Context Aware Adaptable Applications - A global approach

Marc Dalmau, Philippe Roose, Sophie Laplace
LIUPPA, IUT de Bayonne
2, Allée du Parc Montaury
64600 Anglet
FRANCE
Mail : {FirstName.LastName@iutbayonne.univ-pau.fr}

Abstract

Actual applications (mostly component based) requirements cannot be expressed without a ubiquitous and mobile part for end-users as well as for M2M applications (Machine to Machine). Such an evolution implies context management in order to evaluate the consequences of the mobility and corresponding mechanisms to adapt or to be adapted to the new environment. Applications are then qualified as context aware applications.

This first part of this paper presents an overview of context and its management by application adaptation. This part starts by a definition and proposes a model for the context. It also presents various techniques to adapt applications to the context: from self-adaptation to supervised approaches.

The second part is an overview of architectures for adaptable applications. It focuses on platforms based solutions and shows information flows between application, platform and context. Finally it makes a synthesis proposition with a platform for adaptable context-aware applications called Kalimucho. Then we present implementations tools for software components and a dataflow models in order to implement the Kalimucho platform.

Key-words: Adaptation, Supervision, Platform, Context, Model

1. Introduction

Actual applications (mostly component based) requirements cannot be expressed without a ubiquitous and mobile part for end-users as well as for M2M applications (Machine to Machine). Such an evolution implies context management in order to evaluate the consequences of the mobility and corresponding mechanisms to adapt or to be adapted to the new environment. Mobile computing and next, ubiquitous computing, focuses on the study of systems able to accept dynamic changes of hosts and environment [33] . Such systems are able to adapt themselves or to be adapted according to their mobility into a physical environment. That implies dynamic interconnections, and the knowledge of the overall context. Due to the underlying constraints (mobility, heterogeneity, etc.), the management of such applications is complex and requires considering constraints as soon as possible and having a global vision of the application.

Adaptation decision can be fully centralized (A - Figure 1) or fully distributed with all intermediary positions (B&C - Figure 1). The consequence is the level of autonomy of decision as well as the level of predictability. Obviously, the autonomy increases with decentralized supervision. Reciprocally, the complexity increases with the autonomy (problems of predictability, concurrency, etc.).

Self-adaptable applications need to access to context information. This access can be active if the application captures itself the context (see A - Figure 1), or passive if an external mechanisms gives it access to the context (see B - Figure 1). Nevertheless, with mobile peripherals and the underlying connectivity problems, a fully centralized supervision is not possible. A pervasive supervision Error! Source du renvoi introuvable. appears as a good solution and allows managing complexity, predictability while keeping the advantages of autonomy.

In order to be context-aware, applications need to get information corresponding to three adaptation types: data, service and presentation. The first one deals with “raw data” and its adaptations to provide complete and formatted information. Service adaptation deals with the
architecture of the application and with dynamic adaptation (connection/disconnections/migration of components composing the application). It allows adapting the application in order to respect the required QoS. Presentation deals with HCI (not addressed in this paper).

Here is a global schema of an adaptable application:

Whereas [34] [35] do not make distinction between context oriented and application oriented data (functional data), we think that such a distinction makes design easier and offers a good separation of concerns [36].

2. What is context?

2.1 Definition and model

The origin of the term « context awareness » if attributed to Schilit and Theimer [42]. They explain that it is « the capacity for a mobile application and/or a user to discover and react to situations modifications ». Many other definitions can be found in [43]. The application context is the situation of the application so the context is a set of external data that may influence the application [36].

A context management system can interpret the context and formalize it in order to make a high level representation. It creates an abstraction for the entities reacting to situations evolutions, they can be applications [35], platforms, middlewares, etc. In order to make such abstractions, a three layered taxonomy can be organized as shown in Figure 3:

The first layer deals with context information capture. The first type of context is called Environmental: this is the physical context. It represents the external environment where information is captured by sensors. This information is about light, speed, temperature, etc. The second type, called User, gives a representation of users interacting with the application (language, localization, activity, etc.). This is the user profile. The third one deals with Hardware. Most probably, the more “classical” one; it gives information on available resources (memory, % CPU, connections, bandwidth, debit, etc.). It also gives information as displays resolutions, type of the host (PDA, Smartphone, Mobile Phone, PC, etc.). The third one is the Temporal context. It preserves and manages the history (date, hour, synchronizations, actions performed, etc.). The last one is called Geographic and gives geographical information (GPS Data, horizontal/vertical moving, speed, etc.).

The second layer, called « context management » [44] [45] is based on the previous layer representations. It provides services specifying the software environment of the application (platform, middleware, operating system, etc.). The Storage of context data in order to allow services retrieving them, the Quality giving a measure about the service itself or data processed and the Reflexivity allowing to represent the application itself. The localization manages geographic information in order to locate entities, predict their displacements.

The last layer proposes mechanisms to permit the adaptation to the context. We will find several mechanisms in order to react to contextual events. The first one is the software component Composition, the second one is the Migration in order to move entities and the last one, the Adaptation to ensure the evolution of the application. This last point is no-functional, the middleware manages it, it can depend on a user profile or on rules provided by the user. The polymorphism facilitates the migration of entities and their adaptation to various hosts (with more or less constraints).

We propose a context model able to design any context information. This model (called Context Object) provides information needed by entities managing the application. Some information defines the context (its nature) whereas others define its validity. The nature of the context can be [34]:

![Figure 3: Taxonomy of context](image-url)
- **User** (people) as his preferences,  
- **Hardware** (things) as network,  
- **Environment** (places) as temperature, sunlight, sound, movement, air pressure, etc. It is the physical context. It represents the external environment from where information is captured by sensors. It deals with users' environment [36] as well as hardware environment.

Such information is called **ContextInformation** and we call **InformationValidity** the validity area of a **ContextInformation** (example: old information or information which source is very far can be useless).

**InformationValidity** is:
- **Temporal**: Temporal information can be a date or time used as timestamp. Time is one aspect of durability so it is important to date information as soon as it is produced. This temporal information allows making historical report of the application and defining the validity (freshness) of **ContextInformation** [40]. This freshness is the time since the last sensor reads it. Ranganathan defines a temporal degradation function that degrades the confidence of the information.
- **Spatial**: it is the current location (the host (identity) or the geographic coordinates (GPS)) and makes possible to distinguish local and remote context
- **Confidence information**: how precise is the sensor
- **Information ownership**: in some application hosted on a Smartphone for example, privacy is very important, therefore, each information has to be stamped with its owner (the producer).

Let’s notice that some information is strongly coupled as freshness and confidence whereas others are defined using application data as ownership. That is the reason why [39] identified physical, virtual (data source from software application or services) and logical sensors (combine physical and virtual sensors with additional information) Depending on the application, one information type could be a **ContextInformation** or a **ValidityInformation**. For example, location can be a **ContextInformation** for a user in a forest or can be a **ValidityInformation** for the sensor network that supervises temperature and air pressure measurement.

According to this model, we organize all the characteristics of context information that define type, value, time stamp, source, confidence and ownership [37] or user, hardware, environment and temporal [45] [46] [47]. In order to structure such contextual information, we proposed a meta-model structuring **ContextInformation** and **ValidityInformation** (see Figure 4).

2.2 Context and applications

Since several years, the natural evolution of applications to distribution shows the need of more than only processing information. Traditionally, applications are based on input/output, i.e. input data given to an application produces output data. This too restrictive approach is now old fashioned [48]. Data are not clearly identified, processes does not only depend on provided data but depend also on data such the hour, the localization, preferences of the user, the history of interactions, etc. in a word the context of the application.

We can find a representative informal definition in [49] "The execution context of an application groups all entities and external situations that influence on the quality of service/performances (qualitative & quantitative) as the user perceives them". Designers and developers had to integrate the execution environment into their applications. This evolution allows applications to be aware of the context, then to be context-sensitive and then to adapt their processes and next to dynamically reconfigure themselves in order to react as well as possible to demands. This is evidence, but to adapt itself to the context, the application needs to have a good knowledge of it and of its evolutions.

With a research point of view, context needs a vertical approach. All research domains/layers manage contextual information. Many works deal with its design, management, evaluation, etc. Its impact is wide: Re-engineering, HCI, Grid, Distributed Applications, Ubiquitous Computing, Security, etc. But to be honest, the context it not a new concept in computer science! Since the early 90’s, Olivetti Research Center with the
ActiveBadge [Harter, 1994] and most of all, with a lot a regrets, the Xerox PARC with the PARCTab System [51] gave the bases of modern context aware applications.

In order to be aware of the context, the following architecture (see Figure 5) is "classical". An example can be found in [46]. It can be summarized as a superposition of layers. Each of them matches to a contextual information acquisition process, a contextual information management and an adaptation of the application to the context (as defined in Figure 3).

According to Figure 5, context management do imply to have dynamic applications in order to adapt them to variations of the context and so to provide a quality of service corresponding to current capabilities of the environment (application + runtime).

3. Context aware applications

Context aware applications are tightly coupled to mobiles devices and ubiquitous computing in the meaning of "machines that fit the human environment, instead of forcing humans to enter theirs" [1]. These applications are able to be aware of their execution context (user, hardware and environment) and its evolutions. They extract information from the context about geographical localization, time, hardware conditions (network, memory, battery, etc.) as well as about users.

Interactions between an application and its context can then be represented by two information flows (Figure 6):
- Application captures information from its context
- Application acts its context

Achievement of a context aware application can be done:
- By self adaptation
- By supervision

However, even if it is possible to design limited applications according to the use of contextual information, the main interest is to be able to adapt the behavior of the applications to the context evolutions. Particularly, the increasing use of mobile and limited devices implies the deployment of adaptable applications. Such approach allows having a quality of service management (functional and non-functional services as energy saving for example).

3.1 Adaptable context aware applications

Adding adaptation to context aware applications means the addition of a new interaction corresponding to the influence that the context has on the application. That is the property for the application to adapt itself to the context (Figure 7).

The means operated to realize both data flows of the Figure 6 depend on types of context (Table 1). They are system and network primitives for hardware context (resource allocation, connections, consultation of available resources, etc.). The user's context is captured through the interfaces and the information system (user's profile description files). At last, environmental context can be captured through sensors and modified by actuators.

<table>
<thead>
<tr>
<th>Flow</th>
<th>Hardware</th>
<th>Type of context</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>System and network primitives</td>
<td>Interfaces and information system</td>
<td>Sensors</td>
</tr>
<tr>
<td>#2</td>
<td>Resource allocation</td>
<td>Interfaces</td>
<td>Actuators</td>
</tr>
</tbody>
</table>

Table 1: Means of interaction between application and context
3.1.1 Self adaptation

Such systems are expected to dynamically self-adapt to accommodate resource variability, change of user needs and system faults. In [27], self-adaptive applications are described as useful for pervasive applications and sensor systems. Self-adaptive applications mean that adaptations are managed by the applications itself. It evaluates its behavior, configuration, and with distributed application, its deployment. The application captures the context (flow #1) and therefore adapts its behavior (data flow #3). The activity of the application modifies the context (flux #2). This approach, represented in Figure 7, raises the essential problem of accessing to distant context information. Indeed, through the interactions described in Table 1 it is only possible for the application to interact with its local context. In order to get or modify distant contextual information, the designer of the application has to set up specific services on the different sites of its application. It becomes necessary to set up many non functional mechanisms that strongly increases the complexity of the application and are difficult to maintain up to date. Moreover self-adaptive solutions imply to have a planning and an evaluation part at runtime and a control loop. In order to make the evaluation, such application needs components description, as well as software description, structure and various alternatives, i.e. various assembling configurations.

Such solutions do not simplify the separations of concern, and so increase the practical viability of the application and its maintainability and possible evolutions. Moreover, with ubiquitous and heterogeneous environments, such generic solutions are not suitable to exploit the potential of hosts [28]. That are the reasons why most systems tend to solve these problems using platforms.

3.1.2 Supervised adaptation

In these approaches a runtime platform interfaces the application and the context. It allows then access to distant context. The application only senses the context (flow #1) by means of the middleware of the platform. The application can modify the context and the platform itself (flow #2). Both the application and the platform adapt themselves to the context (flow #3). This kind of organization is shown in Figure 9.

Recent works as Rainbow use a closed-loop control based on external models and mechanisms to monitor and adapt system behavior at run time to achieve various goals [32], such solution is closed to the use of pervasive supervision. In order to implement such a solution, we need a distributed platform on all heterogeneous hosts. Such architecture allows to capture local context, and to propose local adaptations. Additionally, communication between local platforms gives a global vision of the context permitting to have a global measure of the context and adapted reactions.

Each platform has three main tasks to accomplish:

- **Capture of the context**: This task is important and implements tools to capture information of layer 1 (see Figure 3).
- **Context Management Service**: Its role is to manage and evaluate information from layer 1 in order to evaluate if adaptation is required.
- **Context Management Tools**: It proposes a set of mechanisms to adapt the application because of variations of the context.

The means operated to realize data flows #1 and #2 of Figure 9 depends on the types of context (Table 2). Interactions with local context use the mechanisms described in (Table 1) whereas those with distant context use services of the platform. The middleware of the platform offers services for context capture providing contextual information completed by time and localisation parameters as described in Figure 4.

<table>
<thead>
<tr>
<th>Flow</th>
<th>Type of context</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Hardware, Interfaces, Sensors</td>
<td>Local</td>
</tr>
<tr>
<td></td>
<td>System and network</td>
<td></td>
</tr>
</tbody>
</table>
The role of the platform in this kind of organisation becomes central. We will now define more precisely the role and the architecture of a platform.

3.2 The platforms

Generally, we consider a platform as a set of elements of virtualization (Figure 10) allowing application designers to have a runtime environment independent of the hardware and network infrastructures, supporting distribution and offering non functional general services (Persistence, Security, Transactions …) or services specific to a domain (business, medical …).

<table>
<thead>
<tr>
<th>primitives</th>
<th>Services of the platform</th>
<th>Services of the platform</th>
<th>Services of the platform</th>
<th>Distant</th>
</tr>
</thead>
<tbody>
<tr>
<td>#2 Resource allocation</td>
<td>Interfaces</td>
<td>Actuators</td>
<td>Local</td>
<td></td>
</tr>
<tr>
<td>Services of the platform</td>
<td>Services of the platform</td>
<td>Services of the platform</td>
<td>Distant</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Interactions between Application and Context with a Platform

The container virtualizes the application or its components in make them suitable and compatible (interface) with the platform. The framework finishes this task allowing the designer to respect the corresponding model. The middleware virtualizes communications and offers services called by the application in order to access to the context. At last heterogeneity consists in virtualization of the hardware and the operating systems on which the application runs.

Interactions between platform and application are bidirectional and represent the core aspect of the whole system (platform/application). The platform has its proper state evolving when modifications occur in the underlying level (context) and in the application. Consequently, the platform can trigger updates of the application state.

The interaction mode between platform and application allows distinguishing two families (Figure 12):

- **Non intrusive platforms**;
- **Intrusive platforms**.

A **non intrusive** platform acts on external elements of the application like data or uses a event based mechanism. It raises events when an internal state change occurs. These events can be caught by specific components of the application (event listeners). These modifications of external elements and these events imply the changes of the application state.

An **intrusive** platform can directly change the state of the application without participation of the application. This can be achieved by a direct action on the functional part either by a modification of the circulating of information either by directly modifying the architecture of the application itself by services, API or middleware calls (Figure 11 left), while in the second case the containers of the business components send to the platform information about their evolution (Figure 11 right). These containers can themselves offer some services to the business components or capture information about their changes of state by observing their behavior.
Interactions between application and platform can be described as follow:

- Data Flow A corresponds to information from the application to the platform through usage of services of the middleware.
- Data Flow B represents the possibility to the application to configure the behavior of the platform (events priorities, filtering of contextual information, etc.)
- Data Flow C corresponds to the non intrusive mode of interaction between platform and application. It deals with events produced by the platform for the listeners inside the application.
- Data Flow D represents the intrusive mode of interaction between platform and application. It deals with updates of the application by the platform (modification of the architecture by adding/suppressing/moving components or by changing their business part).

Now, let’s have a look on different context aware applications types that can be build according to data flows really used. Firstly it is important to notice that for context aware applications, data flow A is essential. In order to be adaptable, at least flow C or flow D need to be provided. If not, the platform is the only one able to be adaptable. The optional data flow B represents the possibility that the application has to configure the interaction modes corresponding to the flows A, C and D.

The Table 3 presents the four models of adaptation that it is possible to realize according to the flows used:

<table>
<thead>
<tr>
<th>Flows used</th>
<th>Type of interaction</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 A</td>
<td>The platform is a middleware (services for accessing to local and distant context)</td>
<td>Only the platform is able to adapt itself to the context</td>
</tr>
<tr>
<td>2 A and C</td>
<td>The platform is a middleware (services for accessing to local and distant context) and offers an adaptation service</td>
<td>Adaptation is decided by the application according to information send by the platform</td>
</tr>
<tr>
<td>3 A and D</td>
<td>The platform is a middleware (services for accessing to local and distant context) and supervises the adaptation</td>
<td>Adaptation fully supervised</td>
</tr>
<tr>
<td>4 A, C and D</td>
<td>The platform is a middleware (services for accessing to local and distant context) and offers an adaptation service</td>
<td>Adaptation is partially supervised and partially decided by the application</td>
</tr>
</tbody>
</table>

Data flow B allows to enrich the interaction types presented in the above Table 3:
In the first case: the application only can configure the services of context access provided by the platform.

In the second case: the application can also choose the events which are indicated to it and their priority.

In the third case: the application can configure the level of intrusion of the platform and eventually protect itself from it at some moments.

The fourth case is the union of the two before.


We can then draw a general schema of an adaptable context aware application (Figure 15).

The platform is distributed on every device hosting components of the applications. Then it can access to all contextual information. It offers a set of services in order to allow the application accessing to local or distant context (data flow A). Moreover it includes an adaptation manager sending events (data flow C) and a manager supervising the application (data flow D). The execution of this supervision manager can be configured by the application (data flow B).

3.4 Functional model of adaptation

The execution of an adaptable context aware application looks like a looped system: the context modifies the application, the execution of the application modify the context and so on. When a platform is introduced between the context and the application, a new loop appears because the platform itself is modified by the context and reciprocally, the platform modifies the context. Depending on using an intrusive or a non intrusive platform model, these loops are achieved by different data flows.

3.5 General architecture of a platform for adaptable context aware applications

The platform is composed of three main parts:
1. The capture of context is done by usual mechanisms as described in Table 1. They are system and network primitives, information system and sensors. Moreover, the platform also receives information about the application’s running context from the containers of the business components (Figure 10).
2. The services concern both the application and the platform itself (more precisely the part in charge of the adaptation):

For the application it corresponds to:
- Services for accessing to the context (hardware, user, environment) with filtering possibilities (time, localisation)
- Other usual services (persistence, ...)

For the adaptation it means:
- Services for accessing to the context
- Services for Quality of Service measurement
- Services for reflexivity that is to say the knowledge that the system constituted by the platform and the application has of itself.

3. The adaptation matches the general schema of adaptation proposed in [3] which distinguishes two parts:
- The evolution manager which implements the mechanisms of the adaptation;
- The adaptation manager which monitors and evaluates the application.

4. Kalimucho platform and implementation tools

The architecture of the application has to be virtualized in order to be monitored by the platform. The general architecture of the Kalimucho platform is the following:

4.1 Kalimucho architecture

We propose to build the architecture of adaptable context aware applications on a distributed platform called Kalimucho.

The application is made of business components (BC) interconnected by information flows. To directly modify the architecture of the application it is necessary that the platform should be able to add/remove/move/connect/disconnect the components. Moreover the platform has to capture the context on every site. We created a container for information data flows named Korrontea and another for business components named Osagaia [26]. These containers collect local contextual information from business components and connectors and send them to the platform. They receive back supervisions commands from the platform. Interactions between the platform and the application are implemented with the flows shown in Error! Source du renvoi introuvable. We can notice that because...
Korrontea containers have a non functional role into the application (information transportation), they do not accept the data flow C and are not event listeners. On the other hand, some BC can react to context events sent by the platform towards Osagaia containers.

Our work deals with various devices as sensors (which are CLDC compliant), PDA, SmartPhones (CDC compliant) and traditional PCs. Such an heterogeneous environment implies several services variations devoted to the platform: The capture of the context is done by components (Osagaia) and flow containers (Korrontea). Depending on the host running the component, it will capture users, environment, hardware, temporal or geographic information (see layer 1 - Figure 3). The second layer (context management services) is done by implementing an heuristic in order to evaluate the current Quality of Service (QoS) and to propose adaptations if needed and if possible. The last layer (context management tools) gives solutions to provide adaptations (add/remove/move/connect/disconnect components).

The platform is distributed on every machine on which components of the application are deployed (desktops, mobile devices and sensors). The different parts of the platform communicate through the network. Communications between BCs (local or distant) are achieved by data flows encapsulated into Korrontea containers.

Various versions of the platform are implemented on the different hosts according to their physical capacities. On a desktop all the parts of the platform are implemented whereas, on a mobile device, and particularly on a wireless sensor, light versions are proposed (one for CDC and one for CLDC compliant hosts). Consequently, only non avoidable services for the host are deployed (for example a service for persistence is useless on a sensor).

In the same way, the adaptation manager implemented on a mobile device can be lightened using internal services of one of the neighbouring platform (for example, only local routing information is available on a limited device). If the platform of this device needs to find others routes in order to set up a new connection, it has to use services of the platforms implemented on neighbouring desktops.

Finally we design the software component model in order to ensure the implementation of distributed applications according to the specifications expressed by functional graphs.

Functional components are called business component since they implement the business functionalities of applications. These components need to be executed into a container whose role is to provide non-functional implementation for components. The architecture of this container is shown in Figure 21, we call it Osagaia. Its role is to perform interactions between business components and their environment. It is divided into two main parts: the exchange unit (composed of input and output units, see Figure 21) and the control unit. The exchange unit manages data flows input/output connections. The control unit manages the life cycle of the business component and the interactions with the runtime platform. Thus, the platform supervises the containers and, indirectly, the business components (a full description of the Osagaia software component model is available in [31]). Thanks to this container, business components read and write data flows managed by Connectors called Korrontea (see Figure 22). Its main role is to connect software components of the applications. The Korrontea container receives data flows produced by components and transports them. It is made up of two parts. The control unit implements interactions between the Korrontea container and the platform while an exchange unit manages the input/output connections with components. The container is the distributed entity of our model, i.e. it can transfer data flows between different sites of distributed applications. The flow management is done according to the business part of the connector implementing both the communication mode (client/server for example) and the communication politic (with or without synchronization, loss of data, etc.). A full description of the Korrontea component model is available in [28].

Figure 20: Interactions between application and platform in Kalimucho

Figure 21: Osagaia Conceptual Model
5. Conclusion

In this paper, we presented an overview of adaptable applications. Because such applications need knowledge of their environment, we made a definition of the context and presented it according to applications uses. Next, we present adaptation management politics and their possible implementation. This part was followed by a presentation of implementation tools able to provide adaptations. We finished by the description of the Kalimucho platform, software and connectors containers models used in order to make adaptations.

Implementing context-aware adaptable applications with a platform helps having a global view of the application and of the context. The global view of the application permits an optimum mobility and resource management. The global view of context permits considering the whole context of the application instead of the only local one.

The problem of such an approach is its inherent complexity. Context aware platforms become more and more complex in order to manage a context more and more variable and evanescent. So, depending on the targeted application, it could be much more interesting to provide various lighter, specialized and reflexive platforms providing a view of their state. Moreover, such platforms are able to be heaped with other light, specialized and reflexive ones.

The influence of the environment on the system behavior leads to strongly couple the execution platform and the application[38]. So design methods for applications and platforms must also be coupled to constitute a sole design method.

Instead of making a whole design step, we propose a life-cycle including both application and platform (which is also an application – this is recursive) to finish with implementation tools (platform specific, component and connector models and specific implementations). Such approach let us imagine wide development with automatic code generation.

6. Bibliography


[42] Bill Schilit, Marvin Theimer - Disseminating Active Map Information to Mobile Hosts - IEEE Network, September, 1994


