming skills are required to define those artefacts. The OPEN framework uses a semantic model to represent context data that can only be extended by skilled programmers.

Context Cloud differs from the above toolkits in the following respects. It is designed to promote the involvement of non-technical users in the development life-cycle. It provides a web environment where context data can be managed using a graphical interface without having to code any programming line. It also provides geospatial functionalities to manage location context data and all the configurations can be extended at runtime.

III. THEORETICAL FRAMEWORK

The conceptualization of the notion of context and the situations that can be inferred based on these data represent the theoretical basis of this research work.

The notion of context has been widely studied and there are several authors that have proposed different definitions of context. Some of these definitions consider context as the surroundings of the interaction between the user and the application [12] [13] [14]. Other authors consider the activity or the task of the user as the main context information for the system [15] [16]. A third group of authors consider that context is the needed information to characterize the situation of an entity [17] [3].

In this research work, context will be considered from a computing perspective, having into account the third group of definitions mentioned before. This way, context is considered as any information that can be obtained and processed by hardware or software systems, in order to identify the situation of an entity (a living being, a place or an object) and adapt the system’s behaviour to that situation.

Apart from that, there are several definitions for the concept of “situation”. These definitions have something in common: they consider context as low-level data, while a situation is high-level data. This way, a situation is dependent on the context information and it can be considered as an abstraction of it [18] [19] [20].

In the scope of this research work, a situation is defined as the state of the current and past context at a certain region in space and a concrete interval in time that are relevant to identify that situation. There are two main principles that have been taken into account for this definition. The first one is the notion of time. A situation can have temporal boundaries and it can be related to past or current situations. The other one is the notion of space, that is, where the situation can be identified. For instance, the situation “cooking” could be detected when the user is in the kitchen (space) and it is time to have lunch (time).

Finally, a context-aware system is defined as a system that reacts to the situations that have been identified based on low-level context data [21].

IV. REQUIREMENTS ANALYSIS

In order to design the platform, several requirements have been identified, which can be considered as the foundations for a platform to develop context-aware systems by domain experts.

A. Architecture

The architecture is based on the common layers identified in the state of the art [2]: sensor layer, data gathering layer, data processing layer, data management layer and application layer. Even so, some advances in complementary scientific fields have been identified that can be applied to improve the architecture of these kinds of toolkits. This way, two main ideas have been taken into account in the design stage of the architecture: the Web of Things [22] and End-User Programming [23] paradigms.

The Web of Things principles can be applied to the sensor layer. This way, sensors must have a web end-point in order to get data from them and to interact with them in a RESTful way. In this manner, these principles can provide a higher level of abstraction over the drivers and low-level
mechanisms that these sources expose in order to get data from them.

The End-User Programming paradigm has to be considered in order to involve domain experts in the development process. The users of these toolkits must have the knowledge about what is happening behind the scenes and they must have the control over the managed context data [24].

As mentioned before, context-aware systems are considered as reactive systems that adapt its behavior to the detected situations of the users. This way, the platform has to be designed in order to generate the needed high-level outputs that will be used by the context-aware systems in order to have a reactive behaviour. There are several context processing patterns that have been identified in the literature review [25], but they are not applicable to a scenario where domain experts configure the system in order to manage context information and produce high-level outputs. In order to solve this gap, the pattern shown in Figure 1 is proposed.

![Figure 1. Proposed context processing pattern](image)

In this pattern, context data is obtained or received by the platform and it produces several outputs with information about the identified situations. Also, the generated outputs can be used as high-level inputs. The outputs (situations) can be modeled using rules that domain experts need to configure based on their knowledge on the application domain.

B. Data model

In order to store the received context information a context data model is needed. This way, the platform can manage data more effectively. There are several data models that can be used in order to manage context information [26], but these have to fulfill the following requirements [27]:

- Heterogeneity: the context data model has to represent data coming from very different data sources, which usually provide information in a heterogeneous way.
- Dependencies and properties: the relationship between different context entities and their properties have to be modeled.
- Inference: reasoning engines or inference systems have to be applied over the specified data model. This way, high-level information (situations) can be detected.
- Flexibility: the defined model has to be flexible enough in order to be extended in runtime. This way, the highly dynamic requirements of these kinds of systems can be supported.

- Spatial representation: location is one of the main context parameters. The model has to provide mechanisms in order to manage this information. Apart from that the model has to have functionalities in order to manage areas or regions in space where the situations can be identified.
- Time: the temporal boundaries when a situation can be detected are also relevant for the context model.

C. Reasoning

As mentioned before, a reasoning engine is needed to infer high-level context and to detect situations. The reasoning engine has to be powered by rules defined by domain experts. Two of the main requirements of the reasoning engine are spatial and temporal reasoning support. Spatial reasoning is needed in order to trigger rules attached to previously configured regions in space where certain situations can be detected. Temporal reasoning is also needed in order to support Allen’s temporal logic [28].

D. Automatic context data life-cycle management

The management of context data involves several tasks. Data has to be transformed into the defined data model, the instances of the model have to be inserted or updated in the knowledge base and data coming from different sources have to be aggregated if they are related to the same entity instance. This management can be quite repetitive and mechanisms that can provide automations are needed.

E. Extensibility

The platform needs to be flexible enough in order to be extended at runtime [29]. In such dynamic environments new context sources can be required in order to identify new situations. This way, the data model of the platform and the defined rules need to be extended according to the new context data requirements at runtime.

F. Mobility

As mentioned before, location is the main context parameter to be considered in order to personalize the behaviour of context-aware systems [30]. The entities that can be involved in a certain situation can be on the move (e.g. Person, Car), so the architecture of the platform has to integrate a GIS service in order to manage their location. Also, this GIS service is needed in order to manage the areas where situations can be identified and to detect the entities that are located at those areas.

G. Web development environment

The platform has to be a web application that can be configured at any time with any connected device. It needs to provide the user with a visual environment in order to manage the context data using the user interface [31]. Like that, domain experts can easily access and modify all the configurations in real time without having to install any development environment.
V. IMPLEMENTATION OF THE PLATFORM

The architecture of the platform is based on the previous requirements. Figure 2 shows this architecture, which is divided into four different layers: sources, providers, management and application. The platform can be considered as a black box that receives inputs from the identified context sources and produces outputs triggered by the defined context rules. The programmer can then use the generated outputs in order to adapt the final service to be developed.

A. Sources

The first layer is where all the context sources are (e.g. sensors, web services, mobile devices). These sources must be connected to the Internet and they must provide context data in XML or JSON format. The programmer has to adapt the identified sources in order to be accessed in a RESTful way, following the best practices established by the community of the Web of Things.

B. Providers

Providers are part of the Context Manager, which manages the context data life-cycle. Providers are software components that obtain or receive data from the identified context sources. There are two types of providers: active and passive. Active providers can be configured in order to make periodic GET requests to the selected context sources and get a document with the context data in response. Passive providers can receive data from sources that make POST requests containing XML/JSON context data. The obtained data is automatically processed by the platform.

C. Management

The Management layer is also part of the Context Manager. It is composed of several modules. The Context Model Store can store a context data model based on context entities (e.g. Person, City) and different typed properties (e.g. name, temperature) that are internally modeled using Java Bean classes. This data model will be populated by the received context data and transformed into context entities that are going to be saved in the Knowledge Base.

Figure 3 shows the context entity creation dialog. Here, the entity name, the description and the properties can be configured. Every entity must have an identifier property in order to aggregate data coming from different sources that is related to the same entity instance. An entity can also be marked as a geo-referenced entity, which means that it will contain latitude and longitude coordinates that are going to be managed by the platform’s GIS component. The created context data model can be modified and extended at runtime.

The transformation between data coming from the sources and the data model is performed by the Mapping Engine, according to the user’s defined mappings. In order to define a new mapping, the source data property, the entity and the entity property have to be specified. Every data that is received and transformed into an instance of an entity is assigned a timestamp. This way, rules can use temporal operators (e.g. after, before) in order to manage these data.

The GIS component is used to store the created areas where situations can be identified. It also translates location coordinates of context entities into registered area names.

The situations can be managed using the rule creation dialog of the platform. In order to create a new rule, it must be assigned to a previously created area. The Rule Engine has been implemented using the Drools Expert System\(^1\) and it has been improved with new functionalities in order to be integrated with the GIS component.

Figure 4 shows some of the widgets that can be used to configure the rule conditions. A calendar option is available, where a predefined date range can be specified for the rule (weekday, weekend). The dialog shows all the available entities, their properties and the conditions that can be applied to these properties. The platform automatically translates these configurations into code that can also be edited programmatically.

\(^1\) [http://www.jboss.org/drools](http://www.jboss.org/drools)
Also, some useful functions can be used in order to configure the actions of the rule once it is triggered. For instance, the “POST” function can be included in the consequence of a rule in order to trigger a POST request to any external web service endpoint. The POST request can contain in the payload any of the entities that are involved in the rule as XML or JSON data.

For instance, the following code has been automatically generated configuring the needed conditions in order to detect that a visitor is “sunbathing” (speed==0) at the “Zurriola” beach with a very high temperature (>=35) for the city of “Donostia”. The rule will send a POST request with the visitor’s phone number to the configured web endpoint. The service will send an alert to the visitor’s smartphone.

When
$\text{Visitor( speed == 0)}$
$\text{Area( name == "Zurriola") from Visitor.currentAreas}$
$\text{City( name == "Donostia", temperature >= 35)}$

Then
doPOST("http://my.server/post", $\text{Visitor.phoneNumber});$

D. Application

Finally, the upper layer is the application layer. Here is where all the systems that use the produced high-level outputs are. These systems can use these outputs in order to react and personalize the behaviour of the service.

VI. USER EVALUATION

The platform has been validated in a controlled laboratory environment. A total of 12 participants carried out the evaluation in pairs composed by a domain expert and a programmer in order to help the domain expert if needed. The evaluation was inspired in a tourism scenario, so the non-technical users were experts in the tourism domain.

A. Methodology

First of all, the users were introduced to the Context Cloud platform and to the experiment’s objectives. They were instructed on how to configure the platform and they were given an example on how to identify a situation.

The participants were given a text document where four different situations were described.

- Waiting for the bus (S1): the visitor waits for the bus at Bus Stop A and she receives an SMS with the estimated time of arrival for the next bus.
- Sunbathing (S2): the visitor is sunbathing at the beach when she receives an SMS advising her that she should use sun cream because the temperature is higher than 30°C
- Waiting for the bus (S3): the visitor waits for the bus at Bus Stop B and she receives an SMS with the estimated time of arrival for the next bus.
- Arriving to the hotel room (S4): the air conditioning is activated when the visitor goes into the room.

In order to simulate the situations, the Siafu Context Simulator\(^2\) was used. It was configured in order to send context data about the visitor on the move to the platform. The simulator also provided some web services in order to obtain the weather information and to access the air conditioning system of the Hotel.

The participants had to configure the platform in order to print a message in the debugging console once a situation was detected. During the test, an external observer annotated all the problems that the participants found using the platform. Also, once a situation was detected, the time spent in its configuration was annotated.

After having completed the user experience, each participant had to fill out a questionnaire based on a six-level likert scale, with values from 1 (totally disagree) to 6 (totally agree). The used survey was designed on the Technology Acceptance Model (TAM) literature, and in particular, it was adapted from David’s studies \[32\]. This way, three constructs were considered: Perceived Ease of Use (PEOU), the Perceived Usefulness (PU), and the Behavioral Intention (BI).

B. Results

These results are focused in the answers given by domain experts. The 83% of the participants find that learning how to use the platform is easy. The 50% of the participants find it easy to get Context Cloud to do what they want to do. However, the other 50% disagree on that. The reason is that the participants were not used to work with these kinds of toolkits. The 83% of the participants also find that the interaction with Context Cloud is clear and understandable. The 83% of the non-programmers state that it would be easy for them to become skillful at using the platform.

The perceived utility of the platform is also highly supported by domain experts. The 100% of the domain experts state that using Context Cloud in their jobs would enable them to develop context-aware systems more quickly and that it would make it easier to develop context-aware systems. Also, all of the participants would recommend other users to use the platform and they would use it in future developments. In addition to this, the 67% of them would pay for the system.

\(^2\) http://siafusimulator.sourceforge.net/
The average time spent by each of the pairs to solve the evaluation test was 90 minutes. It is relevant that the they spent an average time of 36 minutes in order to solve the situation number one, while for the rest of the situations, the average time was 17 minutes. This means that once they know how to configure the platform in order to identify the first situation, it is easier for them to configure it for the rest of the situations. This way, the learning curve is steep, that is, the participants learn in a very short period of time how to use the platform successfully.

VII. CONCLUSIONS

This work presents an innovative platform to ease the development of context-aware systems, which is focused in the involvement of domain experts. The platform also provides the user with novel functionalities that are not available in previous works, making easier the development of context-aware systems. The results of the user evaluation show that non-programmers can also use the platform in collaboration with programmers, in order to manage context data and identify the involvement of domain experts. The platform also provides the user with novel functionalities that are not available in previous works, making easier the development of context-aware systems. The participants learn in a very short period of time how to use the platform successfully.

REFERENCES