# An Adaptive Context Acquisition Framework to Support Mobile Spatial and Context-Aware Applications

André Sales Fonteles[*], Benedito J.A. Neto[**], Marcio Maia, Windson Viana, and Rossana M.C. Andrade[***]

Group of Computer Networks, Software Engineering and Systems (GREat), Federal University of Ceará (UFC), Campus do Pici, Bloco 910, Zip Code 60455-760, Fortaleza, CE, Brazil
{andrefonteles,beneditoneto,marcio,windson,rossana}@great.ufc.br
http://www.great.ufc.br

**Abstract.** The increasing number of mobile devices allows users to access applications anytime and anywhere. In such applications, location is a key information to improve the interaction between user and services. Existing applications combine location with other context information, such as weather, user's activity, temperature, among others. However, developing context-aware applications is still a non-trivial task due to the complexity to implement context management. Additionally, existing context management infrastructures are too brittle to handle changes in the underlying execution infrastructure. In this scenario, this work proposes a context acquisition framework, which tries to reduce the development complexity of mobile spatial and context-aware applications. The framework uses tuples space and OSGi to promote uncoupling and to adapt itself according to application requirements. A proof of concept was developed in order to show how spatial and context filters can be easily implemented during the development of a tracking application.

**Keywords:** GIS, Context-Aware, Adaptation, Mobility, Android.

## 1 Introduction

The increasing number of mobile devices, such as smart phones and tablets, allows users to access the Internet and a wide range of applications anytime and anywhere. This scenario converges to the concept of Ubiquitous Computing predicted by Mark Weiser [20], where the most profound technologies are those that disappear, becoming part of everyday life.

Once one application can be accessed anytime and anywhere, the user's location can be relevant to improve the interaction among user, application and

[*] Master Scholarship (MDCC/DC/UFC) sponsored by CNPQ.
[**] Master Scholarship (MDCC/DC/UFC) sponsored by CAPES.
[***] Researcher scholarship - DT Level 2, sponsored by CNPq.

services provided. For example, if a user is hungry and decides to eat something, an application can provide a list of nearby restaurants without any need of explicit asking the user for his/her location. Many mobile applications have been already developed using spatial data in order to provide personalized services and content, such as Google Places[1], Localscope[2] and Wikicrimes Mobile[3], from the WikiCrimes [7] project.

Moreover, some studies in mobile applications have combined user's location with a larger data set concerning the user's situation, which is considered as user's context. Taking into account that Dey and Abowd [1] defines contextual information as any information that can be used to characterize the situation of entities considered relevant to the interaction between a user and an application, including the user and the application themselves.

The development of mobile applications that integrate spatial information with context data is an important research subject in many domains: from multimedia [3][17] to mobile learning [9]. These applications use contextual information (e.g., weather forecast, temporal data, user's activity, presence of nearby friends or devices) to provide relevant services, to adapt user interface or to produce meta data about multimedia documents.

Despite the benefits of using contextual information to improve mobile applications, to implement context acquisition, management and exploitation methods can be a difficult task. The more contextual information is considered by the application, the greater is the challenge.

First of all, context data is often provided by physical sensors (e.g., GPS, thermometers and accelerometers) as low level information (e.g., latitude, longitude or temperature in degrees). Applications, otherwise, expect high level context data to adapt their behaviour according (e.g., user's current city and if a day is warm or cold). Context-aware applications developers should implement code to provide this high level context information.

Second, to deal with physical sensors may require developer's expertise in some specific hardware. To develop a logical sensor with a context inference mechanism need even more specific knowledge. For example, to develop a logical sensor that given the data from an accelerometer, it infers if the user is walking, running or still is not a trivial task. In addition, mechanisms to detect user's situation and react according are frequently integrated to the code responsible for the application business logic, which makes code reuse a problematical activity.

In this scenario, we propose a context acquisition framework implemented in the Android platform, which tries to reduce the complexity of the development of mobile spatial and context-aware applications. The framework provides context information as tuples in a shared memory space (i.e., tuple space). Mobile applications can connect to the tuple space to query contextual information or to subscribe for being notified when the user's situation changes. Spatial filters can be specified and combined with other context filters to compose this

[1] `http://www.google.com/places/`
[2] `http://www.cynapse.com/localscope`
[3] `http://www.wikicrimes.org`

notification rule. For the application developers, no knowledge concerning the physical layer is required. As a consequence, the framework reduces coupling among applications and the context acquisition layer. This clear separation between context acquisition and context consumption allows the framework to adapt the way context is acquired without the perception of the applications above. This adaptation mechanism tries to reduce resource consumption and provides transparency when sensors are switched.

The remainder of this paper is divided as follows: Section 2 presents the theoretical foundations of our work, which includes the description of the SySSU middleware. Section 3 describes the proposed framework, how it extends the SySSU middleware for including spatial and context-aware filters. The fourth section presents a proof of concept we have developed to illustrate the notification and adaptation mechanisms. Finally, Section 5 contains conclusions and future works.

## 2 Theoretical Foundations

The development of mobile and context-aware applications is not a novel domain. Many technologies have been proposed by researchers and companies aiming to assist developers and software engineers in the task of designing, implementing and testing these applications. In this section, we briefly present achievements and limitations of these approaches. Some of them have been reused, extended or inspired our framework. In particular, we describe SysSU, a middleware proposed by our research group, which is adapted and extended in this work.

### 2.1 Context-Aware Supporting Infrastructures

In an attempt to deal with the complexity of context-aware systems, many frameworks and middleware platforms have been proposed by researchers, as can be seen in [2], [8] and [5]. Baldauf et al. [2] and Marinho et al. [13] identify recurrent architecture layers and their roles in existing frameworks, as follows:

- The Sensors Layer consists of a collection of sensors, varying from physical and logical to virtual. Virtual sensors gather context information from software application or services (e.g., accessing a weather Web Service).
- The Raw Data Retrieval Layer is responsible for retrieving context data. Usually, this layer is composed of wrapper components, which encapsulate sensors, making low-level details, such as hardware access, transparent to applications. Another frequently feature is interface standardization, focused on maximize component reuse. It is possible, for instance, to replace a component that uses GPS by one that uses cellular antenna triangulation to provide location without major modifications in the system.
- The Preprocessing Layer is responsible for translating raw context information into semantic-enriched information. This layer is uncommon in mobile context-aware systems since few inference mechanisms are available on mobile platforms.

- The Storage/Management Layer organizes context information and offers them via a public interface to applications.
- The Application Layer is where the applications using the framework are. These applications are responsible for implementing methods to react and adapt to changes in context.

One important characteristic of existing frameworks and middleware platforms is that they provide means for applications to adapt to context changes. These frameworks, however, do not adapt themselves to the context [5][14]. Furthermore, many of them are monolithic systems, in a sense that their components cannot be deployed separately [14]. Their lack of adaptation capabilities plus their inflexibility to add new components make it difficult to use them in mobile devices with limited resources, inserted in dynamic environments with continual changing contexts. Additionally, analyzing the previous cited surveys on context awareness, there are no references to any work supporting geographic data management and operations.

There are existing frameworks and middleware platforms somehow similar to our approach. In [16] is proposed a context framework for mobile devices running Android called ContextDroid. Its main characteristics are: efficiency, extensibility and portability. Although ContextDroid is unable to adapt its context acquisition mechanism, it proposes a context acquisition infrastructure based on the component oriented programming (COP) model of the Android platform. Our proposed framework uses a similar approach, but the life cycle management of the components is achieved using OSGi[4].

Kramer et al. [10] proposes an infrastructure for context acquisition that monitors context changes independently of application. This infrastructure does not adapt itself as well, but it uses the strategy that a single instance of the infrastructure is shared by many applications in a device, optimizing resource utilization. The same strategy of a single instance is adopted by our proposed framework.

In [14] it is presented a framework capable of performing deployment and runtime adaptation of its components. This framework aims to achieve a better resource utilization in mobile devices. Our adaptation mechanism is inspired by this work. At last, authors in [11] present a context-monitoring framework called MobiCon. Our proposed framework is similar to MobiCon from many perspectives. First of all, both architectures allow different applications to share a single framework instance. Furthermore, MobiCon's approach for context acquisition adapts itself, minimizing the number of working components based on the interest of the applications (expressed by queries). Despite these similarities, MobiCon is event oriented and does not provide contextual information in a synchronous way.

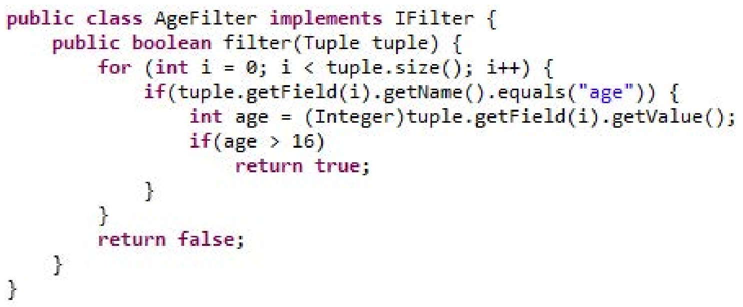
Among the studied context-aware middlewares and frameworks, location data is treated as high-level abstractions (e.g., location=home, theater). They rarely provide mechanisms to deal with spatial relations and operations, such as distance among points and regions, and topology relations (e.g., inside, outside).

[4] `http://www.osgi.org`

## 2.2 SysSU

SysSU (System Support for Ubiquity) [12] is an infrastructure that aims to provide mechanisms to implement the main requirements of ubiquitous systems, such as coordination and service discovery/description. It implements a coordination model formed by the composition of tuple space [4] and event based [6] approaches. The SysSU permits the execution of operations like writing and reading tuples, synchronously or asynchronously.

A tuple is composed of a set of key/value fields. For example, {(user,"John"), (age,10),(gender,"M")}. There are two ways for an application to access a tuple published at SysSU. The first one is to query it. Doing so, the information should be available before it is required by the application. The second one is event based and is used when the information is not yet available or an application wants to be notified every time a new information is published. Both methods make use of templates and filters to select the required tuples. A template is a collection of fields representing a set or a subset of fields composing the required tuple. For example, to find tuples containing a field with a key "user" it would be used the following template: {(user,?)}. A template can also be used to match values. For example, the template {(user,?),(age,10)} returns every user with 10 years old. The templates can be also used to select tuples with same fields, or field with a specific value. To improve the expressiveness of a tuple selection, a filter should be used. A selection filter is created using Java and is only limited by the program language expressiveness. Fig. 1 shows an example of filter that select tuples where a field age is greater than 16. To use a filter it is necessary to create



```
public class AgeFilter implements IFilter {
    public boolean filter(Tuple tuple) {
        for (int i = 0; i < tuple.size(); i++) {
            if(tuple.getField(i).getName().equals("age")) {
                int age = (Integer)tuple.getField(i).getValue();
                if(age > 16)
                    return true;
            }
        }
        return false;
    }
}
```

**Fig. 1.** Example of a filter implementation

a class that implements a Java interface called IFilter. The IFilter is composed by a single method called *filter*. Everytime a query is performed, a template is formerly used to select a subset of all the tuples in SysSU, then every lasting tuple will pass one by one through the *filter* method of the IFilter instance. This method is responsible to check if these tuples meet the given requirements of the filter.

SysSU has showed its efficiency in improve the uncoupling among applications and services that share the tuple space to coordinate their operations. It play a central role in our framework providing uncoupling among Android applications and the context acquisition layer. Originally, SysSU was proposed as a client-server architecture where the server hosted the tuple space to share information among thin clients. In order to reuse SysSU in our framework, we had to adapt its architecture to fully embed it in a single mobile device where applications can simultaneously access an internal tuple space. Its filters mechanisms are extended to support spatial and context-aware notifications.

## 3 Proposed Framework

We propose a framework for Android that acquires context in an adaptive manner designed to be used embedded in mobile devices, which has limited resources and are inserted in dynamic environment. This framework allows deployment and dynamic (i.e., runtime) adaptation of its context acquisition components (CAC) to achieve a better resource usage [14]. Besides, dynamic adaptation enables a system to fulfill its requirements in response to changes in context at runtime [15]. Another characteristic of this framework for better resource utilization is that just one single instance runs per device, while many applications can use it.

Fig. 2 shows the architecture of the proposed framework. The main entities of the architecture are Context Acquisition Component (CAC), CAC Manager, Adaptation Reasoner, SysSU, SysSU Filters and Application.

A CAC is a component type that wraps physical, logical and virtual sensors and provides context information to the framework. These components can be started, stopped, installed and removed through the CAC Manager even during runtime. To provide such flexibility the CAC Manager makes use of OSGi framework. The manager is not responsible for reasoning about what configuration of CACs the framework will use. This is a role of the Adaptation Reasoner. The Adaptation Reasoner decides what CACs should be used at runtime according to the needs of the Applications.

Our framework provides a low coupled interaction with the application using an infrastructure called SysSU. SysSU is proposed by Lima et al.[12] based on tuple space that provides means of coordination and interaction that are decoupled for the development of ubiquitous systems. All the communication between the application and the framework is done by using SysSU. Applications perform queries for context information through SySU. The Adaptation Reasoner then determines a valid configuration of CACs that fulfills the requirement of the queries and use the CAC Manager to achieve this configuration. Finally, the running CACs publish their context information at SysSU where the applications are able to access it.

The SysSU Filters is an optional API intended to provide a collection of filters (see Session 2.2) used to query or to subscribe to events based on contextual information. This API also should provides spatial filters to manage geographic

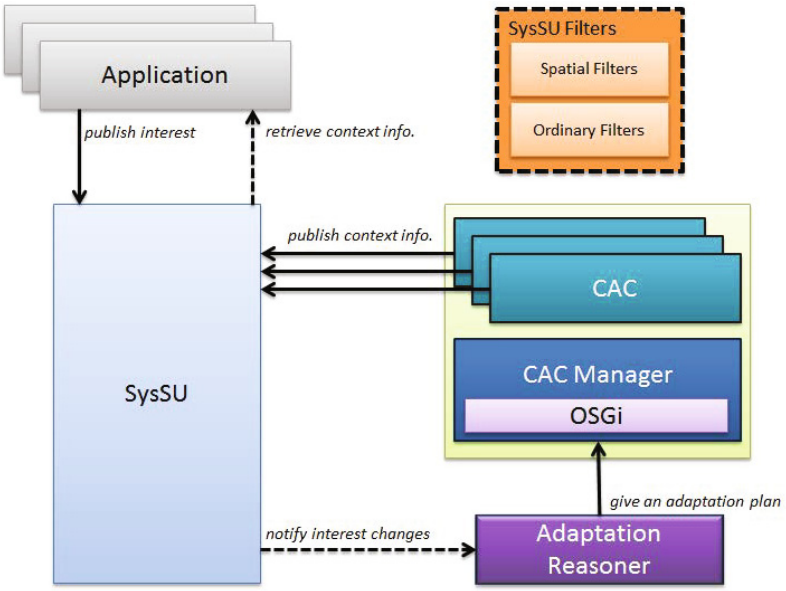


**Fig. 2.** Architecture of the proposed framework

contextual information. At the moment, we only provide two of there filter as a proof of concept.

## 3.1 Context Representation

In order to allow communication between components and applications, it is necessary to use a shared vocabulary to represent types of context information. Each type of context information (e.g., temperature, location and weather) requires a unique key or identifier that should be used by CACs, to determine what kind of information it provides to framework and to publish information, and by applications to query this same kind of information.

We defined, based on the Management Information Base (MIB)[5] model, a hierarchical scheme to generate unique keys, called context keys (CK), to types of context information. Using this schema, a contextual information is referenced using a sequence of names separated by points. For instance, a key context.device.location references to a device's location information. Thus, any CAC that publishes a contextual information of a device's location at SysSU should use this CK. In order for an application to find that information at SysSU, the same CK should be used.

Fig. 3 shows a subset of the hierarchy already defined to compose CKs. As soon as new CACs providing new kinds of context are developed, this hierarchy can be extended from any node at the tree.

[5] `http://www.ieee802.org/1/pages/MIBS.html`

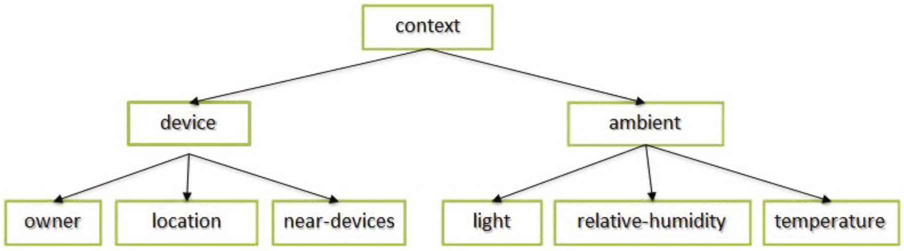


**Fig. 3.** Subset of the hierarchy already defined to compose CKs

Besides the need of CKs to identify kinds of context, it is also necessary to specify how can such information be represented on a tuple space. Based on a context metamodel proposed by Vieira et al. [19], we define that a tuple with context information (called context tuple) must have at least four fields:

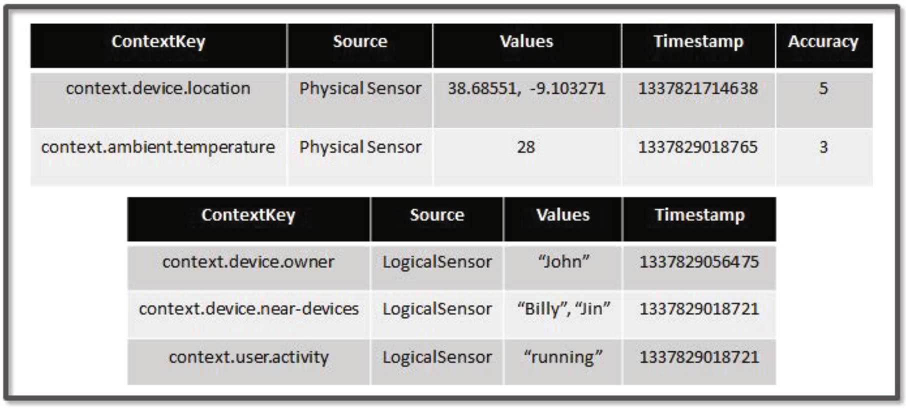
– ContextKey: In this field, the CK representing the kind of context information at the context tuple should be inserted. Applications should use this field to match tuples with their required kinds of context.
– Source: It informs the source of the information. Contextual information can be originally provided by physical, logical and virtual sensors.
– Values: This field contains an array with the actual values of the context. For example, in a context tuple representing an ambient temperature in Celsius, this array would have one index with a value such as 28.
– Timestamp: This field contains the time, in milliseconds, which a contextual information was read.

Others fields can be added according to the needs of contextual representation. For example, in some cases, it is added an *Accuracy* field where the accuracy (precision) of the reading is set (e.g., user's location with a accuracy of 5 meters). Fig. 4 shows examples of context tuples.

Using the CK definition and the context tuple aforementioned, applications may access the tuple space and execute queries or subscribe themselves to be notified when an specific context information is generated, as described in section 2.2. For instance, let an existing CAC be responsible for the address of the user, in which its CK is context.device.location.address. Applications may access the address of the user using the {(contextkey,"context.device.location.address")} template.

### 3.2 Spatial Filters

In most part of the mobile and context-aware applications, user's location information plays an important role. However, the manipulation of geographic data in mobile applications is a difficult task, especially, for inexperienced developers in the domain. In some cases, developers will be unable to use a remote geographic



| ContextKey | Source | Values | Timestamp | Accuracy |
|---|---|---|---|---|
| context.device.location | Physical Sensor | 38.68551, -9.103271 | 1337821714638 | 5 |
| context.ambient.temperature | Physical Sensor | 28 | 1337829018765 | 3 |

| ContextKey | Source | Values | Timestamp |
|---|---|---|---|
| context.device.owner | LogicalSensor | “John” | 1337829056475 |
| context.device.near-devices | LogicalSensor | “Billy”, “Jin” | 1337829018721 |
| context.user.activity | LogicalSensor | “running” | 1337829018721 |

**Fig. 4.** Examples of context tuples

database to answer simple spatial queries such as “what is the minimum distance in meters between the user and a given spatial point?” or “is the user inside a given region?”. For instance, when there is no available internet connection or when the application runs locally in the mobile device.

In addition, analyzing surveys on frameworks and middleware platforms for context management as [2], [8] and [5], one can conclude that these infrastructures provide few services or mechanisms for dealing with geographic data.

To reduce this inconvenience, the proposed framework enables the development and reuse of code for managing this type of data, by extending the notification filters of SysSU with spatial operations. As part of this work, we developed two filters for creating queries and events based on geographical information: DistanceFilter and PlaceFilter.

The DistanceFilter is used when an application wants to know if the user is at a minimum distance of a specific coordinate. It was developed making use of the class Location of the Android Framework. The Location class has a method called *distanceTo* that, according to the Android documentation[6], returns the approximate distance in meters between two given locations.

The PlaceFilter is used to determine if the user is within one or more geographic areas determined by a set of coordinates, as shown in Fig. 5. For example, a PlaceFilter can be used by a mobile learning application, which requires a notification when a user is inside or near to a historical monument in order to play video or audio containing related information. This filter also makes use of a third-party API, called JST[7], to test intersections between a polygon and a point. The JTS is an API of 2D spatial predicates and function. We did not use map projections to transform latitude and longitude to points in a cartesian plan, so this filter is an approximation.

[6] `http://developer.android.com/reference/android/location/Location.html`
[7] `http://www.vividsolutions.com/jts`

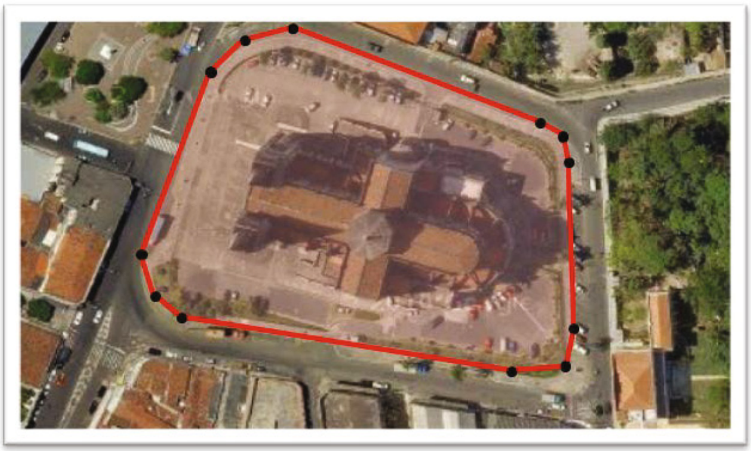


**Fig. 5.** Example of an area used by PlacesFilter

Like the two spatial filters aforementioned, our framework enables developers to create new filters from the domain specific needs of their applications and share it as a Java class for reuse purposes.

### 3.3 Adopted Context Definition

As we discussed before, Dey and Abowd defined context as any information that can be used to characterize the situation of entities considered relevant to the interaction between a user and an application. In this work, we adopted another definition of context made by Viana [18]. Viana extends the former definition but focusing at context acquisition, and affirms that context can be characterized as an intersection between two dynamic and evolutionary sets, as shown in Fig. 6.

A set is named Interest Zone (IZ) and is contains all the information related to environment and user that the system would like to know. The other set is named as Observation Zone (OZ), where there are all contextual information that the system can obtain. The intersection between what is interesting to the system and what the system can observe in a certain time is considered as context. For example, suppose a smartphone application that, according to the ambient sound, adjust the bell volume. In this case, the ambient sound is an element from IZ, because the system wants this information. Through the device's microphone is possible to capture the ambient sound, then, it is also in the OZ. Finally, for being in both sets, the ambient sound is considered as context to the application.

The elements that are in IZ and OZ may change over time. Following the previous example, as the time pass, the user can configure his/her smartphone to silent mode. If this occurs, his device will not emit any sound alert. Thus, it does not matter to the system to have knowledge about the ambient sound to adjust the bell. This condition removes the ambient sound from IZ, although it

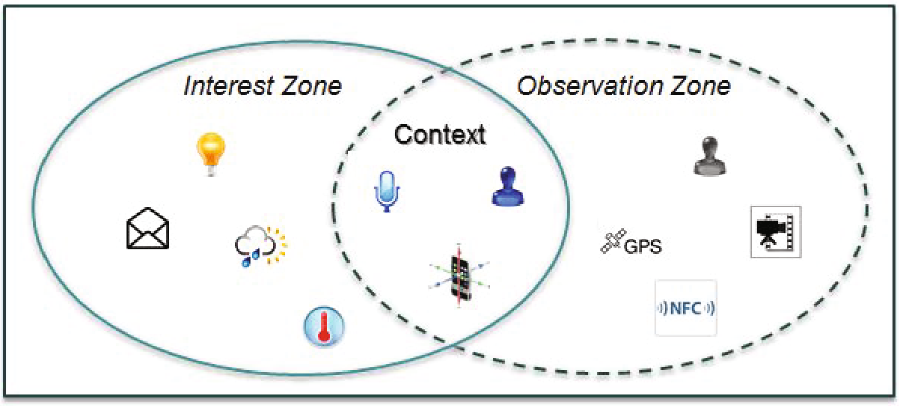


**Fig. 6.** Context as an intersection between interest zone and observation zone. Adapted from [18]

is still in OZ. Once the ambient sound is not present in both sets at the same time anymore, it is not considered as context. Likewise, even if the device was not in silent mode, its microphone might be damaged, and this would be an impediment to the capture of the ambient sound. Therefore, it would not be in OZ and would not be part of the context, despite the fact is still present in IZ. The behaviour previously described is known as the context evolutionary feature. Formally speaking, we can say that the observed elements that compose the context in a certain time *t* are not necessarily the same in another time *t'*.

Each application that uses the proposed framework must send an updated CK list that represents its interests. This list, along with its possible interests (dependencies) of running CACs, represents the IZ for the framework. On the other hand, the set of all context CKs provided by running CACs represents the OZ.

## 3.4 Adaptation Mechanism

The adaptation of the proposed context acquisition framework takes place in two different moments: deployment time and runtime. Deployment time adaptation permits the developer to install strictly those CACs with CKs required by the applications.

Additionally, runtime adaptation is performed based on the context of the running applications. Context, as previously described, is the intersection between Interest Zone and Observation Zone. Thus, the CAC lifecycle management allows the framework to stop and remove CACs in order to optimize resource consumption based on the current context of all applications. The main goal of the runtime adaptation is to reach a state where IZ contains the OZ, as shown in

Fig. 7. Therefore, all unnecessary CACs in a given time are stopped. When the IZ changes, the adaptation mechanism is responsible for detecting this change and searches for compatible CACs to handle the new IZ.

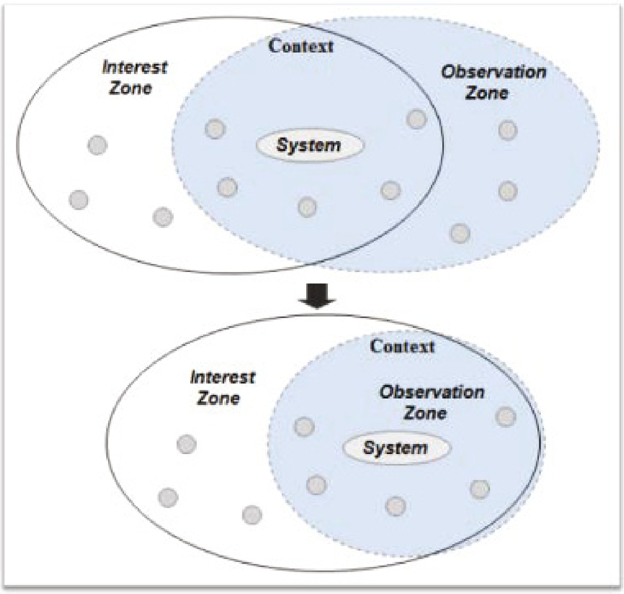


**Fig. 7.** Context acquisition adaptation behaviour

## 4 Validation

As a proof of concept, in this section we describe how an application can be implemented using our framework to incorporate a context-aware behaviour and easily react for location and context changes. We define a scenario where an application tracks a user trajectory. Every coordinates pair of the trajectory is annotated with more contextual information (e.g., ambient light, temperature and relative humidity). These kind of contextual "log" is common on many multimedia applications, such as Photomap [17] and Captain [3]. We also define a region on map (e.g., an area around a square), in a certain point of the trajectory, where an application would desire to be notified to perform some action when a user get inside or nearby (depending on the filter). This feature is very common in mobile location based applications. For instance, a mobile learning application that proposes multimedia content about a historical monument when the user is inside or in the neighbourhood, can use the same filter mechanism.

The track application was implemented in Android using our framework. Fig. 8 shows a visual representation of some tracked points acquired during an real application usage test, in which the user walked through some street blocks. Fig. 8 also presents a region of interest, where our application successfully get warned.

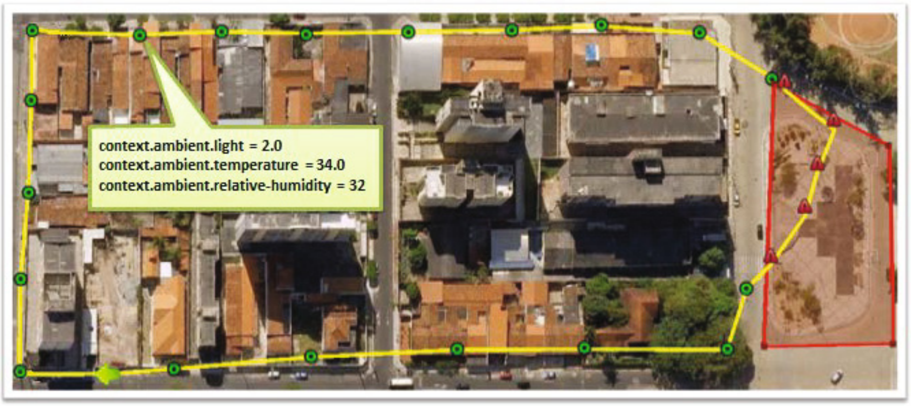

context.ambient.light = 2.0
context.ambient.temperature = 34.0
context.ambient.relative-humidity = 32

**Fig. 8.** Visual representation of some tracked points acquired

To use our framework in the track application, three steps were necessary to follow:

1. publish IZ in the tuple space;
2. filter subscription; and
3. tuple space query

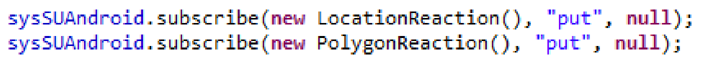
The first step in the application development was to publish its IZ at SysSU tuple space. This step was necessary to allow the framework to adapt its OZ enabling the necessary CACs. Fig. 9 shows how we did it. SysSUAndroid is the hotspot of our framework,

```
sysSUAndroid = new SySSUAndroid();
sysSUAndroid.publishInterests("appName", "context.device.location");
sysSUAndroid.publishInterests("appName", "context.ambient.light");
sysSUAndroid.publishInterests("appName", "context.ambient.temperature");
sysSUAndroid.publishInterests("appName", "context.ambient.relative-humidity");
```

**Fig. 9.** Publishing application's interest

Once the IZ was published at the tuple space, the application has subscribed to listen new context tuples when user's location was published by CACs. Subscriptions need three phases. First, we created a tuple template as following {(ContextKey, "context.device.location")}. Second, we created a class called

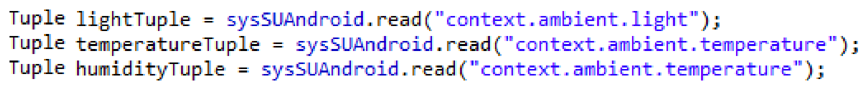
LocationReaction that implements an interface called IReacion and uses the template. This class will be notified every time a new tuple is published if this tuple matches with our template. Finally, in order to use the reaction, the track application subscribed through a method called subscribe, as shown in Fig. 10.



```
sysSUAndroid.subscribe(new LocationReaction(), "put", null);
sysSUAndroid.subscribe(new PolygonReaction(), "put", null);
```

**Fig. 10.** Subscribing to receive events

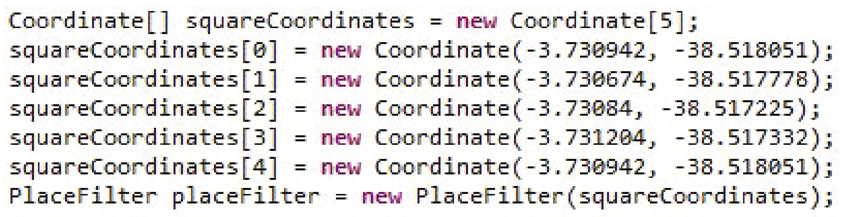
In our IReaction implementation, everytime the application received an event of a new context tuple with user's location, the application read synchronously the state of the ambient light, temperature and relative humidity. All these metadata was associated to the location information in a log file. Fig. 11 shows how one can query a context information from tuple space synchronously with our framework.



```
Tuple lightTuple = sysSUAndroid.read("context.ambient.light");
Tuple temperatureTuple = sysSUAndroid.read("context.ambient.temperature");
Tuple humidityTuple = sysSUAndroid.read("context.ambient.temperature");
```

**Fig. 11.** Querying a context information from tuple space synchronously

Finally, we subscribed again to the framework, this time to receive events when the user entered in a defined region (an area around a square). We created another class implementing IReaction that was responsible to log when this situation occurs. To perform this task, the same template of the previous IReaction were reused. In this case, it was necessary to use the PlaceFilter described in Section 3. To use a filter together with a IReaction, we should request a method called getRestriction, which return this filter. Fig. 12 shows how we created the PlaceFilter instance.



```
Coordinate[] squareCoordinates = new Coordinate[5];
squareCoordinates[0] = new Coordinate(-3.730942, -38.518051);
squareCoordinates[1] = new Coordinate(-3.730674, -38.517778);
squareCoordinates[2] = new Coordinate(-3.73084, -38.517225);
squareCoordinates[3] = new Coordinate(-3.731204, -38.517332);
squareCoordinates[4] = new Coordinate(-3.730942, -38.518051);
PlaceFilter placeFilter = new PlaceFilter(squareCoordinates);
```

**Fig. 12.** Creating a PlaceFilter instance

This proof of concept shows how the context acquisition layer is isolated from the application code. As seen, the developer of a mobile application should only be concerned about creating filters and query the contextual properties to which the application is interested. No code for accessing the physical and logical sensors is required to be implemented by third application developers. This allows the framework to change how the context acquisition is performed without the application worry about these changes (e.g., sensor errors, sensors reconfiguration to reduce resource consumption). In addition, the framework facilitates the development of this kind of application by hiding the complexity of context acquisition and matching.

## 5 Conclusion and Future Work

This paper presented an adaptive context acquisition framework to support mobile spatial and context-aware applications. Its main benefits are the flexibility to adapt the context acquisition infrastructure both at deployment time and at runtime, minimizing resource consumption and improving overall uncoupling between the application and the context acquisition infrastructure.

The second benefit is the ease with spatial filters that are implemented and reused, facilitating the use of mechanisms to deal with spatial relations and operations, such as distance among points and regions, and topology relations. Then, spatial and other context information is aggregated, providing mechanisms for applications to perform more elaborate queries.

As a future work, a online CAC repository is going to be developed, permitting CACs to be accessed at runtime. This repository would be used in case of an application interest can not be satisfied. Also, we would like to increase the number of filters to be used with SysSU to facilitate the management of contextual data. Finally, information regarding quality of service (QoS) is going to be annotated on each CAC, permitting applications to consider QoS when deciding which CAC is more appropriate in a given situation.

**Acknowledgements.** This work is a partial result of the UbiStructure project supported by CNPq (MCT/CNPq 14/2011 - Universal) under grant number 481417/2011-7, and FUNCAP project number PJP-0072-00091.01.00/12. It was also partially funded by the Program of Scientific Cooperation called STIC-AmSud. The sponsored project is entitled Learning While Moving (LWM).

## References

1. Abowd, G.D., Dey, A.K.: Towards a Better Understanding of Context and Context-Awareness. In: Gellersen, H.-W. (ed.) HUC 1999. LNCS, vol. 1707, pp. 304–307. Springer, Heidelberg (1999), `http://dx.doi.org/10.1007/3-540-48157-5_29`
2. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing 2(4), 263–277 (2007)

3. Braga, R.B., de Moraes Medeiros da Costa, S., de Carvalho, W.V., de Castro Andrade, R.M., Martin, H.: A Context-Aware Web Content Generator Based on Personal Tracking. In: Di Martino, S., Peron, A., Tezuka, T. (eds.) W2GIS 2012. LNCS, vol. 7236, pp. 134–150. Springer, Heidelberg (2012), `http://www.springerlink.com/index/0U2052W26025Q545.pdf`
4. Carriero, N., Gelernter, D.: Linda in context. Commun. ACM 32(4), 444–458 (1989), `http://doi.acm.org/10.1145/63334.63337`
5. Da, K., Dalmau, M., Roose, P., et al.: A survey of adaptation systems. International Journal on Internet and Distributed Computing Systems 2(1), 1–18 (2011)
6. Eugster, P., Felber, P., Guerraou, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Computing Surveys 35(2), 114–131 (2003), `http://portal.acm.org/citation.cfm?doid=857076.857078`, `http://dl.acm.org/citation.cfm?id=857078`
7. Furtado, V., Ayres, L., de Oliveira, M., Vasconcelos, E., Caminha, C., D'Orleans, J., Belchior, M.: Collective intelligence in law enforcement - the wikicrimes system. Inf. Sci. 180(1), 4–17 (2010), `http://dx.doi.org/10.1016/j.ins.2009.08.004`
8. Hong, J., Suh, E., Kim, S.: Context-aware systems: A literature review and classification. Expert Systems with Applications 36(4), 8509–8522 (2009)
9. Hwang, G., Tsai, C., Yang, S., et al.: Criteria, strategies and research issues of context-aware ubiquitous learning. Educational Technology & Society 11(2), 81–91 (2008)
10. Kramer, D., Kocurova, A., Oussena, S., Clark, T., Komisarczuk, P.: An extensible, self contained, layered approach to context acquisition. In: Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing, p. 6. ACM (2011)
11. Lee, Y., Iyengar, S., Min, C., Ju, Y., Kang, S., Park, T., Lee, J., Rhee, Y., Song, J.: Mobicon: a mobile context-monitoring platform. Communications of the ACM 55(3), 54–65 (2012)
12. Lima, F., Rocha, L., Maia, P., Andrade, R.: A decoupled and interoperable architecture for coordination in ubiquitous systems. In: 2011 Fifth Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), pp. 31–40. IEEE (2011)
13. Marinho, F.G., Andrade, R.M., Werner, C., Viana, W., Maia, M.E., Rocha, L.S., Teixeira, E., Filho, J.B.F., Dantas, V.L., Lima, F., Aguiar, S.: Mobiline: A nested software product line for the domain of mobile and context-aware applications. Science of Computer Programming (2012), `http://www.sciencedirect.com/science/article/pii/S0167642312000871`
14. Preuveneers, D., Berbers, Y.: Towards context-aware and resource-driven self-adaptation for mobile handheld applications. In: Proceedings of the 2007 ACM Symposium on Applied Computing, vol. 11, pp. 1165–1170 (2007)
15. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM Transactions on Autonomous and Adaptive Systems (TAAS) 4(2), 14 (2009)
16. Van Wissen, B., Palmer, N., Kemp, R., Kielmann, T., Bal, H.: Contextdroid: an expression-based context framework for android. In: PhoneSense 2010: International Workshop on Sensing for App Phones, pp. 6–10 (2010)
17. Viana, W., Filho, J.B., Gensel, J., Villanova Oliver, M., Martin, H.: PhotoMap – Automatic Spatiotemporal Annotation for Mobile Photos. In: Ware, J.M., Taylor, G.E. (eds.) W2GIS 2007. LNCS, vol. 4857, pp. 187–201. Springer, Heidelberg (2007)

18. Viana, W.C.: Mobilité et sensibilité au contexte pour la gestion de documments multimédias personnels: CoMMediA. Ph.D. thesis, Université Joseph-Fourier - Grenoble (2010), http://hal.archives-ouvertes.fr/tel-00499550/
19. Vieira, V., Tedesco, P., Salgado, A.C.: Designing context-sensitive systems: An integrated approach. Expert Systems with Applications 38(2), 1119–1138 (2011), http://www.sciencedirect.com/science/article/pii/S0957417410004173; Intelligent Collaboration and Design
20. Weiser, M.: The computer for the 21st century. Scientific American 265(3), 94–104 (1991)