Enhancing the CATS framework by providing Asynchronous deployment for mobile application

M. Desertot[1], C. Gransart[2], and S. Lecomte[1 2]
mikael.desertot@univ-valenciennes.fr
christophe.gransart@ifsttar.fr slecomte@univ-valenciennes.fr

[1] UVHC, LAMIH CNRS, UMR 8201, F-59313 Valenciennes
University Lille North of France, F-59000 Lille, France
[2] IFSTTAR LEOST
20 rue Elisée Reclus BP 70317 F-59666 Villeneuve d'Ascq cedex, France

Abstract. The work presented in this paper focuses on the use of the CATS framework (a framework dedicated to the adaptation and the deployment of context aware services in the transportation context) on applications designed to assist the user in transportation activities (e.g., driving assistant, visiting a city, finding a parking place etc.). We present a solution to deploy and use these services when users do not have a connection to WAN by using asynchronous solutions based on smart cities infrastructures. A prototype has been developed and evaluated at the end of this paper.

1 Introduction

Nowadays there are many applications that users can benefit from on their handheld devices. 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. In this article, we focus on the transportation domain. Contrary to other application fields like finance or commerce, mobile technologies did not have a big impact on transportation until lately. The only successful application of these technologies that we can mention in the transportation domain concerns navigation devices. Nevertheless, the situation has now started to change. New applications like Waze (http://world.waze.com) have appeared on the market. Such applications rely on a community of drivers who share information about traffic congestions or accidents. For example, Waze centralizes data through the telephony network (3G), while other approaches require direct interaction between mobile nodes.

To simplify application assembly and reactivity according to transportation constraints (lack of communication infrastructure, high mobility...), several researches propose frameworks that hosts multiple applications at once, offering at the same time management functions for context-awareness and the deployment of service when the user or the system need it.

When considering the possible applications that can be used on mobile devices, we can see that many aspects have to be considered. For instance, some

applications must use the telephony network to transfer data and can not work otherwise. Some of them, Locations Based Services, also need a GPS module, which is now embedded in most smartphones. Other applications are meant to be used in an ad-hoc mode, providing services to a small community of users who are at the same place at the same time. For the highly mobile users, the ad- hoc applications can bring many advantages, such as the ability to have a fast access to relevant information (e.g., traffic accidents, intervention vehicles, traffic jams). Another important aspect is the independence to the infrastructure, which suggests the interest of having services for "local" ad-hoc networks. The research for developing reliable ad-hoc networks for highly mobile users is ongoing. We mention [1] who propose a Context Management System using the NEMO (NEtworkMObility) protocol, which addresses the issue of a mobile network also capable of attaching to the Internet.

Among the possible mobile applications for handheld devices, our work focuses on those designed for the transportation domain. By transportation we mean the movement of people from one place to another. To get to her/his destination, a person can either walk, drive or take public transportation modes (or any combination of these). Common features are mostly related to the trip - the environment and the neighboring devices change as the person advances. Most of the transportation applications need positioning information at all time (preferably GPS coordinates). Communication is also an important element, allowing applications to contact either an infrastructure or surrounding devices on the ad-hoc network. Moreover, the type of environment affects the behavior of the applications (e.g., use in indoor/outdoor environments, in urban areas or on highways). The differences between applications (or between the different behaviors of the same application) are related to the considered mean of transportation. A pedestrian travels at low speed and will be able to look often at his/her device and make some adjustments if necessary. Passengers can handle their devices, but move at a higher speed, which can sometimes lead to connection failures. Drivers face the same connection problem but also need to be notified with traffic information and important events, without being disturbed since they are driving.

In our previous work we have introduced CATS (Context Aware Transportation Services), our framework for context-aware applications for the transportation domain. The goal of this framework is to provide an execution environment for service-based applications as well as management functionalities for the deployment and the adaptation to context changes. Some preliminary evaluations of CATS have been presented in [2], showing that the framework is light enough for mobile devices such as smartphones. With CATS, using the ad-hoc network we can detect services that offer a functionality specific to the area we are in, and we can benefit from them by installing them on our device. In [3] we evaluated the time necessary for service download in different situations and find results coherent with our need: services can be downloaded fast enough from one-hop neighbors.

In the same time, we are used to be connected to the network every time we need it. However, using data services can be impossible in some ways : when we travel in another country, or when we are passengers in a train (where using data network is difficult, due to the number of passengers arriving at one time in a cell). This paper proposes to enrich the CATS framework by allowing to invoke services in an asynchronous way.

The first section shortly describes the CATS Framework. The second section describes the need of asynchronous deployment in 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 third section exposes a solution available in distributed computing to take into account asynchronous invocation and deployment. Next, we present an evolution of our platform, detailing our approach for the integration of asynchronous actions and the architecture. Before concluding, we present a use case, experimentation, and evaluation for our framework.

2 The CATS Framework

In this section we describe shortly the CATS Framework, which has been presented in our previous work [2], and finally the need to extend this framework to allow asynchronous invocation.

2.1 The Framework

To provide adaptation to the changes that occur during the execution of an application, there are two things necessary: the services must specify which behavior is specific to which context, and the context situations must be identified. This is where our framework comes in, by providing framework-specific services to help management of the applications by monitoring the context. We consider two kinds of adaptation. First, services can determine relationships between some parameters and values of certain context elements. The second way of adapting an application to context is to change an entire service rather than having a service with much code, trying to cover several situations. Context information is used: during execution, when one or more context elements change enough to justify a change in the behavior of an application; at the download of a new service or application, to ensure compatibility with the device.

Applications are composed of multiple functionalities, which can be divided in independent modules. First, we can identify the functionalities that are common to most applications, like for example positioning. It is always more interesting to have a single piece of code handling the localization, rather than having a lot of applications implementing similar code. Second, each part of an application providing a certain functionality can be implemented in more than one way. So, for each computation we can choose the most appropriate way to get the result.

Our framework (Figure 1) provides an unified environment with non-functional services that assure the continuous and context-adapted execution of the applications. On one hand, the CATS Framework allows the execution of three specific

services (Context Manager, Execution Manager and Trader) which function continuously as long as the framework is started. On the other hand, the framework is the environment where the applications are executed.

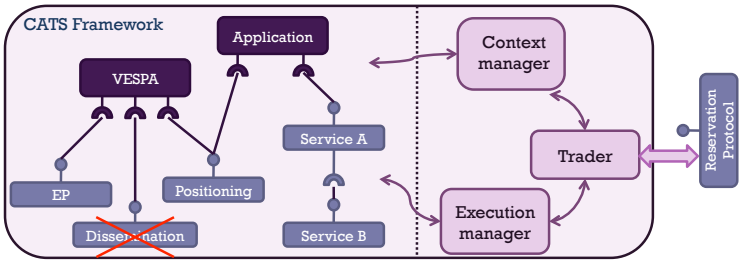


Fig. 1. The CATS Framework architecture

Each specific service has a role identified as follow :

- The trader. The application framework must be able to acquire new services when these are necessary. The trader sends a message to the ad hoc network to demand the needed service. Neighboring devices able to provide the service answer to the requesting device, which then chooses from whom to download. 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.
- The execution manager. The execution of the services in the 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. 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. Another type of problem occurs when services do not 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 correct information. The execution manager stops the service so that another one can be bound.
- The context manager. Based on the classification we have for the context, the CATS framework service called context manager takes "snapshots" of the state of the environment and represents them in an XML file. An XML Schema is used to validate the XML representation. The context manager can evaluate the state of the context on demand, but also on a continuous basis, when certain elements need to be monitored.

We use Java and OSGi (a Java dynamic service-oriented framework) for the development, deployment and installation of services (and applications, which are built out of services). For the three parts of our framework (Context Manager,

Trader and Execution Manager) we have built one or more OSGi bundles. 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 [4] service-component model (hosted by Apache) that is deployed on top of OSGi.

2.2 Needs of asynchronous invocation

Our CATS framework is dedicated to transportation services, because we wish to provide solutions that take into consideration the specific constraints of this domain. Our solution is based on a framework for applications with similar characteristics, designed to accompany users throughout their evolutions in different environments. To respect the constraints of the transportation domain, our solution must react sufficiently fast to the changes that occur. Moreover, services must be light enough to be easily downloaded and installed "on the fly". A local service is not subject to network connection loss.

However, in specific environments, services are still too heavy to be downloaded "on the fly" (services exchange between cars on a highway, or communication between train and car). More over, it could be impossible to use 3G connection (when the user is using roaming). To overtake this drawback we propose in this paper to enrich the CATS framework to allow asynchronous invocations to services.

More over, in a near future, smart cities will propose, through a lot of equipment, connections to services by using an ad-hoc link. Indeed, cities are increasingly integrating communication capabilities, with higher availability an quality, as well as social infrastructure. It is against this background that a modern way of interacting with the city environment has emerged, highlighting the importance of Information and Communication Technologies (ICTs). In this context, smart cities will rely on wireless networks to manage the environment, and people in the city. By multiplying communication node disseminated in the environment, users will benefit from more regular access to the network. From this fact, a user will be capable of connecting frequently and take advantage of this functionality. These connections could act as a mailbox to get the results of asynchronous invocations [5–7].

3 Using DDS in the CATS framework

We presented above the CATS architecture, showing its main components: the context and the execution manager as well as the trader. This last has been first designed to only deal with synchronous requests through a classical or ad-hoc network towards an internet component farm or a user's neighbor. But because of the reasons listed in the previous sections the trader has evolved to a novel architecture to tackle asynchronous issues.

3.1 The DDS framework

The DDS (Data Distribution Service) is a specification of a publish/subscribe middleware taking into account QoS properties[8]. The first specification has been defined by the OMG during 2004. DDS can be used independently from CORBA. This new service is suitable for a new class of applications that require real-time Quality of service. It provides an efficient publish/subscribe system. Information consumers subscribe to information of interest. Information producers publish information. DDS matches and routes relevant information to interested subscribers. DDS furnishes QoS suitable for real-time systems: deadline, levels of reliability, latency, re-source usage, time-based filter. Different programming styles are allowed based on listener or wait-based data access.

3.2 A new architecture for asynchronous deployment

As in CATS framework, every pieces of software are modular and are build together thanks to a service architecture, introducing another messaging protocol only requires the addition of a component wrapping the communication with the DDS server. Thus, the trader can switch from one protocol to another according to the context elements. For now, context information like destination, network availability or battery level are used to decide the proper communication mode. In order to bring context reactivity inside the trader, CATS capabilities have permit to smoothly add the required notifications. Indeed, by just adding an XML description of the context elements to monitor, CATS injects inside the component the appropriate code. The following scenario illustrates CATS behavior when facing asynchronous communications needs.

3.3 Scenario for context-dependent asynchronous requests

On a standard use, our mobile application adapts itself to the environment requesting components or data in a synchronous way. While a context notification about a travel of the user is raised, requests related to its future location are switched to an asynchronous mode (DDS mode) by the trader. In this way, we first minimize the cost of the communication with our neighbors, unlikely to be able to have what we want. Secondly we can make sure that the reception of the response (a component service or some data) will be done at the right time and/or the right place. For instance the most obvious situation is when the trader is notified that the user is closed from its destination, this can trigger a check of the messages posted for him (a service could have been personalized in a specific way for this user). According to battery level, data or software pieces can be downloaded but may be kept aside if the level is low. Finally if the user go through an area with free and high rate network communication capabilities, subscription to this event by the trader could initiate an anticipated download. CATS framework takes in charge those notifications as soon as thresholds and context elements are described with XML meta-data. Afterwards the trader will manage components or data retrieving, and delegate their instantiation/installation to CATS' execution manager.

4 Prototype and evaluation

This section presents experiments and results carried out in the context of this work. It relies, on the client side, on an application built thanks to the CATS framework presented above and implemented in previous works. The goal is to illustrate the feasibility of our proposition by using industrial standard frameworks in a prototypal architecture close to a real environment.

4.1 Prototype description

Our environments is composed of three layers : The mobile client side (i.e. a smartphone) which will send an asynchronous request for obtaining a software service/component or any kind of data. It is capable to dynamically install the software grain it will received. The server side, responsible for interfacing both clients and external component repositories with the DDS environment. Component repositories, offering the capability to potentially download any kind of known component. We use OpenSplice, the PrismTech's DDS implementation [9] as a keystone. It offers the reliable asynchronous framework we require to achieve our goal. But if it could have been smoothly included inside any smartphone application, the architecture would have been lighter as business component managing asynchronism capabilities would have been embedded inside smartphones. But it is not currently available for any smartphone OS like Android or iOS, mainly because its architecture is not designed for now for such constraint environments. It is the reason why we introduce some servers interfacing our mobile client with the DDS message bus. Those servers have in charge collecting requests from clients and delivering messages when a client is connected and is looking forward to unstack its messages.

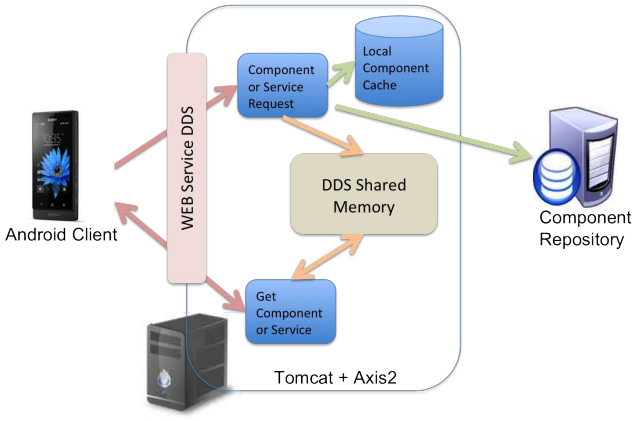


Fig. 2. Overview of the architecture

Connection and communication between clients and servers are performed using a Web Services architecture. Clients build and send a SOAP compliant message (those devices always offers on-the-shelf API to easily generate such messages) to the web services hosted by servers. In our prototype, two web services interfaces are available : one offering the possibility for a client to request a component, service or data and another one allowing to consult message box.

When a query is received by the server through a Web Service, four eventualities can occur :

- The component/service/information is available locally (managing a cache minimizes the access time to a component). The response is packaged and posted to the client via DDS.
- Nothing is available locally. Query is sent to remote component or data repositories. Response is cached and posted to the client via DDS.
- The component has to be adapted. Depending of the kind of customization, such component can be cached or not. Afterward the message is posted to DDS.
- If the request is not understandable or if there isn't any component available a post referencing the issue is made. The interest is twofold, ensuring a response to the client and allowing him to eventually react to the problem.

To optimize the bandwidth, the client can specify a component name to prioritize a particular message containing the information he needs as fast as possible. Here the CATS framework on client side managing context elements is very helpful to determine what is the preferred behavior. A client can then dynamically install the downloaded component, or process the received data. If an error is raised, the request can be re-sent, identicilly or modified if needed. The behavior is in the smartphone application hands at this point.

In the architecture presented in figure 2, only one server is shown but obviously we imagine a constellation of such servers, transparently exchanging information and messages in a reliable way thanks to DDS.

4.2 Experimentation

Lets first describe devices used at server side. To implement this architecture, we used a Mac Book Pro with a dual-core processor and 8 Gb of memory to host the server. Server is run using Tomcat [3] coupled with AXIS 2. The community edition of OpenSplice offers the DDS instance. Its runs on a Linux ubuntu operating system on the Mac Book Pro. On client side we use different kind of mobile devices, running Android in different versions (2.3, 3.1 and 4.0.4). Multiplying the number of devices shows that performances depends for a certain part on the device used but also show that in most of the cases, any device is capable of offering enough power to make the experiment possible for a user.

From this architecture, we evaluate different measures to illustrate request and message un-stacking namely A : The time spent by client to send a request.

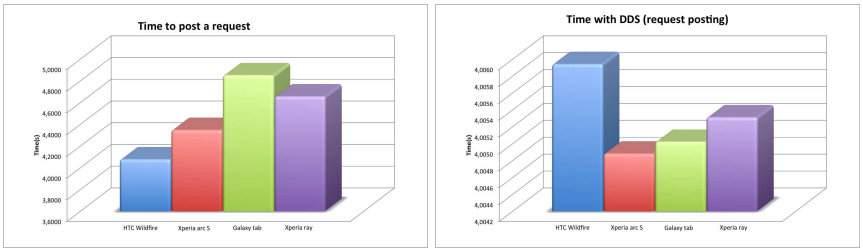
[3] http://tomcat.apache.org/

It includes connecting the client to the web service and receiving an acquittal from the server. B : The time, starting from the client request, for the server to put down the message to the DDS queue if the response is cached locally. C : Same as B but response downloaded from a remote component repository. D : The time spent by a client to connect, authenticate and retrieve a component posted for him

4.3 Results

Figures 3 and 4 show the results obtained with our prototype. They consist in an average based on a hundred values, removing the higher and the lower one.

Figure 3 (A) shows the time a client spend connecting the web service and obtaining an acquittal. This time varies from 4 to 5 seconds depending on the device used. The difference is due to the power of the devices, their ease to manipulate XML and the version of Android used. Here the best result is obtained by the HTC. Independently from the characteristics it is mostly because the HTC overlay is light and the Android version used is also less powerful than newer ones (less services running).



(A) (B)

Fig. 3. A : Sending a request and B : Package a local message via DDS

Figure 3 (B) gives the time spent from a request sent from a client to the moment the response message is effectively posted to DDS. Each of the devices spent the same time to achieve this experiment, showing that the difference obtained in (A) with different devices is mostly due to processing the acquittal. Results obtain in (B) are increased in (C) if the needed component has to be downloaded from an external server. There, the time depends on the device used but in each case, time spent to process a request is still acceptable, knowing that the client is supposed to potentially look for a response later.

Finally, in figure 4 (D), the time spent by a client to authenticate and download a component posted for him is described. The difference between devices is mostly due to the Android environment. 3.1 version manages data in a specific way for os 3.XX. Whereas for Android version 2, the older version includes less services which make XML processing and data downloading longer. This time include the installation time of the component, i.e. the installation and

(C) (D)

Fig. 4. C : Package a distant message via DDS and D : Get a message posted

activation of the component on the Android device until it is available to other components.

5 Conclusion

In this paper we showed the interest of bringing a context-aware framework for transportation services together with asynchronous communication capabilities. Indeed, network connection loss, cost or speed could become an important limitation for mobile user, especially as travel distances increase and public transports are widely used.

Mobile applications are built from different services and the assembly depends on the context. The main issues in this domain are both connecting services and obtaining data. If services could be accessed remotely (like web services), they can also be downloaded locally on the mobile device. And it goes even with data. By relying on asynchronous communication we are able to post request for both data or service downloading. Afterwards, the receiving of components or information could be performed at the appropriate time.

We illustrate the architecture we are proposing with a prototype and results showing feasibility as well as acceptable performances for asynchronous context-aware interaction. Different experiments has been made over a concrete platform to show that using asynchronous request with DDS is not an important additional cost compared to synchronous interaction.

However, the architecture can be still improved. Here we are using a number of context elements identified as the most representative for the mobile domain. These context elements could still be increased in order to refine dynamic adaptation opportunities and be able to switch from asynchronous to synchronous (and vice versa) at the most suitable time. Moreover, due to the multiplicity of communications capabilities, it is still necessary to handle the case where asynchronous request is send but the need is satisfied before the reception of the answer. This could occur when a neighbor or a local access point has been reachable and had the data or services requested at disposal.

References

1. B. Gaultier, R. Ben Rayana, J.-M. Bonnin, Context management systems applied to mobility, in: M. Berbineau, M. Itami, G. Wen (Eds.), ITST 2009, 9th International Conference on Intelligent Transport Systems Telecommunications, IEEE Computer Society, Piscataway, NJ, USA, 2009, pp. 330–335.
2. D. Popovici, M. Desertot, S. Lecomte, N. Peon, Context-aware transportation services (cats) framework for mobile environments, in: International Journal of Next-Generation Computing, Vol. 2, 2011.
3. D. Popovici, M. Desertot, S. Lecomte, T. Delot, When the context changes, so does my transportation application: Vespa, Procedia Computer Science 5 (0) (2011) 401 – 408, the 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011) / The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011). doi:10.1016/j.procs.2011.07.052.
4. C. Escoffier, R. S. Hall, P. Lalanda, ipojo an extensible service-oriented component framework, in: IEEE International Conference on Service Computing (SCC'07), Salt Lake City, USA, 2007, pp. 474 – 481.
5. A. Bodhani, Smart transport, IET JOURNALS AND MAGAZINES 7 (6) (2012) 70–73.
6. Z. Xiong, H. Sheng, W. Rong, D. Cooper, Intelligent transportation systems for smart cities: a progress review, Science China Information Sciences 55 (2012) 2908–2914. doi:10.1007/s11432-012-4725-1.
   URL http://dx.doi.org/10.1007/s11432-012-4725-1
7. L. Anthopoulos, P. Fitsilis, Considering future internet on the basis of smart urban cities a client-city architecture for viable smart cities, in: INTERNET 2012, The Fourth International Conference on Evolving Internet, 2012.
8. OMG, Data distribution service v1.0, Tech. rep., OMG, document formal/04-12-02 (2004).
9. Prismtech, Opensplice dds v5 reference guide, Tech. rep., Prismtech (2011).
   URL http://www.prismtech.com/opensplice