Context-Aware Transportation Services (CATS) Framework for Mobile Environments

Dana Popovici, Mikael Desertot, Sylvain Lecomte and Nicolas Péon
Univ Lille Nord de France, F-59000 Lille, France
UVHC, LAMIH, F-59313 Valenciennes, France
CNRS, FRE 3304, F-59313 Valenciennes, France
firstname.surname@univ-valenciennes.fr

Nowadays there are many applications that users can benefit from on their handheld devices: localization, navigation, e-commerce, social networks and many others. Such capabilities are reaching our vehicles, offering drivers help for driving safely and more efficiently, thanks to the numerous services provided by applications on their devices. To simplify application assembly and reactivity according to transportation constraints (lack of communication infrastructure, high mobility...), we propose a framework that hosts multiple applications at once, offering at the same time management functions for context-awareness. Our framework is intended for mobile devices such as smartphones or in-car devices, which can range from stationary to highly mobile. We propose a service-oriented architecture able to compose applications out of services. This makes our framework flexible and allows for easier adaptation to context changes through the use of a Context Manager for all services (instead of having each service or application monitor the context). In this paper we present our proposition as well as some initial evaluations.

Keywords: Application Reliability, Mobile Environment, Context-Aware Transportation Services

1. INTRODUCTION

The recent advances in technology have lead to what we call ubiquitous computing - computing abilities and information access integrated in our day-to-day life, sometimes even unnoticed. If we consider a user with a smartphone, there is a very wide range of applications that she/he could benefit from. There is a great number of variables related to the technological resources used and the purpose of the applications, making this a vast domain.

The most important technological features we use with our smartphones are: GPS modules (tracking), wireless modules (communication), data transfer over the telephone network (3G), but also an great number of sensors (light, acceleration, noise, etc.). Applications can use any combination of these elements: GPS + data transfer for navigation or GPS + wireless communication for traffic event sharing in an ad-hoc network are just some of the possible examples. A user evolves through the environment and changes often the context of use: she/he can be in a crowded city center, on an isolated country road, on the highway, etc. We consider mainly the case of drivers, but the user can be a pedestrian or a passenger as well.

We see that there are many aspects to take into account: the used resources, the type and goal of the application, the context. The device owners should benefit from a continuous use of their mobile applications despite of the evolution of the environment. The same tools and principles should be used for all types of services and applications, which should be managed all at the same time. This would allow to monitor the context only once for all applications and notify the ones that need to adapt to changes. Adaptation and reconfiguration are important, especially in highly mobile environments.

In this paper we propose a service based framework for mobile devices such as smartphones or in-car devices, offering the possibility to run and manage applications, with a focus on transportation-oriented ones. We use the term transportation to refer to the movement of a

The present research work has been supported by the International Campus on Safety and Intermodality in Transportation, the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technology, the Ministry of Higher Education and Research, and the National Center for Scientific Research. The authors gratefully acknowledge the support of these institutions.

person from one place to another by different means (on foot, public transportation, car). Our goal is to simplify application assembly and reactivity according to transportation constraints, ensuring a continuous functioning in good conditions. For instance, our framework is suited for highly mobile environments, it allows to install and deploy services on the fly and thus adapt to context changes.

The first section describes the particular nature of the transportation context we are considering and the challenges it poses, a scenario of use and the state of the art. The second section exposes our vision and proposition for a Context-Aware Transportation Services Framework, detailing our approach for the context and the architecture. Before concluding, we present a use case and experimentation for our framework to evaluate the reconfiguration of applications due to context changes.

2. MOTIVATION

2.1 Purpose and circumstances of our work

In our work, we consider users who evolve through the environment using transportation-oriented applications on their portable devices. It is obvious that the context in which applications are executed on the device is very different, depending on the location, activity and preferences of the user and on other external influences. We must therefore define the specific context that we consider and the way it influences the applications.

We are interested in assuring a continuous availability for the used applications, no matter the conditions, by adapting or replacing parts of the application in respect with the context evolution. Our focus is on the transportation applications like routing, parking places, traffic events, etc. As the users can be either pedestrians or drivers, our approach takes into consideration the specific challenges posed by the high mobility. If our solution is good enough for the high mobility of VANETs, it should also be sufficient for lower mobility.

The challenges we face are related to the environment changes that occur often and unexpected: GPS and wireless connection loss, unstable communication network, etc. Moreover, the user might not be able to manage the necessary changes (like install a new part of an application) if she/he is driving, so some tasks should be done without user intervention. Therefore, an important part of our work is taking into account all the elements that constitute the context in which transportation-oriented applications can be executed. Most of the applications are influenced by the same elements (like speed or position), so it is more interesting to have a framework that manages these aspects rather than have each application do the same computations.

2.2 Scenario of use

To better understand our proposition, we will present an example of use in Figure 1. Let's consider a driver using a mobile device with navigation capabilities. The destination she/he has chosen is an indoor car park. It implies that the final goal is in fact to get a free parking space. The common way of doing this, once inside the car park, is to drive around, hunting for a free space, which is time consuming and frustrating. Here, the idea is to be able to benefit from different services available in the environment. They could be provided directly by the car park itself or even shared by other users. They should offer additional help and assistance to the user, easing her/his driving experience.

So after being directed to the entrance of the parking by using her/his classic GPS tracking system, the driver is then left alone. But by using a framework dealing with external service download and/or connection, she/he could be able to receive additional help. In our example, when arriving at the entrance, the user will start looking for services offering free space tracking assistance as well as services providing a way of localizing inside the parking area.

Here it appears that two "parking services" are available. One is offered by the car park whereas another user offers a second one. Our user can choose among them, depending on the hardware requirement they have. But the service offered by the pedestrian in our example is

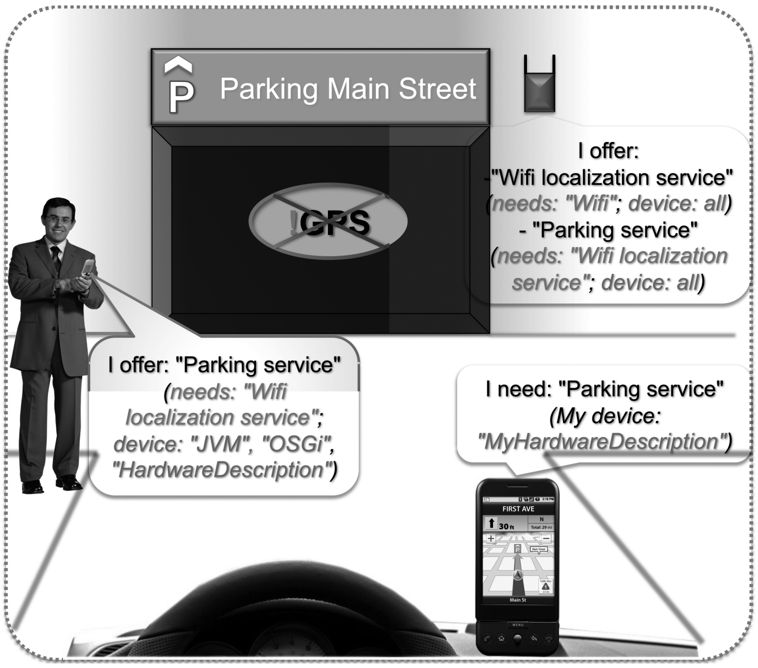

Parking Main Street
!GPS
I offer:
- "Wifi localization service" (needs: "Wifi"; device: all)
- "Parking service" (needs: "Wifi localization service"; device: all)
I offer: "Parking service" (needs: "Wifi localization service"; device: "JVM", "OSGi", "HardwareDescription")
I need: "Parking service" (My device: "MyHardwareDescription")

Figure. 1: Service exchange

adapted to his proper device, which limits the compatibility, whereas the car park could provide multiple service versions for different mobile platform. So our user is free to choose the more suitable service for her/his device.

Here the parking service also expresses that it requires a third party one, identified as a ”wifi location service”, to make up for the lack of GPS reception. This time only one provider shares it, the car park through its wireless access point.

As both needed services are available, our user is in position of downloading and taking advantage of them to be driven to an assigned free place. The services could stay for a while on the user mobile device, according to user properties, or her/his habits (leaving it on the mobile is useful, efficient and will avoid future downloading if the car park is frequently used). During this time, the services she/he has downloaded could also be shared to other users who may require them.

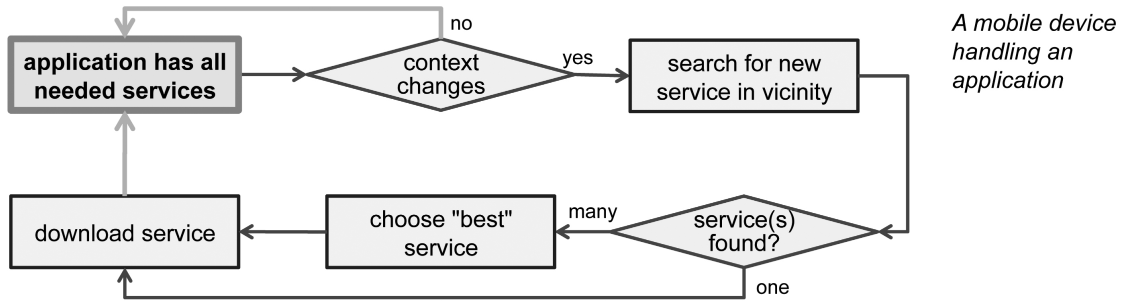

application has all needed services
context changes
no
yes
search for new service in vicinity
service(s) found?
many
one
choose "best" service
download service
A mobile device handling an application

Figure. 2: Device reaction to a context change requiring new services

We present in Figure 2 a simple schematic of what must happen when changes in the external conditions cause an application to stop working properly. The service(s) depending on the condition that changed (here: indoor/outdoor) can be replaced with one(s) that is(are) suited for the new condition. So, if the service is not already on the device, it needs to be searched for on the neighboring devices, downloaded and installed.

The questions that arise from this example are: *how to download a service from the other devices? how to choose which service to install if there is more than one available? how to connect the new service to the existing application?* For the time being, this is impossible. It is why we propose our Context-Aware Transportation Services Framework, that allows the devices to adapt to environment changes by reconfiguring or even replacing services, considering the ones most suited to the context.

2.3 Related works

Users with mobile devices can benefit from a large number of various applications in very different contexts. Most of the existing applications require the same functionalities of the device (GPS positioning, wireless communication) but are independent of one another. Each application manages the device resources on its own, but these could very well be managed only once for all applications if a common execution framework existed.

When considering the possible applications that can be used on mobile devices, we can see that there are many aspects that need to be considered. Some applications must absolutely use the telephone network for data transfer and can not work otherwise. Some of them, the Locations Based Services, need also a GPS module, which is now available in most smartphones. Other applications are meant to be used in "ad-hoc", offering services to a small community of users that are in the same place at the same time. For the highly mobile users, the "ad-hoc" applications can bring many advantages, like the possibility to have fast access to information (traffic accidents, intervention vehicles, traffic jams). Another important aspect is independence from infrastructure, which means having as much as possible "local" services. It is interesting to build services as small bricks that are easily downloadable.

Some projects are concerned with the transportation domain and especially with car-to-car and car-to-infrastructure communication, with the goal to improve user experience and safety. We can cite the European project PRE-DRIVE C2X [Pre 2010], concerned with the specification of of a European architecture for cooperative systems based on COMeSafety[1] architecture description.

A very interesting and important aspect in ubiquitous computing is understanding and using **context** as indicated in [Coutaz et al. 2005]. This is an important issue for our research too. [Dey and Abowd 1999] gives a rather general definition of context, basically including "everything" that could influence the behavior of the applications as context information. It is practically impossible to efficiently model "all" the context, so usually applications limit themselves to particular elements or situations. In our case too, we need to adapt the definition to our domain of research, and especially focus on the transportation related elements like the high mobility of the devices, the need for localization, the large number of different environments.

In [Kirsch-Pinheiro et al. 2006] the authors propose an object-oriented model of context for cooperative systems on the Web; their model is based on the elements important for the cooperation of users on the same task: localization, tool, time, community and process. In [Mukhtar et al. 2008; 2009] the authors present their work about dynamic service composition in a pervasive environment. They use CC/PP profiles [Kiss 2007] to specify the resources of the devices. Another paper, [Geiger et al. 2009], concentrates its focus on delivering messages to users within a certain context, depending on information like location, age, gender.

These are some examples of context models used in different research papers for the ubiquitous environment. While each of them presents interesting contributions, they do not cover all elements of context that interest us. High mobility and localization are often ignored or their influence is not as important as it should (from our point of view). It is why we chose to have our own context model.

Concerning the architecture, it is clear that it should be as modular as possible, with an easy

[1]http://www.comesafety.org

way of changing parts of an application. We can cite [Parra et al. 2009], who propose a Dynamic Software Product Line in order to create applications from the most suited components, taking the context into consideration. They describe a context-aware framework using sensors [Conan et al. 2007]. In their solution, applications have predefined configurations that are chosen with respect to the execution context. We would like to go a step further, by allowing to download and install new services while the application is still running. This would provide more flexibility and adaptability to the applications. To our knowledge, there is no literature concerning the download and installation of application components (services) "on the fly" for mobile devices involved in transportation.

3. DESCRIPTION OF THE CATS FRAMEWORK

Our proposition is about the creation of a framework that can host transportation applications composed of services. The applications, and thus the services also, must be adapted to users moving from one place to another via different transportation middles. Because of the often changes that occur during the execution of the applications, it is important to consider the context. In this section we explain in detail the approach for our Context-Aware Transportation Services (CATS) Framework and the benefits that it offers.

From the user point of view. It is useful to have applications designed for the transportation domain, such that they can offer the most suited assistance in any context. The applications should be easy to install and use, and the adaptation to context changes should occur without the user's intervention. The user should be able to download new applications when she/he desires them. So the user needs a simple interface with the possibility to launch and stop applications as well as download new ones. Reconfiguration and replacement of parts of the applications are not the concern of the user, although she/he can express preferences about how this should happen (especially for payed services like data transfer on the telephony network).

From the framework point of view. Applications must be adaptable so that they respect the expectations of the user and they continue working (as well as possible) under all circumstances. We propose a framework for the execution of applications built out of services, respecting the principles of the Service-Oriented Architecture (SOA). This allows for a flexible architecture with loosely-coupled bindings, where services can be replaced by equivalent ones or used by multiple applications.

3.1 Architecture of the CATS Framework

The framework we propose must host and manage the execution of multiple applications by providing mechanisms for:

- ✦ application composition through the binding of the right services as well as execution monitoring for the detection of non-functioning services
- ✦ service download (for a new application or as replacement of a non functioning service); the download can occur either in an ad-hoc network from a near-by device, or over the internet
- ✦ context management to ensure that the downloaded services are suited to the device and to accomplish service reconfiguration based on the changes in context

Our framework is a unified environment that provides non-functional services to assure the continuous and context-adapted execution of the applications. As can be seen in Figure 3, the framework provides services for managing the execution and the context for all launched applications, as well as a trading service to manage the service exchange. Further we will discuss in detail the elements involved in the framework.

3.2 The services

The word service can be used with different meanings, there are many types of services that are studied and used on mobile devices. A fist example are Web Services that require infrastruc-

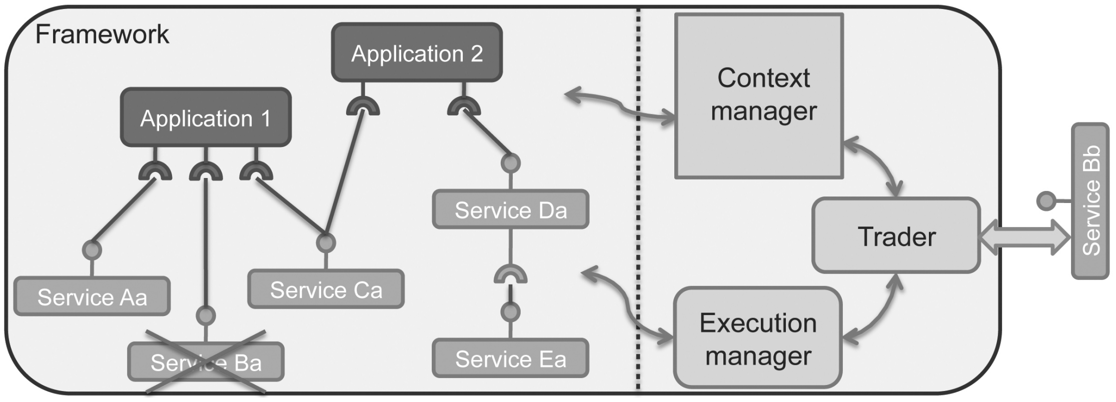


Figure. 3: The framework architecture

ture access; most of the applications proposed by the smartphones use some Web Service, like navigation that streams the map at the moment of use. It is not in this area that we want to bring contributions, as it is sufficiently developed. A second example of services are the ones offered by devices that are in proximity: printer, Bluetooth photo sharing, etc. For these kind of services there are well established protocols like Service Location Protocol (SLP)[2], Apple's Bonjour[3] protocol and UPnP.

Rather than trying to categorize the existing services, we will concentrate on explaining what we understand through services. We consider a Service-Oriented Architecture, where applications are built of services. Like in [Hall et al. 2010], a service is an interface representing the contract between the service providers and clients. The service providers are objects accessed via direct method invocation.

Our goal is to have as many applications as possible on the device, so the user can be as independent as possible with regard to infrastructure access. We focus on the services that can be executed locally and on how to download and install them "on the fly". Of course, we do not exclude the use of other types of services, like Web Services, but we don't focus on them.

As mentioned earlier, it should be as easy as possible for the user to handle the device and the applications installed. Moreover, the applications should react to context changes and adapt their behavior. One way to do this is to take advantage of the flexibility of SOA. A service is represented by an interface, but can have multiple implementations. In this way, there can be different implementations suited for some types of conditions and interchangeable depending on the context. For example, localization can be implemented using the GPS or the wireless module.

The device must handle download, installation and deployment with very little human intervention, so the services must contain all the necessary machine-readable information. We must describe the services using metadata, such that the device can extract different pieces of information depending on its needs. We present here the three parts of service description that we consider.

Functionality description. For each service we must answer the questions "*what?*", "*where?*" and "*how?*". When we need a functionality, we search for implementations of certain interfaces, namely for services doing *what* we want. The service must be executable *where* we need it, meaning on the operating system and the software platform of the target device. We may also specify *how* a service does it's job, by using or not a specific resource for example. This information is useful when searching to download a new service, that should be compatible with the device it will execute on, as well as with the user's needs.

[2]http://www.openslp.org/doc/rfc/rfc2608.txt
[3]http://developer.apple.com/networking/bonjour/specs.html

Deployment description.. Information such as service dependencies (from other services) and service version is necessary for the deployment, once a service has been successfully downloaded.

Context description.. The services can be dependent on the evolution of specific context elements. Our application framework must be able to notify the interested services when a context element has modified it's value so they will be able to reconfigure. In order to do this, the services must clearly indicate the elements of context which influence their behavior.

We must specify more accurate metadata that allows us to find the appropriate service and to use it under the best circumstances. The metadata must describe the elements presented before: the functionality of services, their deployment and their context dependencies. Parts of the description should be available separately from the code itself, as they are needed by the Trader. The different devices participating must be able to exchange information about the services in order to decide which service is most suited for the needs of the requesting device.

The descriptions are at the service level and not at application level, in order to maintain the code independency. For each application, there is one "main service" which doesn't provide functionality to any other service but depends on one or more services. The description of this "main service" should include a user-friendly description of the application itself.

3.3 The Context Manager

Our framework must take into account the changes that occur in the context and offer management functionalities to the applications running on it. We use the Context Manager for this purpose, as we will explain in the following.

[Desertot et al. 2009] begins to talk about context in the transportation domain. We have taken this as our starting point for the context model. We have identified the elements that influence the execution of the applications as well as the download of services in the particular situation of highly mobile users. Our classification of the context elements has been presented in previous work, [Desertot et al. 2010; Popovici 2010]. Here we show in Figure 4 a simplified representation of the context.

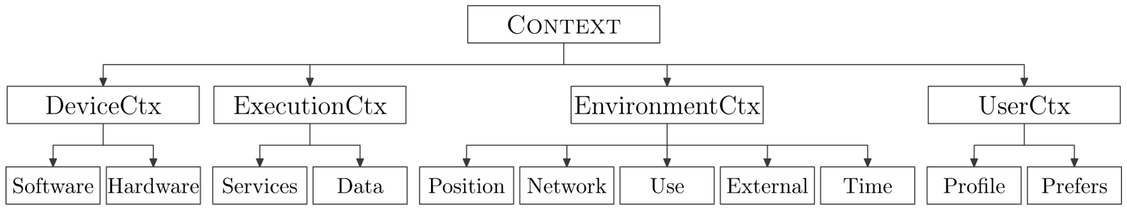


Figure. 4: The context tree

During the use of our framework, there are distinct situations in which the context information is needed and each situation makes use of different elements.

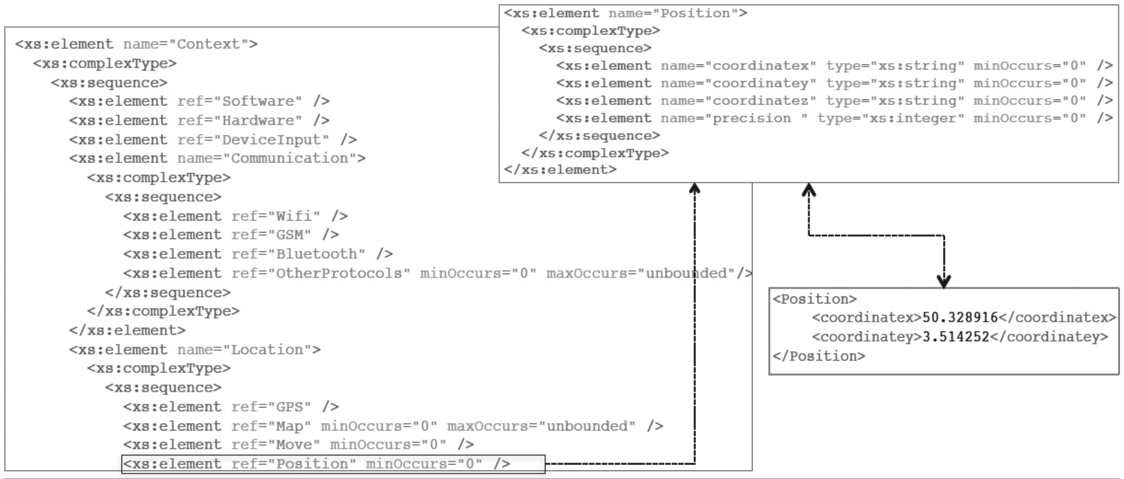
(1) During the trading process, context information must be exchanged between the devices in order to establish which service to be downloaded and what device to downloaded it from.
First of all, the requesting device must specify what service it is looking for and what constrains it has. This first step allows to filter out the services that could not be a solution. The constrains describe the DeviceCtx and ExecutionCtx from Figure 4 and must absolutely be met. For example, one of the constrains is the operating system for which the service is implemented. Another constraint can be a device functionality: if the GPS module doesn't work, then we search for a localization service that doesn't need GPS.
Second, after sending a request, a device must choose the best solution from the answers it gets. There are two choices to make: (*i*) which service implementation is the most adapted with respect to the device resources and (*ii*) which neighbor is most likely to be in reach for

a sufficient period of time for the download. Thus, the descriptions of the available services must be compared to the corresponding device context and the movement of the device and that of its neighbors must be considered to estimate the best peer from which to download.

(2) At service execution, context changes must be reflected in the behavior of the applications. For this, certain elements must be monitored, in particular those reflecting the movement of the device through the environment (EnvironmentCtx). Information like speed or type of user (pedestrian, driver) determine the values of parameters such that the provided services are adapted to each situation.

(3) The configuration of applications must also take into account the user preferences and profile.

Based on the classification presented above, the Context Manager takes "snapshots" of the context state and represents them in an XML file. An XML Schema (that follows the structure of the context tree in Figure 4) is used to validate the XML representation. It is not sure that all information will be available at any time, therefore some of the elements are optional in the XML Schema. In Figure 5 we present part of the XML Schema with the detail for the "Position" element as well as the extract of the XML context file with a possible instance of positioning data.



```
<xs:element name="Context">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Software" />
      <xs:element ref="Hardware" />
      <xs:element ref="DeviceInput" />
      <xs:element name="Communication">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="Wifi" />
            <xs:element ref="GSM" />
            <xs:element ref="Bluetooth" />
            <xs:element ref="OtherProtocols" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Location">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="GPS" />
            <xs:element ref="Map" minOccurs="0" maxOccurs="unbounded" />
            <xs:element ref="Move" minOccurs="0" />
            <xs:element ref="Position" minOccurs="0" />
```

```
<xs:element name="Position">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="coordinatex" type="xs:string" minOccurs="0" />
      <xs:element name="coordinatey" type="xs:string" minOccurs="0" />
      <xs:element name="coordinatez" type="xs:string" minOccurs="0" />
      <xs:element name="precision " type="xs:integer" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<Position>
     <coordinatex>50.328916</coordinatex>
     <coordinatey>3.514252</coordinatey>
</Position>
```

Figure. 5: XML Schema for context representation

For the time being, the Context Manager evaluates the state of the context only on demand. In the immediate future, our goal is to monitor the evolution of the context and evaluate the benefits of notifications for context changes as well as the overhead introduced by the monitoring service.

3.4 The Trader

The applications framework we propose must be able to acquire new services when these are necessary. There are multiple ways of functioning for the Trader, depending on whether there is infrastructure access or not, but also on who initiates the search for a new service: the user or the device. The service Trader handles both outgoing and incoming requests. If infrastructure access is available, it will be preferred, as a centralized registry is more likely to have a large number of services.

If the service trading takes place in an ad-hoc network, the devices participating are most likely moving through the environment. The requesting device must choose to download from a device that is close for a sufficiently long period of time. To assure that this will happen, we propose to use mobility vectors, like described in [Delot et al. 2008; 2010]. They are an indication of the direction and speed of a device. Combined with the distance, they allow to estimate the duration for which two devices are close enough for communication.

In the following we describe in detail the functioning of the Trader in different types of situations.

3.4.1 *Service research.* The device D searches for a given service S, based on its functionality description. This can occur when the framework needs a specific service in order to resolve the running applications. The framework will search for the service without the user's intervention.

(1) In the case infrastructure access is available, the following steps will take place:
   - ✦ Send a request for service S to the centralized service registry.
   - ✦ Get response from the registry with the download address.
   - ✦ Download service S.

(2) In the case without infrastructure access, the trading will be done in the ad-hoc network:
   - ✦ Broadcast request for service S in the ad-hoc network.
   - ✦ Wait for unicast response from all devices possessing the service: functionality description of the service, position and mobility vector of the device.
   - ✦ Select a neighbor based on the mobility vector of D and those of the neighbor devices. Considering the size of the service to download, the bandwidth, the distance and the mobility vectors, it can be estimated if the service could be fully downloaded from a certain device.
   - ✦ Unicast service download request to the selected device.
   - ✦ Download service S.

Services might describe functionalities they depend on (other services that they consume) but also functionalities that are "forbidden". In this way, if a functionality is not available and thus causing an application to stop working, we avoid downloading services that require this functionality. For example, let's consider the device we are using is in a building, and thus not have GPS signal anymore. We need a positioning service that doesn't make use of the device's GPS module. This can be expressed as a list of forbidden packages, so that only the services not depending on these packages can be chosen for download.

3.4.2 *Application research.* Device D searches for all available applications at user request. In this case, the user must choose which application to install and the "main service" will be downloaded. If all the dependencies are present in the framework, nothing else will be done. Otherwise, each missing dependency will be searched for, like described above in case (3.4.1)

(1) Infrastructure access is available.
   - ✦ Send request for service descriptions to the service registry.
   - ✦ Get response from the registry with groups of n service descriptions. Only the ones that are not present on the device will be presented to the user. The number n could be a user preference or a server setting.
   - ✦ Let the user selects the desired application(s).
   - ✦ For each selected application Ai, the device D sends a request to the service registry, like in case (1)

(2) No infrastructure access.
   - ✦ Broadcast request for all applications in the ad-hoc network.
   - ✦ Get unicast response from all devices with a list of their services, their position and mobility vector.
   - ✦ Present the services grouped by neighbor and ordered descending by the estimated time of connection to the neighboring device. The user chooses which services to download.
   - ✦ Send unicast service download requests to the selected devices beginning with one with the "best" connection. Only one request is sent at a time, so there is no parallel download.

3.4.3 *Service upload.* Each device must listen to service requests from neighboring devices and respond to them.

(1) Receive service information request.
   - ✦ If the service is present on the device, respond to the request and indicate the current position and the mobility vector
   - ✦ Otherwise do nothing.

(2) Receive service download request.
   - ✦ Send service to the requestor.

If an internet connection is available, services can easily be downloaded from a centralized registry. Yet, there are many situations where there is no infrastructure access: in-car devices, unavailable GSM (foreign country, etc). In these cases, an ad-hoc exchange of services is desired.

An advantage of a decentralized service trading is the possibility to access services specific to the current location in a more direct manner, without the overhead of research on a centralized server. Thus, a driver entering a new city could download a map with city attractions from any gas station or from other vehicles, or a bus passenger might get information like the time to reach each stop and the connecting routes at each stop from the device of the bus she/he is on.

3.5 The Execution Manager

The execution of the services in our framework must be monitored in order to detect failure. When a service stops working, the Execution Manager must react and take the necessary actions to correct the situation. The simple solution is to search for an equivalent service that is already on the device and start it. By equivalent, we mean a service providing the same interface or offering the same data. If there is no equivalent present locally, the Execution Manager must call the Trader to launch a search for the missing service on the neighboring devices.

We rely on the OSGi framework, which offers the possibility to manage the services throughout their lifecycle, as long as the services are installed on the framework. There are two types of problems that can occur regarding the services. The first is when and application is launched but it doesn't have all necessary services started. In this case, dependencies between the services can not be solved and some applications can not run. The missing service must be started (or downloaded, installed and started) in order to resolve the situation.

The second type of problem occurs when services don't function correctly because of an external cause. A simple example is the loss of GPS signal; the service is still running, but unable to provide the necessary information. In this case, the service must inform the Execution Manager of it's temporary inability to run, so that a replacement can be found. It is the Execution Managers job to stop the service so that another one can be bound. If there is no equivalent service on the device, the search for a replacement on neighboring devices must take into consideration the context element that caused the original service to stop working.

4. EXPERIMENTATION AND EVALUATION

The goal of this work is to allow applications to be executed in our framework and provide management functionality. We use Java and OSGi for the development, deployment and installation of services (and applications, which are built out of services). For each of the three parts of our framework (Context Manager, Trader and Execution Manager) we have built one or more OSGi bundles, each of them offering one or more services. The user applications are also built out of services exposed by the components and they are executed in the same environment as the management services.

To help managing service binding and automate as much as possible dynamic and adaptive bindings, we use the iPOJO [Escoffier et al. 2007] service-component model that is deployed on top of OSGi (and hosted by Apache). Application description and bindings can be expressed in different manners (xml metadata, annotation, configuration files), allowing the component container to take in charge instance creation and connections as soon as the requirements are met.

4.1 Framework execution

We present in Figure 6 a schematic of the functioning of our framework. First of all the user must start the runtime environment consisting of the OSGi framework with the iPOJO runtime and the management services (Trader, Context Manager and Execution Manager). Once this is done, the user can launch the desired applications that will execute on our framework. What we call *standard functioning* is when all services that are launched on the framework have their dependencies resolved and are giving results suited to the context of use.

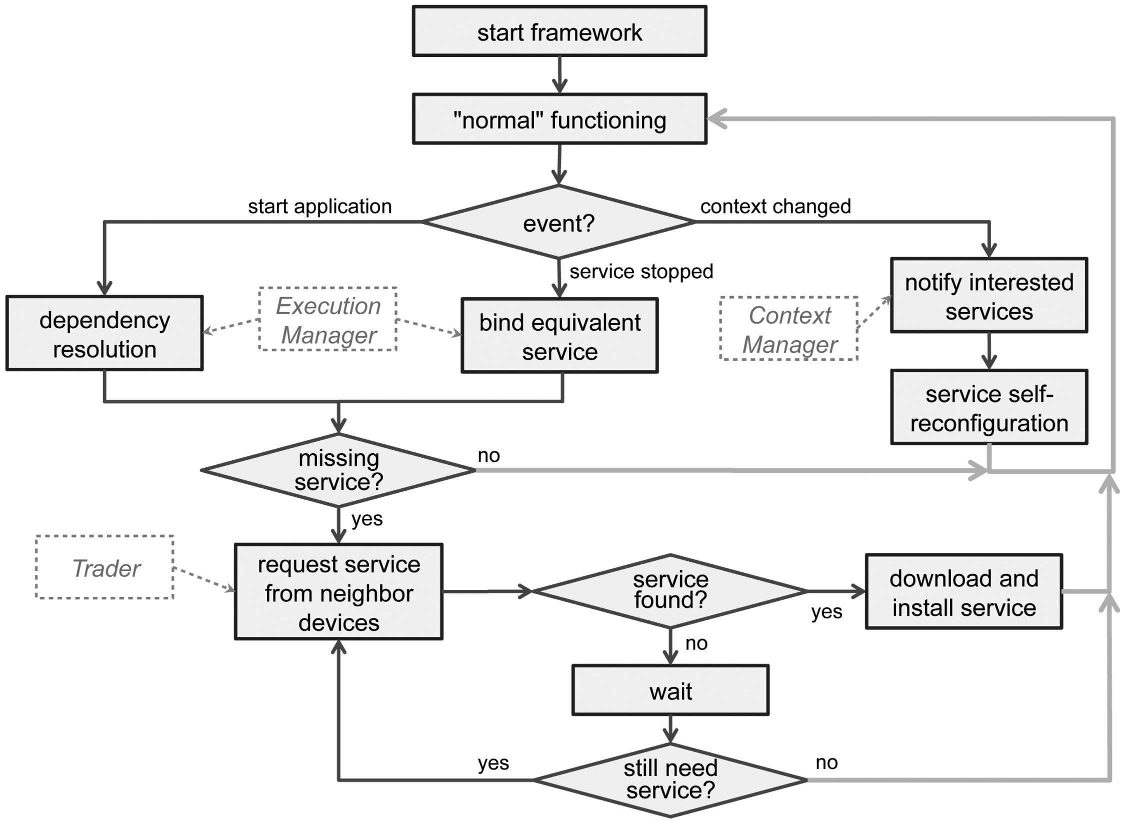


Figure. 6: Functioning of our framework

During *standard functioning*, three types of events can occur, requiring the framework to react to them. These events are the starting of a new application, the stopping of a service (that might cause an entire application to stop functioning) and a change in context. We will further explain what happens in each case.

Application started. For each application that is launched, the framework must resolve the dependencies between the services that compose it. Some of the services could be already started while others not. We are sure that at least one implementation of each composing service is available on the device, as this is taken care of at application installation. Still it can happen that the implementation of one of the services is not working in a particular context. In this case, the service is marked as missing and an "equivalent service" must be found.

As services are represented through their interfaces, two services are "equivalent" if they offer the same interface. This means that they will provide the same result, even if the implementations can vary. It is the Trader that will take care of requesting the service to neighboring devices. If the service is found, it is downloaded and installed in order to complete the starting of the application. Otherwise, the Trader waits a certain period of time. Two context changes can happen which will resolve the problem. First, the condition that was previously stopping the original service to work could disappear, making it useless to search for an equivalent service. Second, the neighbors could change, allowing the Trader to find the service and download it. When all dependencies are resolved, the application is started and the framework is in "standard" functioning.

Service stopped. Due to a context change, some services might not have the necessary conditions to function properly any more. Some examples that can cause services to fail are: loss of GPS, loss of the 3G network, loss of the wireless connection. The services are stopped, making the applications that depend on them stop as well. Our framework must replace the non-functional service with an equivalent one which will not be affected by the condition that stopped the original service. The search for the equivalent service is done by the Trader, as described before.

Context changed. A context change can be an event that triggers adaptations. Services can have parameters that depend on the value of some context elements. In this way, the behavior of a service can change and adapt to the context it has to run in by adapting the corresponding parameters. Each service should subscribe to the Context Manager for the context elements it is interested in. The Context Manager monitors the evolution of the context and notifies the changes to the interested services.

To better understand this, let's take an example. Consider a traffic application that alerts drivers of events on the road ahead of them. Depending on the road profile and the speed, the user should be alerted for some of the events and not for other ones. On a highway, accidents should be alerted several kilometers ahead, while in the city this would be useless. This is why the services deciding whether an event is relevant or not must be notified of changes in speed and road profile.

4.2 Stress situations and fault tolerance

A service-based application can encounter problems due to the malfunctioning of one or more services. We have discussed in the previous section what happens when a service stops working properly: an "equivalent service" is searched for. But it is possible that no satisfying service is found. The search for a service will continue in background until one of the following conditions is met: the service is found (the neighbors have changed); the context changed such that there is no more need for the service; the application has stopped.

If the "equivalent service" is not found after a certain time, the applications should start working in a degraded mode. They should use some "basic services" that depend as little as possible from outside conditions. Consider for example the traffic application that notifies important events. A "normal" functioning will have a map, with the important events displayed on it. If there is no 3G connection, the map can not be downloaded, but the GPS is still working. In this case, a solution is to display in graphical way the events on a blank map, giving the driver some basic information about the distance and direction to the events. If there is no GPS signal, the application could work just by notifying accidents. If the message is directly from the event source or just after one hop, then the accident is potentially very close and it could be worth to inform the driver. Of course, it can also happen that an application is unable to function, not even in degraded mode. In this case, it must display a message to the user to inform her/him that it is waiting for a service.

Another important issue when changing services concerns the fault tolerance. The services should not influence the state of the applications. Usually the services return independent results on demand, meaning that they do not depend from previous states. In this way, the application (the "main service") is responsible for its state and for managing the lack of response from a service.

Services can be replaced in two cases: first, when one is unable to work due to outside conditions, or second, when the context has changed, making another implementation more suitable. In the second case, our framework allows to start the new service before stopping the one that must be replaced, to assure that the new one is working properly and also to minimize the interruption of the service.

An issue which must be considered in the future is the failure of our framework. For the time being, this problem has not been studied. The applications should be able to save their state to allow for recovery after a crash. The issue will be treated in future work.

4.3 Evaluation of the CATS Framework

We have started testing our framework in order to evaluate its behavior. As said earlier, we rely on the OSGi framework with the iPOJO runtime. For the implementation we use Felix 3.0.3, which is a certified platform[4], conforming to the latest specification release of OSGi, Release 4 Version 4.2 from 2010.

As material, we use HTC Hero smartphones with the following characteristics:

- Android 2.2 operating system
- The Felix OSGi 3.0.3 implementation
- The iPOJO runtime

The phone used for testing has only the operating system installed, with the default applications. We also use the emulator for Android 2.2 on a Mac Book with 4GB memory DDR3 and Intel core 2 duo at 2,4 GHz processor.

Experiment description. The experiments that were realized are meant to determine in the first place the usability of the framework and it's performances. Our goal is to be able to replace non-functioning services with equivalent ones. An "equivalent service" has the same interface as the original service, provides the same result, but has a different implementation. Some of the differences can be between a lightweight service with bad precision vs. a resource consuming service with very good precision or services which need different types of resources to provide the result (for example positioning with the GPS module or with the wireless one).

We used a scenario based on the example presented earlier in Figure 1: during the execution of an application, one of the services stops working and must be replaced if possible. There are two possibilities, the fist is that an equivalent service is already present on the device and the second is that a service must be downloaded. In the example we presented, the positioning service had to be replaces with one using the wireless connection instead of the GPS. Most probably the server of the parking is able to provide the suited positioning service. For the time being, we have not yet implemented the Trader, but we test the framework without this part. All services are thus local to the testing device.

Results. We have performed the following measurements for our framework:

(1) Time to start the framework: $\sim$ 15 seconds on the phone.
We measure the time from the moment when the user selects the framework for starting until all the functionalities of the framework are loaded (all bundles are started) and it is ready to be used. This test has been realized with the standard OSGi specification, but we intend to do a new test when the OSGi ME framework is released, as it should be optimized for embedded systems.

[4]http://www.osgi.org/Specifications/Certified

(2) Time to switch between two equivalent services, when both are started and ready to use: unmeasurable.
Let's consider a bundle A that requires a service S and two bundles $B1$ and $B2$ providing S are started in the OSGi framework. Both $B1$ et $B2$ are ready to use at any time, but only one is used at each moment. Let's suppose that bundle A is bound to bundle $B1$, but this bundle stops unexpectedly. In this case, the iPOJO runtime resolves the binding to bundle $B2$ practically instantaneous, so the time period while bundle A can not use service S is negligible.

(3) Average time to install and start a service that is local to the device: $\sim 0.6 - 0.8$ seconds on phone and emulator.
This test reflects a part of the situation presented earlier in our example of a driver entering an underground parking and thus losing GPS localization. In order to make an application work again after the loss of a service, the device must download, install and start a service equivalent to the lost one. For now, we have only tested with local services, that are present on the device but not started. We measured the time between the "start" command that begins the instantiation of the bundle and the end of the instantiation process, when the bundle has all dependencies solved and is ready to use. We present our results in Figure 7. When the Trader implementation will be finished, we will complete these results to estimate the total time necessary to replace a service when no equivalent is available on the device.

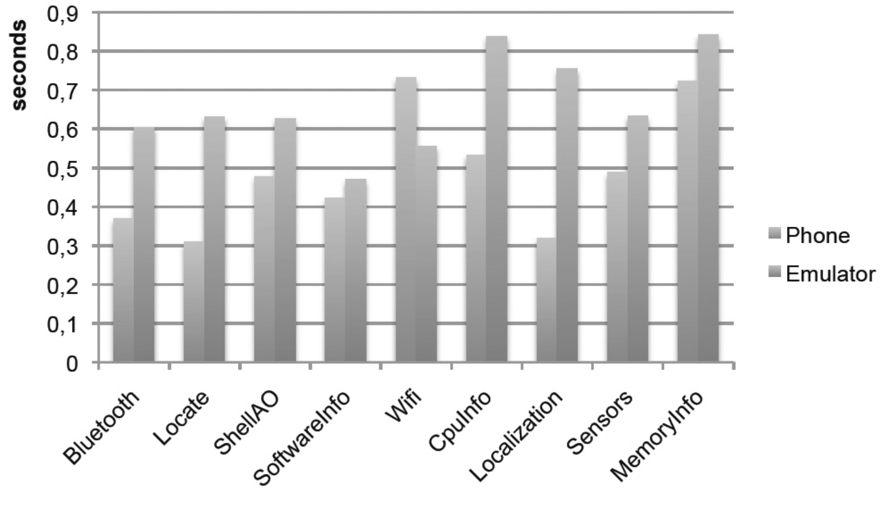


Figure. 7: Service install & start time

The results presented in Figure 7 are the starting times for some of our test bundles, none of which have dependencies from other bundles. The size of the .jar archives vary from 4 to 12kB. The test bundles are:

- ✦ Bluetooth: is the bundle allowing access to the Bluetooth module of the device; it has no dependencies from other bundles.
- ✦ Locate: is a positioning bundle relying only on the telephone network.
- ✦ ShellAO: is a test bundle offering a shell interface on the phone.
- ✦ SoftwareInfo: is the bundle that reads the software information of the device.
- ✦ Wifi: is the bundle allowing access to the wireless module of the device.
- ✦ CpuInfo: is the bundle that reads information about the state of the processor.
- ✦ Localization: is a positioning bundle that can use either the telephone network or GPS.

- ✦ Sensors: is a bundle allowing to handle the sensors of the device (accelerometer, etc.).
- ✦ MemoryInfo: is the bundle that reads information about the state of the resources (memory, etc.).

Each of the bundles has been stopped and started seven times. The results presented are the average times to start each bundle on the phone and on the emulator. As we can see, on the phone the times vary from 0.31 to 0.73 seconds to start a bundle. On the emulator, the times are slightly larger, from 0.47 to 0.83 seconds. The times vary slightly depending on the size of the bundles and on the phone functionalities that need to be accessed.

These preliminary results are satisfying, the time to start a new bundle being sufficiently small to be acceptable for the user. Our future efforts will concentrate on finalizing the implementation and evaluation of the Trader, which will download services from near-by devices. Once the trader will be complete, we will be able to evaluate the total time needed to search for, download and start a service.

5. CONCLUSIONS

In this paper, we propose an embedded application framework called CATS. This framework is dedicated to mobile devices, such as smartphones and in-car devices, offering an execution environment for transportation-oriented applications. By transportation we mean any kind of movement to get from one place to another, where a hand-held device can help the user (who can be a pedestrian, a driver or a passenger) by indicating the route and interest-points near-by or by notifying important events. The applications conform to the SOA principles and consist of loosely-coupled services. This allows for a flexible architecture, easily configurable and re-configurable if needed.

We consider the context of execution for our framework, in order to adapt the functioning to each situation. Our goal is to assure continuity of the execution for the applications despite unfavorable conditions of context by replacing or adapting services. We propose to use several implementations for some services (the ones that can stop functioning in certain contexts). If an application stops working because of a non-functioning service, then an equivalent service will be bound to the application. If no such service is available on the device, it must be searched for and downloaded from neighboring devices.

In this article, we present some preliminary results for service replacement when the context changes determine a service to stop working. For the time being, we measure the time to start a local service, which happens sufficiently fast, as we could see from our measurements. In the near future we must implement and test the service Trader to evaluate the total time necessary from service discovery until having the service operational.

In the next period of time we must concentrate our efforts on the Trader. First of all, we must work on the communication between the devices during the trading process. When a device requests a service, the near-by devices respond with the description of the implementation that they possess. The service description must contain all necessary information so that the requesting device can choose the most suited implementation based on the current context. We will focus on the service description and decision taking mechanism that allow a device to choose which service to download. Obviously, testing the proposed solutions will allow us to evaluate the functioning of our framework.

REFERENCES

2008-2010. Pre-drive c2x project, "preparation for driving implementation and evaluation of car-2-x communication technology", available at: http://www.pre-drive-c2x.eu/.

Conan, D., Rouvoy, R., and Seinturier, L. 2007. Scalable processing of context information with cosmos. In *Proceedings of the 7th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*. DAIS'07. Springer-Verlag, Berlin, Heidelberg, 210–224.

Coutaz, J., Crowley, J. L., Dobson, S., and Garlan, D. 2005. Context is key. *Commun. ACM 48,* 3, 49–53.

Delot, T., Cenerario, N., and Ilarri, S. 2008. Estimating the relevance of information in inter-vehicle ad hoc networks. In *Proceedings of the 2008 Ninth International Conference on Mobile Data Management Workshops*. IEEE Computer Society, Washington, DC, USA, 151–158.

Delot, T., Cenerario, N., and Ilarri, S. 2010. Vehicular event sharing with a mobile peer-to-peer architecture. *Transportation Research Part C: Emerging Technologies 18,* 4, 584 – 598.

Desertot, M., Lecomte, S., and Delot, T. 2009. A dynamic service-oriented framework for the transportation domain. In *Intelligent Transport Systems Telecommunications*. Lille, France.

Desertot, M., Lecomte, S., Popovici, D., Thilliez, M., and Delot, T. 2010. A context aware framework for services management in the transportation domain. In *2010 10th Annual International Conference on New Technologies of Distributed Systems*. Tozeur, Tunisia, 157–164.

Dey, A. K. and Abowd, G. D. 1999. Towards a better understanding of context and context-awareness. In *In HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. Springer-Verlag, 304–307.

Escoffier, C., Hall, R. S., and Lalanda, P. 2007. ipojo an extensible service-oriented component framework. In *IEEE International Conference on Service Computing (SCC'07)*. Salt Lake City, USA, 474 – 481.

Geiger, L., Schertle, R., Dürr, F., and Rothermel, K. 2009. Temporal addressing for mobile context-aware communication. In *Mobiqutous 2009*.

Hall, R. S., Pauls, K., McCulloch, S., and Savage, D. 2010. *Osgi in Action: Creating Modular Applications in Java*. Manning Publications.

Kirsch-Pinheiro, M., Villanova-Oliver, M., Gensel, J., and Martin, H. 2006. A personalized and context-aware adaptation process for web-based groupware systems. In *UMICS*.

Kiss, C. 2007. Composite capability/preference profiles (cc/pp): Structure and vocabularies 2.0.

Mukhtar, H., Belaid, D., and Bernard, G. 2008. A policy-based approach for resource specfication in small devices. In *UBICOMM '08: Proceedings of the 2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*. IEEE Computer Society, Washington, DC, USA, 239–244.

Mukhtar, H., Belaid, D., and Bernard, G. 2009. User preferences-based automatic device selection for multimedia user tasks in pervasive environments. In *Proceedings of the 2009 Fifth International Conference on Networking and Services*. IEEE Computer Society, Washington, DC, USA, 43–48.

Parra, C., Blanc, X., and Duchien, L. 2009. Context awareness for dynamic service-oriented product lines. In *13th International Software Product Line Conference SPLC 2009*, J. McGregor and D. Muthig, Eds. Vol. 1. 131–140.

Popovici, D. 2010. Context elements for transportation services. *IEEE International Conference on Mobile Data Management, PhD Forum*, 287–288.

Dana Popovici is a PhD student at the University of Valenciennes since 2009. She obtained her Engineering degree in Computer Science at the University ”Politehnica” of Bucarest, Romania in 2008. She then obtained a Research Master from the University of Valenciennes, France in 2009. Her research concerns distributed information systems. She’s interested in context-aware mobile applications, service-based applications and service trading..

Mikael Desertot is assistant professor at the University of Valenciennes since 2008. He defended his PhD in computer science in 2007 after working 3 years in collaboration with Bull SA on dynamic application containers for Java application servers. He then spent one year and a half working in Canada for Oracle Corp, on EJB container implementation. Currently, his research area still focuses on dynamic software architecture and reconfiguration, relying on service oriented architectures, and targeting transportation services and vehicle-to-vehicle communication.

Sylvain Lecomte is full Professor at LAMIH-University of Valenciennes. He has obtained a HDR in Computer Science at the University of Valenciennes (Title: Conception and adaptation of technical services dedicated to ambient computing , 2005) and a PhD in Computer Science at the University of Lille 1 (Title: COST-STIC : Smart Card embedded into transactional services and transactional services embedded into smart Card, 1998).// Relevant Work Experience: Strong background in Context aware computing, Mobile services and Component model

Nicolas Peon obtained his M.Sc in Computer Science from the Universit de Valenciennes, France in 2009. Since 2009 he is an engineer at the LAMIH Laboratory at the Universit de Valenciennes. He is interested in context aware applications and mobile development.