Journal of Universal Computer Science, vol. 16, no. 15 (2010), 2081-2098
submitted: 31/1/10, accepted: 28/7/10, appeared: 1/8/10 © J.UCS

Multi-Purpose Infrastructure for Delivering and Supporting Mobile Context-Aware Applications

Juan Miguel López
(University of Lleida, Lleida, Spain
juanmi@diei.udl.cat)

Montserrat Sendín
(University of Lleida, Lleida, Spain
msendin@diei.udl.cat)

Abstract: The use of contextual information in mobile devices is receiving increasing attention in mobile and ubiquitous computing research. An important requirement for mobile development today is that devices should be able to interact with the context. In this paper we present a series of contributions regarding previous work on context-awareness. In the first place, we describe a client-server architecture that provides a mechanism for preparing target non context-aware applications in order to be delivered as context-aware applications in a semi-automatic way. Secondly, the framework used in the server to instantiate specific components for context-awareness, the *Implicit Plasticity Framework*, provides independence from the underlying mobile technology used in client device, as it is shown in the case studies presented. Finally, proposed infrastructure deals with the interaction among different context constraints provided by diverse sensors. All of these contributions are extensions to the infrastructure based on the *Dichotomic View of plasticity*, which now offers multi-purpose support.

Keywords: context-awareness, mobile development, adaptation, multi-sensor context
Categories: C.2.4, D.1.m

1 Introduction

Mobile devices are changing users' technological habits. Advances in mobile technology are providing mobile users with the capacity to interact and perceive with their surrounding world. A user with a mobile device can find that its standard functionality can be extended with those applications and services provided by the environment. These kinds of systems have to address more sophisticated techniques to react to changes in the context in order to adequately adapt the system's behaviour and the User Interface (UI henceforth) appearance to the context of use. It is necessary to provide multi-purpose tools that facilitate the development and delivery of context-aware mobile systems, which be able to gather information from the surrounding in order to provide better awareness of situational context.

This paper presents a software infrastructure to automatically adapt mobile systems according to their context of use. This infrastructure provides a multi-sensor support and is independent of mobile technology used on the device. Proposed solution follows the *Dichotomic View of plasticity* approach [Sendín, 2007], where infrastructure is based on a client-server architecture, providing a communication

mechanism between mobile applications and the server along the execution. According to this approach, there are two types of adaptations: implicit and explicit. When the complexity involved in the adaptation process is too high to be handled on the device, the client requests the server for an *explicit plastic* adaptation, which implies a higher level of reasoning or reconfiguration of the UI. On the other hand, if the adaptation complexity is low enough to be solved locally, the client tackles it, providing an *implicit plastic* adaptation. Last kind of adaptations are carried out by means of a specific *Implicit Plasticity Engine*[1] (IPE henceforth) previously embedded on the client application. In the case of components for context-awareness, simple sensor adaptations can be done locally by the client. Server is required to solve situations where multi-sensor context constraints enter into conflict.

The server allows originally non context-aware mobile applications to be uploaded. Then, these applications are prepared for being aware to certain contextual needs and can later be delivered to a client mobile device by request. Once running on the client device, applications present a context-aware behaviour, providing adaptations in an autonomous and dynamic way. To do that, the server makes use of a multi-technology framework, which is easily be instantiated to a specific IPE –the required component for context-awareness-, ready to be embedded on the original application. This framework can be configured so that certain characteristic can be specified manually, according to particular parameters and particularities for each application, thus establishing a specific adaptation engine. Once the recently context-aware application is installed on the device and starts to operate, in the presence of multi-sensor conflict situations where different types of adaptations can be enforced, involved contextual information is sent to the server, looking for an *explicit plastic* adaptation. The server applies the necessary inferences in order to determine what adaptation is the most convenient for each particular circumstance.

In this paper we present a series of contributions regarding the previous work on context-awareness. In the first place, a mechanism for preparing target applications in order to be delivered as context-aware applications in a semi-automatic way. Secondly, the fact that the framework used in server to instantiate components for context-awareness is independent from the mobile technology used in the client device. This framework is called *Implicit Plasticity Framework* (IPF henceforth) [Sendín, 2006]. Finally, proposed infrastructure deals with the interaction among different context constraints. We are referring to the integration of diverse sensors for a more enriched awareness. All these contributions have been implemented providing extensions to the previous infrastructure based on the *Dichotomic View of plasticity*.

The paper is structured as follows. Next section discusses some related work. Section 3 presents special considerations for designing components for context-awareness, following the general software structure for IPEs. This section concludes presenting a multi-sensor support based on the *Dichotomic View of Plasticity*. The general architecture in the server and the generic framework to derive components for context-awareness are presented next. Particularly, it is shown how it is applied in the development of specific components for two case studies with different mobile technologies. Conclusions and further work conclude the paper.

[1] Client-side runtime adaptive component with the capacity to detect the context and reacting in order to adapt the UI and the device operative to the contextual variations on the fly.

2 Related Work

Context awareness in mobile environments has been a research area for a long time. One of the initial works is [Schilit, 1995], where an architecture capable of supporting context in mobile environments was proposed. It is based on the experience acquired with the ParcTab prototype from Xerox PARC. On the other hand, [Dey, 2000] defined seven basic requirements a context support infrastructure should fulfil, which have been used by a large number of context aware systems developed since then.

Context awareness lies heavily on a series of concepts that have been developed over time. Since the first context architectures appeared, there is an effort to incorporate sensors and different technology providers to context, considering it as an overall evolution of the whole system [Couto, 2005]. Besides, in order to make the development of context aware systems easier, context middlewares are defined. Their task is low coupling context aware applications from dependences with underlying layers, relieving the programmer from the need to build a communication framework. Moreover, in order to support possible context evolution, some existing middleware explore the resource discovery concept (one of the requirements defined by [Dey, 2000]), such as Virtual Information Towers [Leonhardi, 1999] and Context Toolkit [Dey, 2000], being this last one of the most relevant context infrastructures in the field. There is also work in progress for developing run-time middleware that enables the generation of UIs on portable devices based on sets of abstract models, and adapting them to the context of use [Yaici, 2008].

Ubiquitous computing is characterized by heterogeneity of devices used to access services; services that users expect to receive anytime and anywhere. Hence, mobile services need to efficiently adapt to different contexts of use, so that mobile users can exploit then. [Malandrino, 2009] provides a middleware for context-awareness with an intermediary based architecture for content adaptation.

Another field where context has also been taken into account as a crucial factor has been the personalization of UIs. To cite an example, [Sendín, 2009] presents a system that offers dynamic support to real limitations or difficulties users can encounter during the use of a mobile UI. Moreover, as the context of the mobile user includes user culture and the influence of culture on mobile device use, models have been proposed to represent the influence of mediating factors and determining factors on actual mobile device use [van Biljon, 2008].

Support for context awareness has grown to include different types of mobile environments. The use of mobile applications in the educational environments has led to the necessity to lead with context awareness to improve their efficiency. [Gómez, 2009] provide a framework for the generation of instructional designs considering the context, which can be used in both learning management systems and diverse mobile devices. Moreover, mobile computing devices and a proximity model can be used to organize collaborative activities according to the domain context and physical proximity in educational environments [Zurita, 2007]. Numerous types of mobile environments where context is a key issue to improve their basic operation, and in which contextual information has also been proven useful could be enumerated.

The rest of the paper is destined to present our infrastructure for delivering and supporting mobile context-aware applications. The two case studies presented serve to prove its validity and flexibility to be applied in different mobile technologies.

3 Implicit Plasticity Engine: General Guidelines to design Components for Context-Awareness

As described in the introduction, the IPE is framed into a software infrastructure based on client-server architecture, following the so-called *Dichotomic View of plasticity* [Sendín, 2007]. This section starts presenting the general software structure for IPEs, then explains special considerations to design components for context-awareness, and finally concludes presenting our multi-sensor support.

3.1 Software Structure for the Implicit Plasticity Engine

The framework presented in this paper and the components for context-awareness obtained from it guarantee the properties of transparency in adaptation and reusability; a level of reusability that allows applying it to (1) different families of systems; (2) different needs of context representation; and (3) different adaptation mechanisms, looking for a multi-purpose use. One of the main points to reach flexibility and reusability is that adaptive mechanisms and system core functionality are handled orthogonally, allowing them to evolve individually. Managing orthogonality needs to apply some kind of *separation of concerns* technology.

Based on these premises, the IPE follows a software architecture divided into three layers. The *logical layer* contains the application core functionality. The *context-aware layer* contains the control and modelling of the context constraints, the so-called contextual model. This layer carries out the context detection and maintains contextual information for further use. Finally, the intermediary layer is responsible for adapting the UI or system behaviour according to the context. According to the approach and guidelines defined in [Sendín, 2005], and as it has been discussed in previous works [Sendín, 2006], [Sendín, 2007], [Sendín, 2009], we use Aspect-Oriented Programming (AOP henceforth) as the separation of concerns technology chosen to transparently integrate adaptation mechanisms for real time constraints in the system operation. Thus, we model each context constraint as a program unit called *aspect*, which is in charge of intercepting the operative of the core system to apply the suitable adjustments to the UI, according to the current state of the context. This is the reason why the intermediary layer is called *aspectual layer*. Provided added value is that this layer acts as a transparent link between the other layers. It is this way because dependences are produced exclusively from the *aspectual layer* towards the other two ones. Consequently, the *logical layer* and the *context-aware layer* are completely unaware of the existence of the *aspectual layer*. It implies that, in any case, neither the application code nor the internal structure from the original system are affected, thus fulfilling the transparency property.

3.2 Special Considerations for Context-Awareness

One of the possible applications of the software structure for the IPE outlined above is to provide context-awareness to a certain application that does not present originally these kinds of considerations. In this section, the particular characteristics of a component for context-awareness based on the IPE software structure are presented.

First of all, it is worthy to say that components for context-awareness are particularly characterized by a high level of orthogonality. It implies a low coupling

in relation to the base application, factor that acts in favour of reusability to different systems. The difference is significant in comparison to other kind of components also designed following the structure for the IPE, such as user personalization components [Sendín, 2009]. The reason is that the parameters to be considered in the adaptation do not come from the user behaviour nor are captured from the execution of the application, thus not being necessary to monitor the use of the interface. Instead, parameters for adaptation come from the sensors connected to the application, which keep feeding the application with the values being captured, in this case, from the environment. In this section we describe the design corresponding to the *context-aware* and the *aspectual layers* for these kinds of components.

Context-aware layer. The design for this layer responds to general design principles for software engineering, looking for the maximum flexibility and reusability. Thus, the design consists of different pieces of code, cohesive enough to be reused in different systems. The main class is called *ConstraintManager*, which controls and coordinates all the other classes in this layer. In particular, this class is responsible for (1) acceding and communicating with the sensor, which is represented by the *Sensor* class; and (2) managing the historical of changes in the context constraint -represented by the *ChangesHistorical* class-, in order for them to be conveniently registered. These two functionalities are attained encapsulating the corresponding objects from both classes (*Sensor* and *ChangesHistorical*), as it is depicted in Figure 2. Apart from these classes, there intervenes another class called *ScreenRegulator*. This class encapsulates the code corresponding to the adaptation, generally a call to a specific method from an API proprietary, in a class method visible to the *aspect*. The invocation of this method by the *aspect* triggers the adaptation in a platform-independent way, removing thus any technology dependence in the *aspectual layer*.

Finally, it is convenient to point out that each system and particular contextual need require a different configuration in the communication with the sensor. Sometimes, it is enough controlling certain concrete *pointcuts*, but in other cases it is necessary to establish certain regularity, distinguishing different levels of interactivity with the sensor. In those cases where a high interactivity level with the sensor is required, it is used a thread acting in background in order to accede the sensor in a cyclic way. The class *SimpleThread* in Figure 2 represents and encapsulates this thread. Apart from that, the structure and code for the *aspect* (explained next) also changes in these cases, because it is necessary to monitor the continuous consultations to the sensor. Keeping in mind that our goal is to offer a reusable framework, the so-called IPF, it is necessary to isolate these differences in the *aspectual layer*. Details about this are discussed in section 5.2, where the IPF is described.

A*spectual layer.* In this layer a program unit *aspect* acts, which manages all the actions in favour of adapting system's behaviour according to a certain context constraint. It is the *ConstraintControl aspect* in Figure 2. To be more precise, this *aspect* is in charge of two main responsibilities: (1) coordinating the accesses to sensors in order to capture values from the environment; and (2) putting to work the functionalities and adaptations encapsulated in the *context-aware layer* in order to be applied in the base application (*logical layer*), according to both, the values captured and a set of parameters included in the *aspect* unit. These parameters are: (1) the points in the execution of the application that have been established to check the

sensor; and (2) the threshold to be taken into consideration in order to determine if a certain variation in the sensor value is significant enough to apply an adaptation. These two parameters are specific for each system, and they are expressed by means of *pointcut*[2] constructions. We are referring to the *ActivateSensorConsultation()* and the *ControlChanges() pointcuts* in Figure 2 (the latter is in charge of keeping the system updated by environment changes). Setting values for these parameters are part of the actions that need to be done manually (in particular, by the IPE designer in the IPE derivation step –the second- shown in figure 1), as part of the instantiation of the framework to a specific system and contextual need. We are referring to the completion of the code from the framework in order to obtain the concrete component to be embedded in the target application.

The third *pointcut*, the so-called *CloseHistorical()*, is triggered when it is required to send the historical of changes in sensor values to a server, in order to satisfy the fifth requirement defined by [Dey, 2000], requesting for a *explicit plastic* adaptation. By default, this functionality is done when the application finishes. However, sending the historical to the server can be useful in other types of circumstances during execution, i.e. when different context constraints interfere with each other. We are referring to the multi-sensor problem presented in section 3.3. As it is discussed there, we solve this problem carrying out a process of inference in server, once all the values from sensors have been received. The server infers a solution for the sensor interaction and returns it to the client-side, expressed as a set of proposals of adaptation. As it has already been pointed out, this kind of support to multi-sensor awareness is based on the *Dichotomic View of plasticity* [Sendín, 2007].

Finally, it is important to note that the more concise and simple the code from *aspect* units is, the more flexible and reusable the software structure for IPEs is. This is the reason why the operations related to adaptation are encapsulated in the *context-aware layer*. *Aspects* only have to invoke the corresponding method. As a consequence, the *aspectual layer* is relieved from this code, so fulfilling the function of an intermediate link between the other two layers. The consequent simplification of the *aspectual layer* also contributes to a major understanding of the framework.

This fact is especially remarkable for user personalization components. In order to obtain this kind of components, it is necessary to establish a major number of hooks and dependences among layers, so it is essential to encapsulate the adaptation operation related in the *context-aware layer*. By the way, these components require a deep knowledge from the base application code, which reduces considerably the level of orthogonality. Despite of that, as in components for context-awareness, dependences are produced exclusively from the *aspectual layer* to the two other layers, so the target application is keept equally unaware of the existence of the *aspectual layer*. In this sense, the framework works the same way with independence of the purpose of the component.

[2] Programming construction to establish those points in the program flow (for instance, a method call, an access to a variable, a condition and so on, which are called *joinpoints*) to be interfered at runtime by and *aspect* unit.

3.3 Providing Multi-Sensor Support

It is well-known that providing automatic adaptation to a specific context constraint helps improving user comfort while using a mobile system. However, in lots of situations it is not enough considering a unique sensor to characterize real user circumstances. The combination of multiple diverse sensors that individually capture the environment may result in a total picture to better derive user particular needs.

Setting bounds to and parameterize primary context can be done with relative ease. However, combination of two or more sensors and types of context introduces not only new possibilities, but also some added difficulties. In fact, situations and data from different sensors may relate, being thereby necessary to consider possible relations among them in order to understand as much as possible the situation the user is living while interacting with the environment. We are referring to multi-faceted characterizations of a contextual situation. Considering each sensor independently typically causes a different adaptation for each one, incurring sometimes on incompatible or even contradictory actions. In these cases where diverse context constraints enter into conflict, a more elaborated adaptation is required, looking for decisions in favour of merging all the individual considerations, as a result of substantial analysis and fusion of data from individual sensors.

Integration of diverse sensors was initially investigated in the Smart Badge, a device inspired by the Active Badge, equipped with a large variety of sensors to capture context beyond position [Smith, 1998]. This issue has also been considered in [Golding, 1999] for an indoor location technique that does not require any infrastructure. However, perhaps the most relevant work on multi-sensor context-awareness is the one carried out by [Gellersen, 2002]. They have been involved in a series of projects and reported experience from development of a number of device prototypes explored in various applications to validate multi-sensor to awareness. These include the Mediacup, a coffee cup with embedded awareness of its own state; the TEA awareness module and the Smart-Its platform, both used for aware mobile devices. Their research has provided new perspectives and insights into sensor fusion for awareness and inference of generic situational context. They have shown that integration of diverse sensors is a viable approach to obtain context representing real-world situations. Their experience suggests that some degree of abstraction can be implemented independently of specific applications. However, their work has not been specifically focused on architectural issues. In this sense, there is a need of future work to develop principles for the architectural design of multi-sensor context-aware systems.

As far as we are concerned, because of the complexity involved in integration of multiple diverse sensors, treatment of possible sensor interferences may not be handled locally on the mobile device. This situation is mainly due to the difficulty to include an inference engine in the target application. It would increase significantly the size of the application and the computation resources needed for the execution. Following the *Dichotomic View of plasticity* approach, all those situations not pre-established on the IPE are delegated to the server. To be more precise, the client requests the server for an *explicit plastic* adaptation, in this case focused on solving sensor interaction situations. It is worthy to say that these situations can be managed automatically –and in particular, by an *aspect* unit-, triggering the sending of the sensor values for all the implied context types. Following artificial intelligence

foundations, the server puts to work all the contextual parameters received and supports decision making by means of an inference engine specially prepared to derive some inferences about context constraint combination.

4 Infrastructure for Delivering and Supporting Mobile Context-Aware Applications

Our main objective is to provide a system in which originally non context-aware mobile applications are automatically prepared for context-awareness and then delivered to the client device. Apart from that, this system also offers a multi-device support along the execution of the applications. This section presents the system architecture operating in the server and the generic framework for context-awareness; the IPF. Particularly, it is shown how it applies in the development of specific components for case studies, showing its validity for different mobile technologies.

4.1 System Architecture for the Server

The system has been developed following client-server architecture. The server side has been developed using Java 2 Enterprise Edition technology, while the client side can be any device that supports Java 2 Mobile Edition or .NET Compact Framework. Figure 1 displays how the system works. Assuming that target application is already uploaded in the system, together with its associated configuration file (depicted by the step 1 in Figure 1), the IPE derivation tool proceeds to take it into consideration, in order to instantiate the IPF (depicted by the step 2 in figure). In order to be more precise, apart from performing the appropriate instantiation of the framework, that is, the completion of the code from the framework as explained in section 3.2, in step 2 the IPE designer is also in charge of (1) making a proper AOP library selection and (2) guaranteeing the proper platform-specific operation for each device. As a result, a component for context-awareness, specific for particular needs of target application is obtained (it consists in the block composed by the *context-aware* and *aspectual layers*, which is depicted by the step 3 in Figure 1).

Then, the system compiles and weaves this component together with the original code from the application initially uploaded (step 4 in Figure 1). The result is the expected IPE, that is, the original application now with specific support for context-awareness (step 5 in Figure 1). By this interlacing process, the IPE is obtained without neglecting the property of transparency that lays the foundations of the framework. In fact, any change is introduced, neither in the original source code nor in the internal structure. Once the IPE has been generated, it is registered and made accessible in its repository of applications, so that the client is able to download and install it in its device (step 6 in Figure 1). It must be taken into account that the system does not perform this process in a fully automatic way. The IPE designer conducts the instantiation process, according to parameters and particularities established in the configuration file.

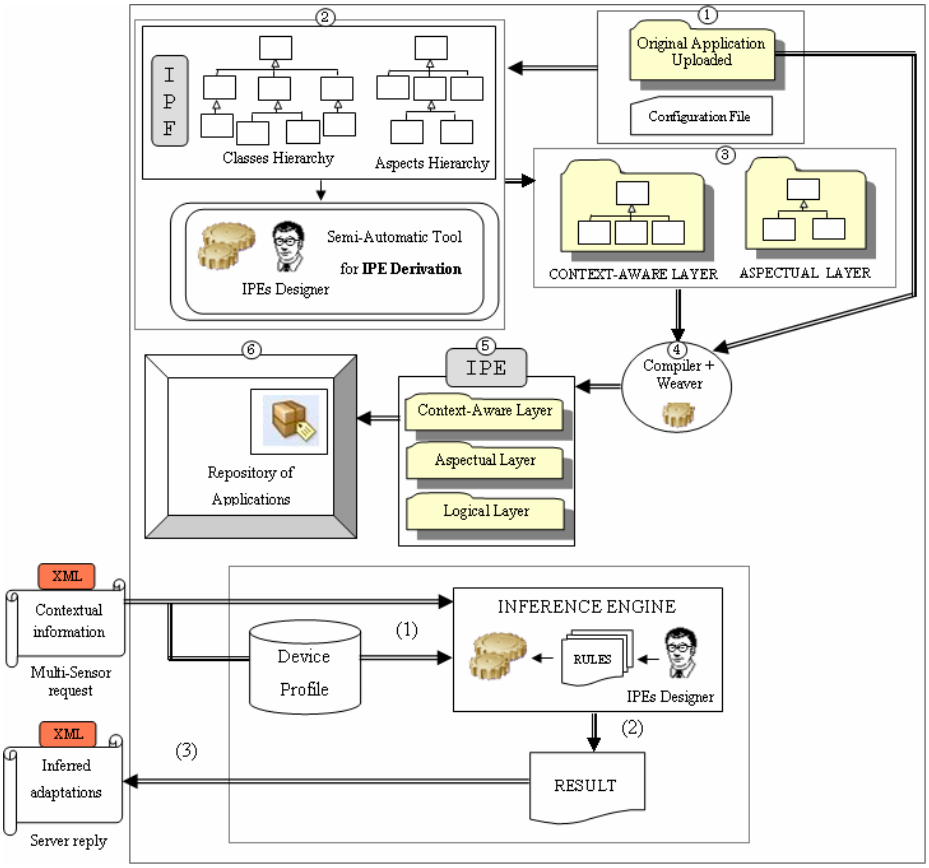

Figure 1: System architecture for the server

Since the application is delivered to the client, contextual information captured by sensors, which is used to apply automatic adjustments to the device's behaviour or the UI appearance, is gathered along the use of the application. Client application only communicates with the server in the cases that a possible conflict among different adaptations is detected. In these cases, the client application sends a request to the server with all the contextual information related to sensors involved in the interference. The system delivers received information to the inference engine together with the device profile of the client. The inference engine is a rule based system able to decide which adaptation best fits the client device and current situation according to the contextual information received, making inferences from existing rules. Additionally, it allows IPE designers to create new rules. In this case the system has made use of the Drools Expert rule engine. Once inferred which one among the different possibilities is the most appropriate, a reply is sent to the client specifying which adaptation or set of adaptations should be conducted.

It must be noticed that *aspect* inclusion is independent from the underlying technology used in the original code of the application. In this sense, the system includes *aspects* for J2ME applications by using the AspectJ library, while for .NET Compact Framework based applications PostSharp *aspects* library is used.

4.2 Implicit Plasticity Framework: a Generic Framework to obtain Components for Context-Awareness

Context-aware layer. The differences introduced in the IPF, in comparison to the IPE described in section 3.2, consist in the introduction of two abstract classes: *adaptatorAccodingConstraint* and *constraintDetector*. The former declares the *applyAdaptation()* method, to be implemented by the *ScreenRegulator* class in order to apply a particular reaction in the device. The latter declares the *obtainSensorValue()* method, to be implemented by the *Sensor* class in order to accede a certain sensor embedded to the device and capture its current value. As both functionalities vary from each technology and software platform, and consequently require access to a certain proprietary API, specific for each case, they are declared as abstract methods in an abstract class (or *interface*). As a result, the instantiation of the IPF to a particular technology consists in the implementation of these two methods, thus isolating these kinds of dependences from the rest of the code.

Aspectual layer. As mentioned in section 3.2, depending on the contextual situation that characterizes the use of a certain application, as well as on the way it is used, needs for adapting the system's behaviour vary significantly. We distinguish up to three levels of interactivity with sensors, thus establishing three different levels of operation for checking for a new adaptation. Each level of operation corresponds to a different design of the *aspect* unit. To manage this variety, the *aspectual layer* in the IPF is designed as a hierarchy of *aspects*, providing three different *sub-aspects*. In each case the most appropriate *sub-aspect* is instantiated. From left to right, they appear from the least interactive with the sensor to the most interactive one. They are the *ConstraintControlByDemand aspect* (punctual checking for new adaptations in particular points in the execution -*ActivateSensorConsultation() pointcut*); the *ConstraintControlInteractive aspect* (extends the checking points and introduces the *ControlChanges() pointcut* for monitoring the values captured by the sensor); and the *ConstraintControlBackground aspect* (activates a thread in background in order to carry out a cycled sensor checking). The *pointcut ActivateSensorConsultation()* is declared as an abstract *pointcut*, in order to be defined in each *sub-aspect*. However, the *CloseHistorical() pointcut* definition is inherited directly from the *aspect* base.

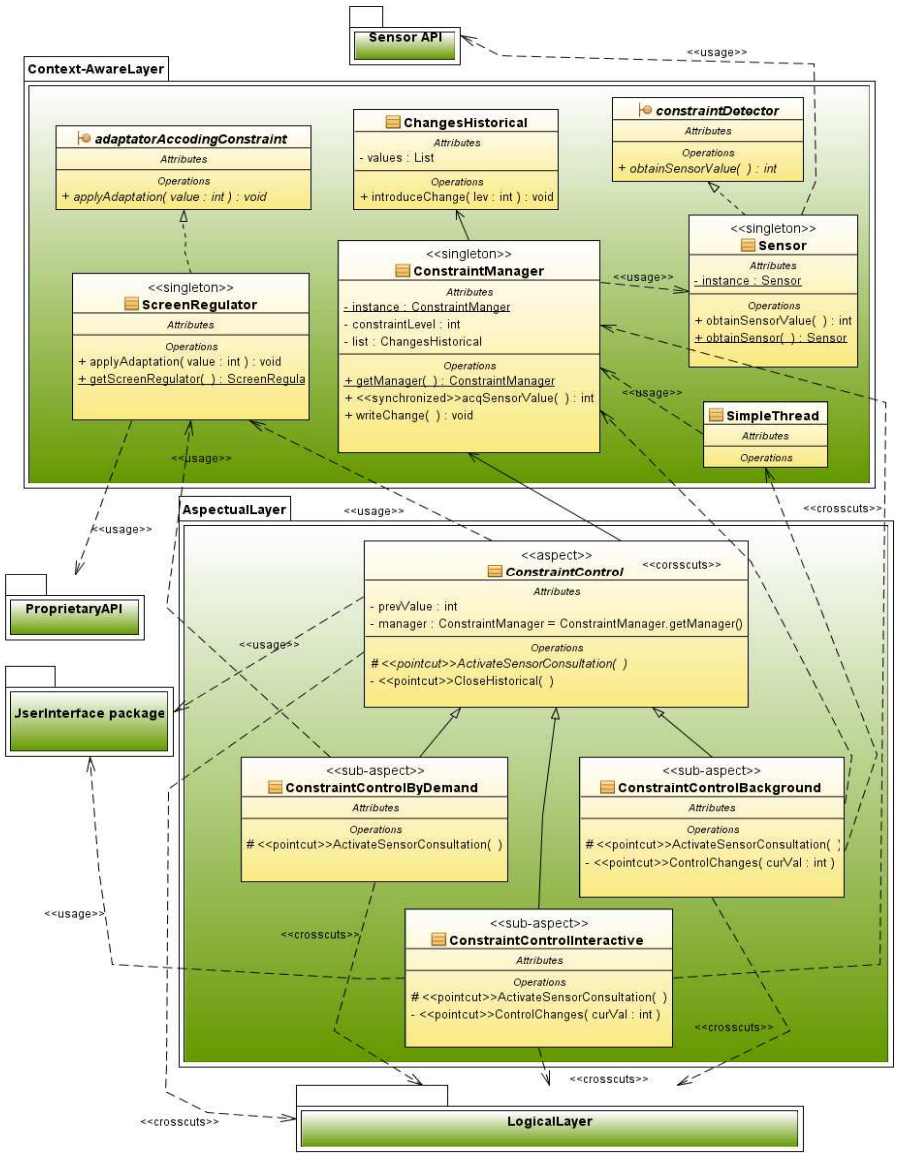

Figure 2: Generic component for context-awareness in the IPF

4.3 Application of the Framework to the Case Studies

Regarding the case studies presented in this subsection, it was aimed not only (1) to prove the validity of the proposed framework for the different mobile technologies employed in the case studies, but also (2) to explore the possibilities for sensing environment in each software platform.

4.3.1 Case Study 1: Brightness-aware J2ME Feeds Reader

The application in this case study is a mobile feeds reader application to download and manage news and announces from a company. The aim is allowing the staff to keep informed about all events in its worker's department, not only during the office stay, but also in different circumstances on the move and work journeys in which a laptop can not be used. It is in these kinds of circumstances that mobile Internet becomes a real help. This application has been developed in J2ME technology to work in mobile devices based on the S60 5th Edition software platform, that is to say, running under the Symbian operating system. In particular, it was installed and tested in a Nokia N97 mobile phone with Symbian 9.4.

Because reading pieces of news requires gazing at the screen for a while, possibly crossing different brightness conditions, it is likely to become not enough comfortable to the user. We have instantiated the IPF to obtain a component for context-awareness capable of making our feeds reader aware of the external brightness conditions, aiming to automatically adapt the screen brightness to surrounding brightness level.

Instantiation of such a component consists of next simple steps:

1. *Instantiation of the aspectual layer*. Taking into account the contextual needs for brightness adaptation, the *sub-aspect* most appropriate to be instantiated is the *ConstraintControlBackground* one, whose particularity consists in activating a thread, which is encapsulated on the *SimpleThread* class. The parts of code from the *aspect* to be completed are the next ones:
   - The *joinpoints* for the *ActivateSensorConsultation() pointcut*, which intercept the *commandAction* method from the *CommandListener* interface and the *itemStateChanged* method from the *ItemStateListener* interface from the *javax.microedition* package on the MIDP profile.
   - The *ControlChanges() pointcut* for monitoring the values captured for the sensor and deciding whether an adaptation has to be done or not. These parameters can be established in a configuration file.

   As it is depicted in Figure 3 the *aspect* unit for our component is called *BrightnessControlBackground*, and it conforms the *aspectual layer*.
2. *Instantiation of the context-aware layer*. It consists in the definition of two methods that encapsulate two platform-specific operations:
   - The *applyAdaptation(int)* method in *ScreenRegulator* class, which consists in a call to the method *setLights* of the class *DeviceControl* in the Nokia UI API, to alter the screen brightness.
   - The *obtainSensorValue()* method in *Sensor* class to enable data retrieval from the brightness sensor. It consists in a call to the *getData* method from the *SensorConnection* interface for a synchronous mode, and a call to the *dataReceived* method from the *DataListener* interface for an asynchronous mode. The latter method will be called by the *SimpleThread* class. Both, *DataListener* and *SensorConnection* interfaces belong to the Mobile Sensor API.

   As it can be observed in Figure 3, the main class in this layer is called *BrightnessManager*.

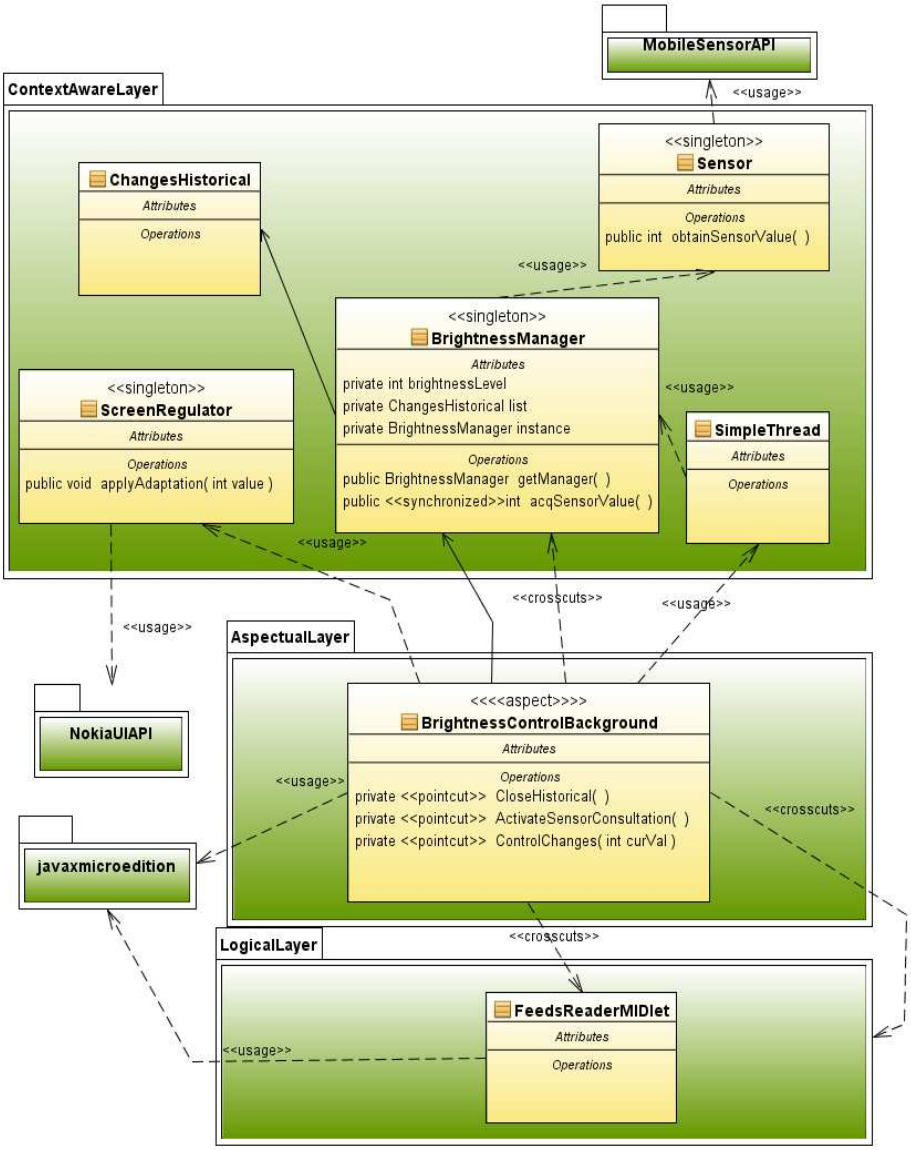

Figure 3: IPE for brightness-awareness in the Feeds Reader application.

4.3.2 Case Study 2: Sound and brightness aware .NET Compact Framework Tutor System

The purpose of the application in this case study has been to provide a user interface intended to help users with cognitive disabilities, elderly people and users who do not usually make use of new technologies. This application is intended to provide support

for performing common use tasks in mobile phones such as making a call or sending a text message. These tasks are explained with text descriptions, images and videos with didactic intention, allowing users to interact with the user interface in order to repeat actions such as repeating playback of a video. This application has been developed in C# using the .NET Compact Framework platform, so it works in mobile devices running under the Windows Mobile OS. In particular, it was installed and tested in a HTC Diamond Touch 2 mobile phone with Windows Mobile 6.

Taking the particularities of this case study into account, both sound and light related adaptations have been considered as both of them can provide remarkable benefits to end users. For instance, users with hearing impairments could benefit from the ability of the application to adjust the volume. The rest of the subsection describes the particularities for the instantiation of the IPF to obtain a sound component capable of making the application aware of the external sound conditions on sonorous environments. The component for brightness-awareness will be pretty the same as the one described previously. The steps followed to instantiate the sound component have been the same as for the Case Study 1:

1. *Instantiation of the aspectual layer*. Taking into account the contextual needs aforementioned, the most appropriate *sub-aspect* to be instantiated is the *SoundControlInteractive* one, whose particularity consists in establishing an interactive, but not necessarily constant, control over the sensor, which in this case is focused on the video reproduction periods of our Tutor System application. In order to instantiate this *aspect*, it is necessary:
   - Completing the *joinpoints* for the *ActivateSensorConsultation() pointcut*, intercepting the *play* method from the *player* class which controls the media player embedded in the application. In this case it is intended to capture a data stream from the sound sensor, so this information is later used to determine whether the volume of the video has to be adjusted to the level of the surrounding sound.
   - Completing the *ControlChanges() pointcut* for monitoring the values captured for the sensor and establishing the making decision.

   The *aspect* unit for this component is called *SoundControlInteractive*, and it conforms the *aspectual layer*.
2. *Instantiation of the context-aware layer*. As in the Case Study 1, it consists in the definition of the two following methods:
   - The *applyAdaptation()* method in *ScreenRegulator* class. For adjusting the sound, it makes a call to the *waveOutSetVolume* method of the *coredll* library in the Waveform Audio API.
   - The *obtainSensorValue()* method in *Sensor* class, which consists in a call to the method *AutoSetVolume* of *WaveIn* class. It calculates ambient sound by obtaining sound samples, storing them in a buffer and calculating their RMS.

UML diagram for this component has not included in the paper for extension reasons. It is pretty similar to the shown in Figure 3, replacing any reference to brightness by sound in the names of classes and *aspects*. Apart form that, the names of APIs also change, as this component is making use of particular packages for the Windows Mobile, and more specifically of the HTC proprietary API.

As this case study has two different possible adaptations regarding contextual elements, the fact of making an adaptation based on one of the contextual elements can interfere with adaptations performed with the other contextual element present. An example of conflict between proposed adjustments is the case in which the client is in an environment with a highly variable luminosity, i.e. on the train way. In case of deriving two adaptation proposals for such environments, the system may decide to give priority to the sound volume change adaptation instead of adjusting brightness, based on the fact that the system detects that luminosity change intervals are so short that it is not worthwhile to include this adjustment. To resolve such situations, as it has been shown, the adapted application communicates with the server, which proceeds to decide which adaptation deserves to be performed and which not. This decision is made according to the rules implemented in the server by the IPE designer that defines the procedures to follow in these interference scenarios.

Listing 1 displays a rule defined in the server using Drools. This rule decides which adaptation must be performed just in case of interferences between possible sound and brightness based adaptations are detected in the client. Prior to be used in the rule, current and mean values for both sound and brightness are normalized using the information provided in the device profile regarding the spectrum of values that sensors on client's device can cover. In this case, the rule decides which adaptation must be performed based on the difference between the absolute values between current and mean values for both sound and brightness.

```
package main

import main.AdaptationDataBean;
global main.AdaptationOutputBean outputBean;

rule decideAdaptations
   when
      data: AdaptationDataBean()
   then
      int ambientSound = Math.abs(data.getAmbientSoundMean() -
data.getCurrentAmbientSound());
      int brightness = Math.abs(data.getBrightnessMean() - data.getCurrentBrightness());
      if (ambientSound > brightness)
        outputBean.enableAmbientSound();
      else
        outputBean.enableBrightness();
end
```

Listing 1. A rule designed to solve brightness and sound interferences

5 Conclusions and Future Work

The main goal of this paper is to show the advances that developed framework involves about providing adaptation mechanisms to context-awareness in mobile applications. The most relevant issue can be claimed saying that it is a multi-purpose oriented infrastructure, as it contemplates: (1) multi-sensor for a more enriched

awareness of situational context; (2) multiple underlying mobile technologies; and (3) multiple contextual needs, always concordant with the application main goal. It is remarkable that the only pre-conditions for the target application in order to be prepared as a context-aware application by the framework are two. On the one hand, that the source code follows a object-oriented paradigm. On the other hand, for this particular case, the source code must be implemented in a programming language supported by the framework (both in the base language and in the corresponding AOP library). Currently, the languages supported by the framework are Java and C#, and the AOP libraries are AspectJ and PostSharp. Any application fulfilling these restrictions can be prepared to present a context-aware behaviour.

Another remarkable advance developed system provides is a substantial improvement regarding the automation. Apart from uploading source code and downloading the application once it is ready for context-awareness, human being intervenes only in two moments along the delivering process. One of them is to specify the rules that would allow the server to decide what to do when adaptations to be triggered in the client enter into conflict. This allows not only extending the bank of rules existing, but also personalizing the treatment of specific situations. Human being also intervenes in the framework instantiation process, as it is pointed out next.

The fact that the system provides independence from the underlying mobile technology used in the client device in order to support context-awareness is another remarkable advantage. The case studies demonstrated the validity of the system for different mobile technologies as it has been tested in different mobile software platforms, such as the S60 5th Edition and the .Net Compact Framework. There have been a series of handicaps that have had to be overcome. For instance, the difference in the support for AOP has been a major issue, as the different existing libraries and weaving process did not work on exactly the same way for the different platforms used. Additionally, the need of APIs provided by vendors in order to access to information provided by sensors and adjust the UI needs a continuous updating and is another major issue. These kinds of considerations irremediably need the human supervision, task that is carried out by the IPE designer. However, once the application is running, both implicit plastic and explicit plastic adaptations are performed in a completely automatic way.

It is also noticeable that proposed infrastructure deals with the integration of diverse context constraints. In this sense, applications can be prepared for sensing multiple diverse sensors on the device. Additionally, a major support is also considered in server in order to tackle sensor interferences. It is expected to gain access to rich data from which useful context can be inferred with comparatively little computation. It is worthy to say that the IPF component for context-awareness, such as it has been described in paper, is also useful for the treatment of a more particular context-awareness field. We are referring to sensors giving information of the device state, such as network field intensity and battery charge sensor. Combination of environmental and device state sensors can lead to a very interesting experimentation in context-awareness for mobile applications. In this line, it is necessary to include support for more vendor APIs related to gathering information for different kind of sensors. As the *aspectual layer* works independently of these APIs, we could even experience with sensors used for example in the field of wearable computing.

These three main points described above make up significant advances developed over the original *Dichotomic View of Plasticity* infrastructure as presented in [Sendín, 2009], and fulfils six of the seven requirements defined by [Dey, 2000], that is, all of them except the one related to resources discovery. In particular, the context interpretation requirement is related with the combination of diverse sensors. It is remarkable that the original *Dichotomic View of Plasticity* infrastructure already contemplated any kind of contextual need and adaptation mechanism to be considered. We are referring to components for user personalization, dynamic hardware adaptation and different components for context-awareness.

In near future it is foreseen to include support not only for more mobile development technologies (i.e. Android, Symbian^2, Blackberry, Linux and iPhoneOS). Another interesting challenge is to extend the framework for other kinds of environments (not necessarily mobile), such as groupware development, where context also makes up a major issue, thus extending the kinds of context supported by the framework, as well as its multi-purpose goal. This objective adds in the requirement of providing proper AOP support for each platform so the *aspectual layer* behaves the same way in all of them.

Acknowledgements

This work has been partially funded by Spanish Ministry of Science and Innovation through TIN2008-06228 and TIN2008-06596-C02-01 research projects. Authors would like to thank Juan José Pardo, Rosa Maria Jiménez and Miquel Canfran for their helpful suggestions.

References

[Couto, 2005] Couto, R., Endler, M. Evolutionary and efficient context management in heterogeneous environments. Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing ISBN:1-59593-268-2

[Dey, 2000] Dey, A. K. Providing architectural support for building context-aware applications. Thesis dissertation, College Computing, Georgia Institute of Technology, 2000. ISBN 0-493-01246-X.

[Gellersen, 2002] Gellersen, H. W., Schmidt, A., & Beigl, M. (2002). Multi-sensor context-awareness in mobile devices and smart artefacts. Mobile Networks and Applications, 7(3), 341–351.

[Golding, 1999]A. Golding and N. Lesh, Indoor navigation using a diverse set of cheap wearable sensors, in: Proceedings of the IEEE International Symposium on Wearable Computing (ISWC'99), San Francisco, CA (October 1999) pp. 29–36.

[Gómez 2009] Gómez, S., Huerva, D., Mejía, C., Baldiris, S., Fabregat, R. Designing Context-Aware Adaptive Units of Learning Based on IMS-LD Standard, EAEEIE 2009 Conference.

[Leonhardi, 1999] Leonhardi, A., Kubach , U., Rothermel , K., Fritz, A. Virtual Information Towers-A Metaphor for Intuitive, Location-Aware Information Access in a Mobile Environment. Proceedings of the 3rd IEEE International Symposium on Wearable Computers table of contents. ISBN:0-7695-0428-0

[Malandrino 2009] Malandrino, D., Mazzoni, F., Riboni, D., Bettini, C., Colajanni, M., Scarano, V. MIMOSA: context-aware adaptation for ubiquitous web access. Personal and Ubiquitous Computing (Article accepted for publication)

[Schilit, 1995] Schilit, W.N. A system architecture for context aware mobile computing. Thesis disssertation, Columbia University, New York, USA, 1995. UMI Order Nº GAX 95-33659.

[Sendín, 2005] Sendín, M., Lorés, J. Towards the Design of a Client-Side Framework for Plastic UIs using Aspects. Proceedings of the International Workshop on Plastic Services for Mobile Devices (PSMD05), in conjunction to Interact 2005. 2005.

[Sendín, 2006] Sendín, M. Implicit Plasticity Framework: a Client-Side Generic Framework for Context-Awareness. CEUR-Workshop Proceedings. ISSN: 1613-0073. Vol. 208. 2006

[Sendín, 2007] Sendín, M. Infraestructura Software de Soporte al Desarrollo de Interfaces de Usuarios Plásticas bajo una Visión Dicotómica. PhD. Thesis, University of Lleida, 2007

[Sendín, 2009] Sendín, M., López, JM. Contributions of Dichotomic View of plasticity to seamlessly embed accessibility and adaptivity support in user interfaces. Advances in Engineering Software 40 (2009) 1261–1270

[Smith, 1998] M.T. Smith, SmartCards: Integrating for portable complexity, IEEE Computer (August 1998) 110–112, 115

[van Biljon, 2008] van Biljon, J., Kotzé, P. Cultural Factors in a Mobile Phone Adoption and Usage Model. Journal of Universal Computer Science, vol. vol. 14, no. 16 (2008), 2650-2679

[Yaici, 2008]Yaici, K. Kondoz, A. A model-based approach for the generation of adaptive user interfaces on portable devices. IEEE International Symposium on Wireless Communication Systems. 2008. ISWCS '08. On page(s): 164-167. ISBN: 978-1-4244-2488-7

[Zurita, 2007] Zurita, G., Antunes, P., Baloian, N., Baytelman, F. Mobile Sensemaking: Exploring Proximity and Mobile Applications in the Classroom. Journal of Universal Computer Science, vol. 13, no. 10 (2007), 1434-1448