# Design of Mobile Services for Embedded Device

Guy Lahlou Djiken
Laboratory of Algorithms, Complexity and Logics, LACL, UPEC University
Créteil, France
guy-lahlou.djiken@lacl.fr

Sanae Mostadi
Ecole Supérieure d'Informatique Appliquée à la Gestion, ESIAG, UPEC University
Créteil, France
mostadis@miage.u-pec.fr

Fabrice Mourlin
Laboratory of Algorithms, Complexity and Logics, LACL, UPEC University
Créteil, France
fabrice.mourlin@u-pec.fr

***Abstract*—The design of distributed applications requires theoretical knowledge and hands-on experience. Our work is about distributed applications based on embedded platforms, such as smartphones or tablets. We define a software chain development from design to implementation where services are designed through interface diagrams and component diagrams. From these declarations, we are able to generate software descriptions into two languages. Android Interface Description Language (AIDL) is utilized for local services to an embedded platform. Web Application Description Language (WADL) is utilized for remote services. Such services are called from one platform to another one. The first kind of description allows developers to create Android services. Then, WADL description provides all the features for building Restlet Web services. We applied our strategy to the design and building of a case study on medical picture set management. Embedded tablets can take pictures during the users' activities. Local services allow users to display their medical picture through specific viewers. Remote services are set to expose these data to specific medical material. So, we provided a way to exchange technical data from well spread platforms to medical application servers.**

***Keywords-mobility; data collection; mobile service; distributed application.***

## I. INTRODUCTION

Tanenbaum defines a distributed system as a "*collection of independent computers that appear to the users of the system as a single computer*". This means that two features are essential: independent and suitable software for hiding the architecture from the users [1].

We consider a distributed system as a collection of autonomous computers linked by a network and using software to produce an integrated computing facility. The size of a distributed system can belong to a local area network (10's of hosts) or a metropolitan area network (100's of hosts) or a wide area network (internet) (1000's or 1,000,000's of hosts). The key characteristics of such distributed systems are the resource sharing, where data source or external device are used by applications. Then, the use of open standard allows to build applications which need to have the components of a solution work together [2]. The concurrency property is also important in the fact that multiple activities are executed at the same time [3]. This reduces latency and allows to hide blocking with some computing.

The scalability in size deals with large numbers of machines, users, tasks, etc. This property also occurs in a location with geometric distribution and mobility [4]. The subject of our work is the design of distributed applications based on services. When considering a scalable application design, a service helps to decouple functionality and thinks about each part of the application as its own service with a clearly defined interface. For SOA application (Service Oriented Architecture), each service has its own distinct functional context, and interaction with anything outside of that context takes place through an abstract interface, typically the public-facing API of another service.

Building a system on a set of complementary services decouples the operation of those pieces from one another. This abstraction helps establish clear relationships between the services, its underlying environment, and the consumers of that service. Our work is about the use of services, which are Web services or embedded services. Both types occur in real projects, and it seems to be essential to adopt the same design approach. Section II describes our methodology for specifying both types of services. Section III is about the use of intermediate representation between design charts and computer representations. The following section is about a way to provide an implementation. In the last two sections we describe a case study and we built on the management of the pictures with their localization. Finally, we summarize the results explained in this paper.

## II. DESIGN OF DISTRIBUTED SERVICES

Client/server, 3-tier and n-tier distributed applications and cloud computing, open up new opportunities and ways to design systems and develop applications. The design challenge is the main step of the life cycle of any project. The definition of message exchange pattern is essential for the declaration of each remote service. An object-oriented modelling approach is often used to describe business requirements, identify components, their interactions and placement in a multi-tier environment.

We have chosen UML description language [5] as a specification language. There are a lot of charts which can help designers perform requirement specification. We have selected a deployment diagram for architecture level and how materials are linked. Next, the use of component diagram is the core of our methodology with the specification of interfaces and the declaration of signatures.

 ISBN: 978-1-61208-547-0 

### A. Design step of distributed services

#### 1) A service approach

Similar to other distributed applications, Web services have a specific structure and behavior. The structure is the static part of Web services, which is composed of the candidate classes and their associations. The behavior is called the dynamic part. It represents how the Web service are executed in terms of sending requests, preparing responses to these requests, and how they will be sent back to the clients.

The Unified Modelling Language (UML) [5] gains greater acceptance among software designers, not only because of its standardization by the Object Management Group (OMG) [10], but also because of the high support from tool vendors, such as IBM and Oracle.

#### 2) First step in our case study

Throughout our paper, we use a case study about the management of pictures which are taken with mobile devices such as smartphones and tablets. The main goal for an end user is to know precisely where a given picture is. More precisely, if several devices are used in a lab, it could be convenient to localize the pictures on the devices without any upload of working pictures on a common data server.

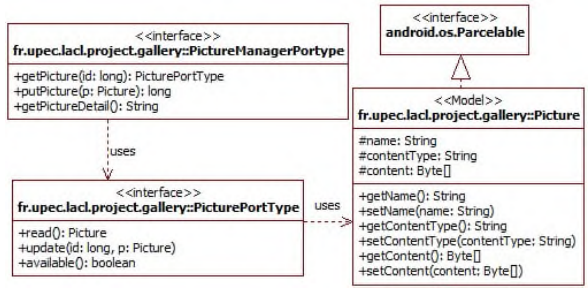


Figure 1. Precise declaration of interface and signature

The main goal of the Web service requirements analysis task is to capture and gather the requirements for the target Web service. This includes the identification of the precise services that have to be provided. This means that UML interfaces are defined in a package structure. For instance, assume a context where a set of pictures has to be exposed to a network with HTTP methods. So, Figure 1 will be the first step of the requirement specification.

This short example stresses 2 main tasks: the naming and the signature definition. Type and name of the domain and co-domain are essential to the future implementation and the clients. All these definitions are relative to a namespace (in our example `fr.upec.lacl.project.gallery`). This allows reducing name conflict. A package structure is an ideal entry point into a project dictionary.

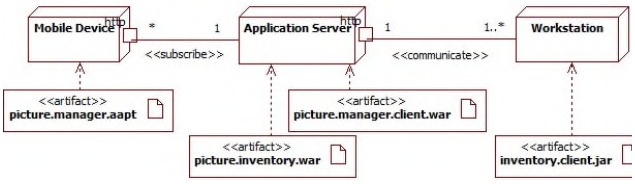


Figure 2. Deployment diagram

On the other hand, a material description provides all the details useful for the deployment step. In our previous example, the occurrences of the service are deployed on mobile devices. The clients could also be installed on mobile platforms or workstations. In Figure 2, a potential deployment diagram is described as a mobile application server deployed over a mobile device. Its client is installed on an application server. When all data are collected about the pictures, the other artifact, called picture.inventory.war deployed over the application server, can answer the requests of the standard clients.

From this view, we define several artifacts. They play the role of deliverables. Each of them will provide one or more components. A component diagram gives a snapshot of a runtime. Each component has provided interfaces and also dependencies on other parts of the software. Also, we can check how precisely the requirements are defined. This allows defining the used network protocol and the message exchange pattern. For instance, the requests to the `PictureManager` service is considered synchronous and parameters are exchanged through an XML format

This component diagram is also the support to express no functional properties, such that the maintainability of the set of services and the management of several versions. All the components follow the OSGi specification (Open Service Gateway Interface). A feature of OSGi technology is its portability, since it can be implemented both in the terminal board as well as in conventional applications or servers [6]. In this context, the OSGi technology is designed to address the other non-functional aspects, such that to enable the management of complex applications and to improve the quality of service applications for administration at runtime; see Figure 3.

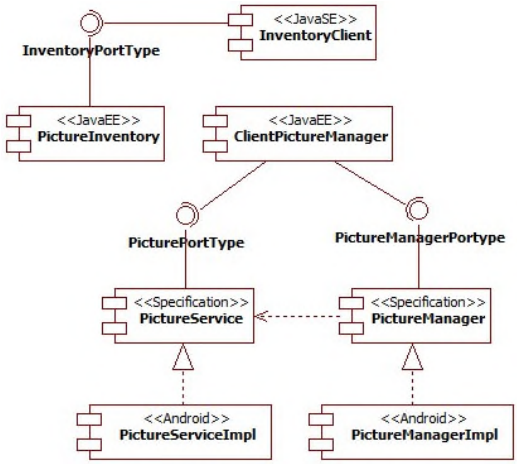


Figure 3. Software architecture of case study

In Figure 3, all components are placed. The naming convention allows readers to understand the correspondence between components and artifacts. There are three kinds of components depending on the kind of deployment node. This diagram highlights the roadmap of our development. So, because Figure 2 requires different kinds of platforms, then,

 ISBN: 978-1-61208-547-0

the next refinements are going to provide more details about the technical features.

### B. Integration testing

The integration testing is a level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. In our context, it means the integration of the three parts: mobile part, server part and a client part. This level of test can be considered as business routes where each of them is a use of our distributed application. In Figure 4, we describe the integration scenario where the application server sends requests to mobile platforms and collects the URLs of pictures and their technical features.

This sequence diagram plays the role of validation after the integration of all the components and their deployment on to the set of materials.

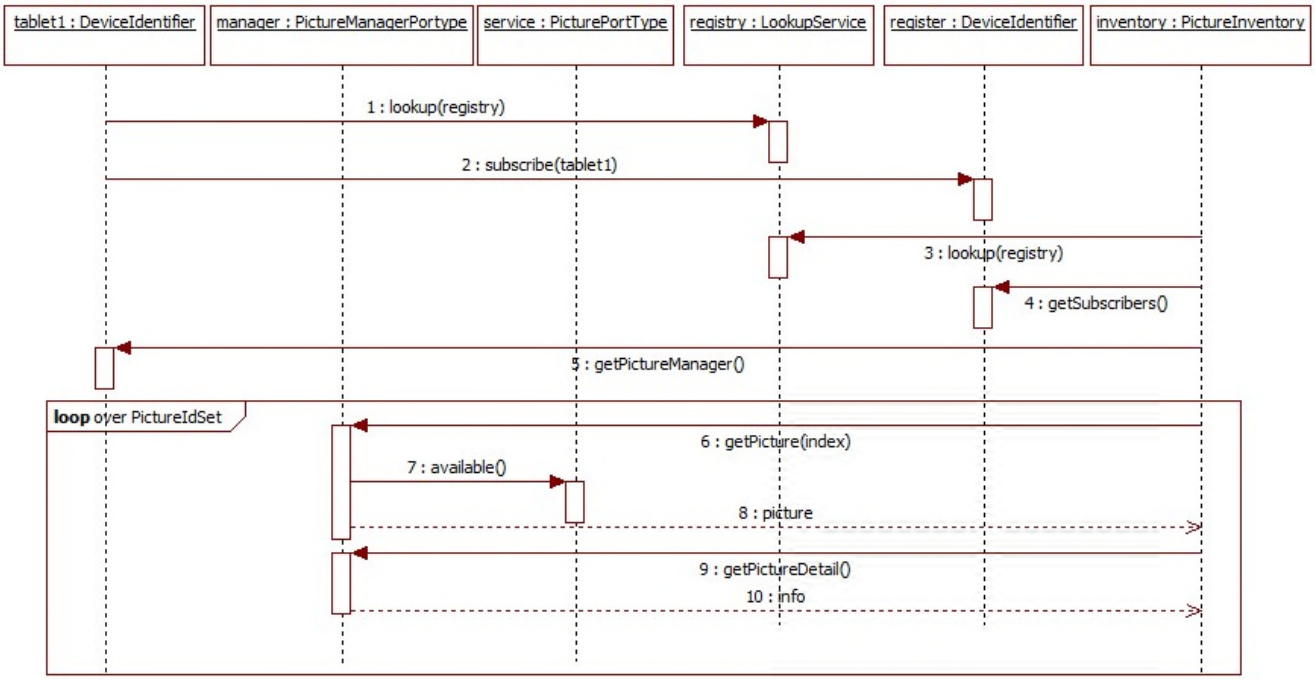


Figure 4. Interaction diagram as integration test

We also use such kind of diagrams when we study the impact of a scenario on the other behaviors of the application server. For instance, the problem can be to understand what the consequences of the data collections are during the subscription of other mobile devices. It seems to be obvious to require that the main business functionalities have to be isolated and the use of one mobile device is independent from the use of another one.

Figure 4 shows the interactions between a tablet and the application server. First, the mobile device is registered and a collector service validates the availability of all the data around the pictures (content, format, identification, localization, etc.). This diagram can be extended with the introduction of other mobile devices or the interaction with other scenarios, but this will introduce some noise into the description and the role of such diagram will be reduced.

## III. INTERMEDIATE REPRESENTATION

From the previous set of diagrams, we have to continue towards a more technical representation. As we can observe, this distributed application is based on the use of remote service. These services are clearly defined and, depending on the kind of platform, we use a precise approach.

### A. AIDL services

The IDL (Interface Definition Language) [7] is generally language independent for the service specification. It is used theoretically for generating C++ or Python stub code from it. The Android one is Java-based though, so the distinction is subtle. One difference is that there is only a single interface in an .aidl file, while Java allows multiple classes/interfaces per Java file. There are also some rules for which types are supported, so it is not exactly the same as a Java interface, and it is not allowed to use one instead of AIDL.

In the context of mobile programming, a service is an application component that runs in the background without a user interface. In our case study, the picture manager can perform data collection by using a background service to prepare data for a foreground application. This means another application of the mobile device. This is quite important because the consequence is that a service built from AIDL cannot be used remotely.

Services work in the background, even though the application is running neither in foreground nor in the background. A service might handle long running tasks like network connections or retrieving database records with the help of a content provider from the background. In our case

 ISBN: 978-1-61208-547-0 

study, two interfaces are defined to expose services on the mobile platform: these are `PictureManagerPortType` and `PicturePortType`; see Figure 1. So from these declarations, we transform them into two .aidl files.

These files (called PictureManagerPortType.aidl and PicturePortType.aidl) define the interfaces that declare the methods and fields available to a client. AIDL is a simple syntax that lets the designer declare an interface with one or more methods, that can take parameters and return values. These parameters and return values can be of any type, even other AIDL-generated interfaces. Then, the AIDL compiler creates an interface in the Java programming language from the AIDL interfaces. These interfaces have an inner abstract class named Stub that inherits the interface and implements a few additional methods necessary for the IPC call (Inter Procedure Call).

The next step is to create two classes that extend our previous interfaces PictureManagerPortType.Stub and PicturePortType.Stub and implements the methods we declared in our .aidl file. Then, we extend the Service class and override `Service.onBind(Intent)` to return an instance of one of our classes that implements one of our interfaces. The parameter intent plays the role of incoming message. The corresponding aidl descriptions of Figure 1 are given in Figure 5.

The primitive types are in direction by default. We limit the direction to what is truly needed, because marshalling parameters is time expensive. We have a class called `Picture` that we would like to send from a client process to the implementation process through an AIDL interface. We have made the Picture class which implements the Parcelable interface. The consequence is the overriding of the method public void `writeToParcel(Parcel out)` that takes the current state of the `Picture` and writes it to a parcel. The dual method is the method `public void readFromParcel (Parcel in)` that reads the value of a parcel into a `Picture`.

```
package fr.upec.lacl.project.gallery;

interface PictureManangerPortType {
    PicturePortType getPicture(long id);
    long putPicture(in Picture p);
    String getPictureDetail();
  // other methods are added in the case study.
}
```

```
package fr.upec.lacl.project.gallery;

// Declare Picture so AIDL can find it, knows
// that it implementsthe parcelable protocol.
parcelable Picture;
```

```
package fr.upec.lacl.project.gallery;

interface PicturePortType {
        Picture read();
        boolean update(long id, in Picture p);
        boolean available();
  // other methods are added in the case study.
}
```

Figure 5. aidl output files

### B. REST services

The use of AIDL is required because of application sandboxing. Each application in Android runs in its own process. An application cannot directly access another application's memory space. In order to allow cross-application communication, Android provides the inter-process communication protocol. IPC protocols tend to get complicated because of all the marshaling/unmarshaling of data that is necessary, but it has also a main limit: it is not possible to use it in a remote manner.

Today, a remote access is a common requirement, but the installation of a Web server on a mobile platform is not so natural. Also, we propose to use remote access by the use of the REST (Representational State Transfer) service through the use of Google implementation called Restlet. It relies on a stateless, client-server, with cache communications protocol, and generally, in all cases, the HTTP protocol is used. REST is an architecture style for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines.

As a programming approach, REST is a lightweight alternative to Web Services and RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, etc.). Much like Web Services, a REST service is platform-independent, language-independent, standards-based, runs on top of HTTP, and can easily be used in the presence of firewalls.

There are several reasons for having a Web server on a mobile phone. The main one is to allow third-party applications, on other phones or other platforms to access the phone remotely. This requires strong security mechanisms that are provided in part by the Restlet framework as well as network level authorizations by the carrier. We have decided to apply a Proxy design pattern to hide Restlet mechanism. So, each AIDL service is equipped with a Restlet service. To sum up, the AIDL implementation is used as a local facet on the mobile device and the Restlet implementation can be considered as a remote facet from other platforms.

In accordance with the Proxy design pattern, we have declared a subclass of the `ServerResource` class which belongs to the Restlet framework. Our class is called `PicturePortTypeResource` and has an attribute which is the previous AIDL implementation. Both classes implement the same business interface, but this last one provides our local service on the http protocol as a web resource. Figure 6 shows the main changes. Two technical packages are drawn to precisely the role of our technical classes.

Now, this mobile part is accessible from other mobile devices and also from workstation and application server, if necessary.

## IV. CODE CONSTRUCTION

We design the embedded part with respect to such properties, such that the independence of the layers and interoperability remains. It means that the client part of the

previous service does not know any technical details of our solution. This preserves the client from the changes of the new versions.

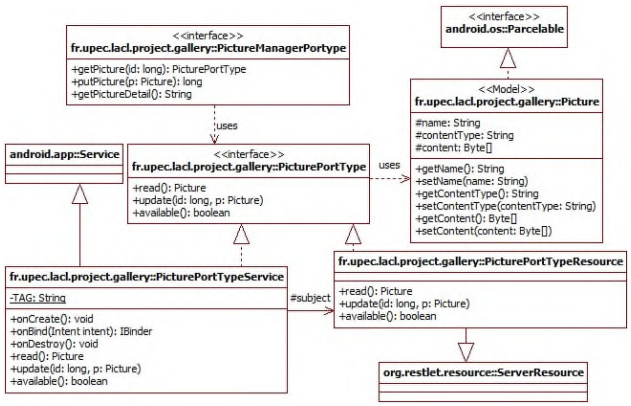


Figure 6. Design class diagram of the mobile part

### A. JavaEE implementation

As explained previously, the middle layer is the pilot of the data collection. After the subscription of a mobile device, requests are sent periodically from the application server to the mobile device. Applications that model business work flows often rely on timed notifications. We schedule a timed notification to occur at time intervals. Then, the collected data are stored on the application server. Of course, other mobile devices can subscribe to that picture manager service even if several data collections are running. Both functionalities are isolated.

Another artifact is deployed on this application server: it is the inventory service. It is a stateless component which answers to the presentation layer running on a client workstation. The role of the inventory service is to answer to the client about the previous data collections. For instance, assume several mobile devices are previously registered, so a client can ask precisely to know where a picture, called "picture1" under a JPEG format is. The structure of that part is more convenient: it is a three tier layer. These different responsibilities of an application are broken up into distinct tiers, typically:

- The integration tier for data transformation and persistence services. The persistence unit is about the details which are collected during the data collection.
- The business tier for the validation, business rules, workflow and interfaces to external systems. The request is expressed by a subset of the features of the pictures. This means the content type, the size the annotations, etc.
- The presentation tier for user interface generation and lightweight validation. The web panels allow the requester to define his need.

The requests between the presentation and business layers are synchronous over TCP protocol, but a message broker is used to separate client and service. The exchanges are totally asynchronous between the business and the integration layers. This is essential because the integration part can be considered as a cache of the database for several Web applications.

### B. JavaSE implementation

First, we use a web explorer to send http request and to display html tier. This display is a default graphical user interface used to send requests about the location of images. Next we have provided an API to develop new requests into programmatic clients. This is particularly useful for the automatic functional tests. This allows us to replace the use of Selenium tool of our own test application.

Our API also allows other developers to program new client tiers. It is based on the use of REST services which send requests to our business tier. Because we have chosen a REST implementation with the WADL generation (Web Application Description Language) [8], other developers can build their own version of our API. Also, SOAPUI tool [9] provides an easy way to create test suites of our business tier.

Our next case study is built with a lightweight client tier. In this context, the user is sure that the Web client is well suitable for the version of the business tier. Moreover, a comparison with other testing tools can be done, especially for performance measures.

## V. CASE STUDY

As we explained in our contribution, our case study is about the management of the pictures on Smartphone. Several embedded devices are used, for instance, in a lab or in a classroom. So, a distributed tool is necessary to locate precisely where the pictures are. More generally, such kind of tool is useful for the whole management of the pictures. This means collect, remove, transfer, duplicate or transform to an appropriate format.

### A. Deployment view

Before starting our case study, we have to deploy all artifacts on a given computer, as mentioned in Figure 2. Next, services have to be started by local servers. So, observations and measures could be done by a tester.

#### 1) Mobile data tier

Under Android 4.2 operating system, the mobile devices are used by members of a laboratory to take photos. The camera records the pictures into a gallery where each of them corresponds to a separate file with a set of features (name, format, size, date, owner, etc.). Because, a gallery can be considered as a set of pictures, each picture has its own name for their identification. Often, the name is generated by the software component which manages the camera. This means that the name is not easily known by the scientist.

For a test phase, the first activity is to take several photos and then, register the mobile device as a data tier to a business server. This will engage a set of REST services as and points to the gallery of photos

#### 2) Business tier

Its first objective is to be ready for receiving registration for all the mobile devices. From its point of view, the mobile devices are considered as a distributed data set of pictures. Concurrently, it performs a data collection about the features of the photos. This is not a collect of the photos because this

 ISBN: 978-1-61208-547-0 

will spend too much time. But this activity is to bind all the features, such as localization, into a registry for future requests. The inventory activity is managed by a timer. Also, regularly, a mobile device receives requests about new pictures if there are until the end of its registration onto the business server.

A third activity is to answer to the end users who want to localize the photos which are taken during a given period of time. Additional conditions can be set, such as the content type, the dimension of the picture, the size of the file, etc.

*3) Client tier*

In the test phase, we use a Web client for sending the requests. This client is received by sending an HTTP request from a navigator. It allows end users to define precisely the photos that they want to have access to. The answer of a request is a set of links. They can be used to access the embedded devices and the concrete photo. So, by the end of a test, this means: a request and a click on a hypertext link, a photo is displayed in the Web browser of the end user.

### *B. Artifact deployment*

*1) Mobile data tier*

In order to install third party applications on our Android phone, we need to install APK (Android Package, files). The way we usually do is like the next iteration, but it is for testing:

- Plug in an USB cable to a PC and mount a SD card on my computer
- Get the APK file some
- where on the SD card on the phone
- Unmount the SD card on the PC, allowing the phone to see the SD card contents again
- Use Astro File Manager or some similar app to browse to that file on the SD card and select it, which will prompt us to answer if we want to install the app on the phone.

For the end users, we have defined a simpler strategy based on the use of the local repository. We deploy the .apk file on a local server (apache http server) with a static IP to make the file available for download. Now, the end user has to open the download link of the apk file in his mobile browser. The device will automatically start the installation after the download completes.

*2) Business tier*

We use an application server called JBoss where our applications are installed though an ear file (Enterprise Application aRchive). The standard configuration of JBoss provides a very simple and convenient system for deploying applications, but not necessarily suitable for a production environment.

As standard, the deploy directory is a configuration for deploying services, components and applications. Just include a file according to the specific type of component specifications for JBoss deployment take into account. It is possible to deploy the files to the deploy directory or its subdirectories. Each file type is taken into account by an appropriate service deployment. The EARDeployer service is used for our two main components: the registration of tablets and the data collector.

The AbstractWebDeployer service is used for the Web application called by the client. It is implemented for the servlet container TomcatDeployer. The archive files are in the format war (Web ARchive).

*3) Client tier*

In the test phase, we use a Web client for sending the requests. This is a set of JSP pages which belongs to the previous Web application. Also, the client tier is just a Web browser which is already installed on the computer of the client.

We also use Java Web Start, which is a mechanism for program delivery through a standard Web server. The Java GUI client is downloaded to the client and executed outside the Web browser. The GUI client does not need to be downloaded again in the next run. If the GUI client is updated, a new version will be downloaded automatically. The jar file contains an XML descriptor with an XML schema. It specifies the resources needed to run Java Web Start applications. It also defines the URL location of the jar file, VM arguments and other resources that JRE on the client side should know to start Java Web Start GUI client.

Such GUI client that needs access to system resources, like file system, network connections, etc., needs to be signed. Also, we generate a keystore (certificate) and attach it to the jar file. After that, an end user is able send requests to the business tier and also to access to mobile device.

### *C. Measures*

Measuring the execution time is a really interesting, but also complicated topic. To do it right, in Java, we have to know a little bit about how the JVM works: generation decomposition and so on. But, we do not have the same VM on all the nodes of the network. The mobile devices have a DVM (Dalvik VM), the business tier and the client tier have a JVM (Java VM) but the versions are not correlated.

Also, we use a "ready to run" benchmarking framework that addresses many of our issues [7].

*1) Measures Method execution time:* The framework's essential class is named Benchmark. It is the only class that we use for the computation of measures; everything else is ancillary. Client and business tiers are observed by instances of the `Benchmark` class. We supply the code to be benchmarked to the `Benchmark` constructor. The benchmarking process is then fully automatic. Then, we generate a result report. The only restriction is that the code needs to be contained inside a `Callable` or `Runnable`. Otherwise, the target code can be anything expressible in the Java language.

*2) Business tier Measures:* there are two sets of measures. One is about the requests between the mobile devices and the application server. There two main tasks are: one is the registration of the mobile devices, and the second is the data collection which is started and ended by the application server.

The other set is about the treatment of the requests of the clients. Each request is received and treated by a business

 ISBN: 978-1-61208-547-0 

action which is also a `Runnable` instance. This means that we have measures on it. Both are interesting and their observations involve future improvements.

*3) Results:*

Table I presents measures of `RegistrationTask` class. It is a `Callable` subclass and its method is invoked when a mobile device needs to belong to the community of the mobile data tier. Next, a data collection will be applied.

TABLE I. REGISTRATION OF MOBILE DEVICES

| Measures | Method execution time | | |
|---|---|---|---|
| | *First time* | *Mean time* | *Standard deviation* |
| Registration | 112.901 ms | 108.501 ms | 725.510 µs |

In the meantime, we have additional information on it: deltas: -35.205 µs,+46.206 µs).

For the standard deviation execution time, we have the info: deltas: -161.405 µs, +361.108 µs

Table II presents measures of `DataCollectionTask` class. It is a `Runnable` subclass and its behavior is managed by a timer. At each interval of time, a data collection is started on a given mobile device. By the end, the changes are updated on the business server. This task is not linked to the previous one and several data collections are started concurrently in a manner that there is no effect from one data collection onto the other ones.

TABLE II. DATA COLLECTION ON A MOBILE DEVICE

| Measures | Method execution time | | |
|---|---|---|---|
| | *First time* | *Mean time* | *Standard deviation* |
| Data collection | 225.910 ms | 220.050 ms | 555.004 µs |

In the meantime, we have additional information on it: deltas: -31.520 µs,+41.602 µs).

For the standard deviation execution time, we have the info: deltas: -124.040 µs, +302.088 µs

Table III presents the measures of the `ClientRequest` class. It is also a `Runnable` subclass and its method is invoked when the end user sends a request about the URL addresses of several photos. Next, all the features of the user request are parsed and a result is computed from the previous data collections. Then, an answer is built with a set of URL instances. Each URL instance is a REST call to a service deployed on a mobile device.

TABLE III. CLIENT REQUEST ABOUT PHOTO ON DISTRIBUTED DEVICES

| Measures | Method execution time | | |
|---|---|---|---|
| | *First time* | *Mean time* | *Standard deviation* |
| Client request | 164.621 ms | 158.921 ms | 605.233 µs |

In the meantime, we have additional information on it: deltas: -41.115 µs,+51.261 µs).

For the standard deviation execution time, we have the info: deltas: -103.523 µs, +112.561 µs.

## VI. ANALYSIS

The first time that `RegistrationTask` instance was called, it took 112.901 milliseconds to execute. A point estimate for the mean of the execution time is 108.501 milliseconds. The 95% confidence interval for the mean is about -35/+46 microseconds, which is relatively narrow, so the mean is known with confidence.

A point estimate for the standard deviation of the execution time is 725.510 microseconds. The 95% confidence interval for the standard deviation is about -161/+361 microseconds about the point estimate, namely [235.389, 1086.51] µs, which is relatively wide, so it is known with much less confidence. In fact, the warning at the end says that the standard deviation was not accurately measured. The result also warns about the outliers. They are not significant in this case because the scenarios contain network connections. This involves blockings and time consuming only for negotiation between mobile devices and business server.

In the case of the data collection the first time that `DataCollectionTask` instance was called, it took 225.910 milliseconds to execute. A point estimate for the mean of the execution time is 220.050 microseconds. The 95% confidence interval for the mean is approximately -31/+42 microseconds, which is relatively narrow too, so the mean is known with confidence.

The standard deviation of the execution time is 555.004 microseconds. The 95% confidence interval for the standard deviation is about -124/+302 microseconds about the point estimate, namely [430.964, 857.092] µs, which is less wide than the previous case. So it is known with much confidence. In fact, the warning at the end notes that the standard deviation comes from the size of data which is collected. The result also warns about the variability in the measurement. The latter is sometimes excluded from the data set.

The last case is about request treatment. The first time that `ClientRequest` instance, was called, it took 164.621 milliseconds to execute. A point estimate for the mean of the execution time is 158.921 microseconds. The 95% confidence interval for the mean is approximately -41/+51 microseconds, which is relatively narrow too, so the mean is known with confidence.

Then, the standard deviation of the execution time is 605.233 microseconds. The 95% confidence interval for the standard deviation is about -103/+112 microseconds about the point estimate, namely [501.71, 717.794] µs, which is relatively small. So, it is known with confidence. In fact, the warning at the end notes that the standard deviation comes from the number of requests which are received by the Web application. The result also indicates an experimental error because of the latency of the network. When we compute other measures on a sample with a bigger volume of

 ISBN: 978-1-61208-547-0 

requests, then this overhead time is hidden or recovered by the computation of the answers.

## VII. Conclusion and Future Work

We have presented in this document our approach to the design (D), the implementation (I) and the evaluation (E) of mobile applications based on services. We have shown that there are two families of services: some of them are local and others are called from outside the mobile platform. Our esign is based on the use of UML diagrams and stereotypes to identify interfaces and the locality.

The implementation is based on Java programming and the use of frameworks, such as Restlet and Android. We have shown how to refine the diagrams towards a more technical description. A designer can sketch his/her applications with the use of local or remote services.

The evaluation is also described by interaction diagrams, which will become a test suite. We have built a case study based on our approach. It highlights all kinds of services (local and remote). So, interoperability is insured by the use of XML messages.

To conclude, our approach, called D.I.E. validates our design choice. Our experiments highlight the use of mobile devices as mobile data tier. As the number of embedded devices increases, our prototype shows that our software protocol supplies a way to exploit data on mobile devices without big data transfers.

## References

[1] J. B., Warmer and A. G. Kleppe, "The object constraint language: Precise modeling with uml", addison-wesley object technology series, 1998.

[2] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, (2006, September). "Reorganizing UNIX for reliability", In Asia-Pacific Conference on Advances in Computer Systems Architecture (pp. 81-94). Springer Berlin Heidelberg.

[3] M. Gould, M. A. Bernabé, C. Granell, P. R. Muro-Medrano, J. Nogueras, C. Rebollo, and F. J. Zarazaga, (2002). "Reverse engineering SDI: Standards based Components for Prototyping", In *Proc. of the 8th European Comission GI&GIS Workshop, ESDI-A Work in Progress*.

[4] J. Kołodziej, S. U. Khan and E. G. Talbi, (2013). "Scalable optimization in grid, cloud, and intelligent network computing–foreword", Concurrency *and Computation: Practice and Experience*, *25*(12), 1719-1721.

[5] T. Erl, (2004), "*Service-oriented architecture: a field guide to integrating XML and web services*", Prentice Hall PTR.

[6] O. Alliance, (2003). "*Osgi service platform, release 3",* IOS Press, Inc.

[7] O. M. G. Corba, "Common object request broker architecture" (Vol. 2), 1995.

[8] M. J. Hadley, (2006), "Web application description language (WADL)".

[9] C. Kankanamge, (2012), "*Web services testing with soapUI*", Packt Publishing Ltd.

[10] R. M. Soley and C. M. Stone, (1992), "*Object Management Architecture Guide: Revision 2.0"* (Vol. 92). Object Management Group.

 ISBN: 978-1-61208-547-0 