A mobile distributed system for remote resource access

Cyril Dumont
Leuville-Objects
3 rue de la Porte de Buc
78000 Versailles
France
cyril.dumont@leuville.com

Fabrice Mourlin
Logic, Algorithm, Complexity Lab
U-PEC university
61 avenue du Général de Gaulle,
94010 Créteil, France
fabrice.mourlin@u-pec.fr

Laurent Nel
Leuville-Objects
3 rue de la Porte de Buc
78000 Versailles
France
laurent.nel@leuville.com

ABSTRACT
Mobile and distributed systems involve multiple mobile computers processing data and communicating the results to each other, such as in electronic commerce or online voting, where the users are geographically separated. Our contribution is on mobile distributed applications based on embedded platforms such as smartphones or tablets. We provide a definition of a protocol called MEXP which stands for Mobile Exchange eXperiment Protocol. It allows the exposure of local resources on a mobile device to other mobile computers of the distributed system. The kinds of resources are pictures and sounds which are recorded with a mobile device during lab activities. They require the use of a local Wi-Fi network for the security of the recorded data. The lab activities evolve over time and the observers have remote access to the pictures and sounds for validation and tagging. This work has resulted in the acceptance of our mobile distributed application by an academic training team in the Biology department.

CCS Concepts
• Software and its engineering → Software notations and tools → Development frameworks and environments → Application specific development environments.

General Terms
Algorithms, Design, Experimentation,

Keywords
Mobile architecture, mobile REST services, NFC exchange, data collection; distributed application

1. INTRODUCTION
Professor Christopher Gill defines a mobile distributed system as a "collection of connected mobile computers that appear to the users of the system as a single entity". This means that two features are essential: software that is connected and is suitable for hiding the architecture to the users [1].

We consider a distributed system as a collection of autonomous mobile devices linked by a network and using software to produce an integrated computing facility. The extent of a distributed system can belong to a local area network (LAN) (10's of hosts) but not a metropolitan area network (MAN) (100's of hosts) nor a wider area network (WAN). The key characteristics of such distributed systems are resource sharing where data sources or external devices are used by applications. The use of an open standard then allows building applications which need to have the components of a solution work together [2]. The concurrency property is also important; in fact, multiple activities are executed at the same time [3]. This reduces latency and allows the hiding of blocking with some computing.

The scalability in size covers large numbers of mobile devices, users, tasks, etc. This property also occurs in a location with geographic distribution and mobility [4]. The topic of our work is the design and construction of service-based mobile distributed applications (deployed in a local cloud). When considering scalable application design, a service helps to decouple functionality and treat each part of the application as its own service with a clearly defined interface. For Service Oriented Architecture (SOA) [5], 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 Application Programming Interface (API) of another service. The use of local cloud offers some restricted access to our distributed system. In this context, the security is based on application roles. The use of other security aspects is out of the scope of this work.

Building a system on a set of complementary services decouples the operation of these 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 web services over http protocol, the definition of a business protocol, its implementation, its tests and its deployment. In Section II, we present related works about Web services on mobile devices and the need for communication protocol definition. Section III is about our protocol called MEXP (for Mobile Experiment eXchange Protocol). Section IV specifies our strategy to provide an implementation. The last section is dedicated to a case study we built on the management of scientific pictures with their localization. Finally, we summarize the results we have explained in this document.

2. RELATED WORKS
This section introduces the topics related to our contribution.

2.1 Mobile application and Web service
Many enterprises are creating mobile applications for their internal staff, for their customers, or both. These applications need access to data, business rules [6], and business processes. For architectural and security reasons these applications are typically built to access remotes services that provide the data and functionality that are required by the users, because many web applications have been built over the last decade using SOAP-based web services. Many software architects have made the assumption that these same SOAP-based services (Simple Object Access Protocol) are the best choice for mobile applications [7]. We believe that the use of RESTful services, with data in JSON format is a better choice for mobile applications whether the client device technology is iOS, Android, or even Mobile Web [8].

There are a lot of differences between SOAP and REST for use by mobile applications; SOAP [9], is a protocol specification for exchanging structured information via XML in the implementation

of Web Services in computer networks. REST: (REpresentational State Transfer), was defined by Roy T. Fielding [10] without this format constraint. It stresses that services can be interacted with via stateless representations of the targets of the service. The message exchanges occur over http protocol without any state memento. It leverages the application layer that was implemented to support the world-wide web.

SOAP is a well-known technology that has been used effectively within a SOA framework for some time. Changing services that use SOAP often means a complicated code change on the client. When this client is a web application server, this change is not necessarily problematic, but when the SOAP client is running on a mobile device, the problem of application update dissemination arises. Generating SOAP client code from WSDL's and XSD's is not really possible, and in the mobile world, this complexity is magnified by the fact that many organizations must produce the same mobile application for several platforms (iOS, Android, Tizen, etc.) [11].

REST was designed to operate with thin clients, like a web browser or test engine, through all types of networking equipment, to a variety of service types (Java, .NET, PHP, Swift, JavaScript, etc.). Because of this requirement for lightweight and flexible implementation, it is very simple and also very similar to the majority of web traffic on the Internet. While SOAP services always return XML, REST services provide flexibility in regard to the type of data returned. A REST endpoint can just as easily return a payload of XML data as a PNG image [12].

For the reasons mentioned in this section, we believe that RESTful services, when implemented properly, offer the best combination of: performance, bandwidth efficiency, robustness. It often makes sense to organize those services into a pattern mobile facade built using REST, and delivering data in JSON format. The mobile facade aggregates internal service endpoints into a single contact point for the mobile application. This facade will isolate the mobile devices from changes within the enterprise, thereby reducing the number of required mobile application updates. The facade can also implement some business logic to aggregate multiple internal service calls into a single mobile call for the purpose of simplifying the mobile application and improving the customer experience. This applies in the case of REST clients; however, this is less true in the case of REST servers where the implementation is resident on the device. A new abstraction should be added to this context with the application of a programming skeleton [13]

2.2 Protocol definition with formal language

When technical aspects are defined, the next step is how to use them. Common experience provides standard approach based on semi-formal languages like UML and a subset of diagrams [14] which are built in a given order. This strategy is time costly and today, the projects are built with other criteria than the respect of a methodology. The end users want to experiment an application quickly even if it is not complete. The developers look for users' feed-back as soon as possible because they can adapt or change their solution earlier. Therefore, it seems to be more comfortable to adopt a design approach in three main steps [15]. The first one is based on the requirement definition which is all the business rules and domain constraints of the end users. Its output is often a set of interaction diagrams. They define all the vocabulary of the message exchanges, the signature and also the elements of the distributed system at run time. The causality of the interactions is fixed with potential temporal aspects. The second one is a refinement of the first step based on the use of a rigorous intermediate language like Swagger, RAML, or API Blueprint, etc. [16]. These tools are specifications for documenting REST API. They specify the format (uniform resource location (URL), method, and representation) to describe REST web services. They also provide tools to generate and compute the documentation from application code. After the reading of the interaction diagrams, the designer describes in more detail all exchanges in a technical context; for instance, the use of the REST service over http protocol. The output is a rigorous description of a protocol in a computer science language, like JSON or YAML.

JSON stands for JavaScript Object Notation, and is a lightweight text-based open standard designed for human-readable data interchange [17]. It is derived from the JavaScript programming language for representing simple data structures and associative arrays, called objects. Therefore, the second step provides a JSON output where all endpoints are clearly defined with a uniform resource location, a verb and a structure for the payload [18]. This description is then the basis of at least three document generation steps. The most obvious one is the documentation of the API. It plays the role of a written contract between the developers and the users. The other two are for the service part and the client part. Each of them has its own technical constraints, for instance a programming language or a library version. They play the role of a skeleton for the service part and the role of stub for the client part [19]. At that point in the project life cycle, the business rules or the semantic actions should be implemented in the skeletons of the services. And their results have to be handled by the use of the stub on the client side. By the end of that approach, a packaging operation provides the artifacts which are deployed on the mobile devices and the embedded systems.

This development methodology considers the changes of the requirements as a common input of the project [20]. Because its three steps use tools with well-defined input and output, any changes are easily taken into account until the code generation. Programming rules have to be applied to isolate the generated code from the handcrafted code. It is obvious that the cut and paste operations in programming are always bug sources. Therefore, the handcraft code has to be preserved during code generation, especially on the server side where semantic actions have impact on a common data model. This involves the delegation principle which was already applied in past technology like Corba and the elaboration of POA (Portable Object Adapter) [21]. The same advice has to be respected with the client part where the generated stub has to be unchanged by any developer. When the changes do not consider the business part but the technical part, the consequences on the methodology definition are deeper. Its elaboration is based on a RESTful architecture as a first constraint [22]. If the change consisted in programming a SOAP solution, then we should have defined another multi-step approach with other tools like SOAPUI for code generation and document construction [23]. For the following parts, we consider as fixed the choice of RESTful architecture over our mobile distributed system.

3. Mobile Experiment eXchange Protocol

This section is on the definition of our communication protocol (MEXP) which is used to exchange between devices during biology experiments. The data are shared between the lab technicians and the analysts who play the role of domain expert. Because time is always too short, the analysis process starts during the experiments with the taking of pictures. These pictures and videos are taken in laboratory rooms where safety rules are applied about the number of people, the dress code, lab constraints, etc. The exchanges between all the practitioners are not as easy as a discussion near the

coffee machine. The dialog between the experimenters and the analysts is implemented by our communication protocol and deployed over the wireless devices of every member of the biology team. Even this network communication has its limits with the sharing of a local Wi-Fi access, and control over all the exchanges is a key feature.

3.1 Definition of the business context

All business applications are first defined by a list of requirements or a use case diagram with additional text descriptions. The job of the business analyst includes other statements such as the description of main case studies. In this section, we detail a nominal case study for a better understanding of outsiders interested in this topic. A biology experiment is a rigorous procedure done in a controlled environment for the purpose of gathering observations, data, or facts, demonstrating known facts or theories, or testing hypotheses or theories. Therefore, the experts define experiments for many reasons: the gathering of new data on a phenomenon, or the tracking of a precipitate during a time period, etc. In many cases, the process could evolve non-linearly over time and it could be essential to capture a whole experiment through a video or a set of pictures. These data are directly accessible to the experts who evaluate the results and tag the data depending on their criteria. A video file can be deleted if the time period is of no use to analysts.

At the beginning of an experiment, an access point is defined for a time period and all the devices are registered as Wi-Fi client. The device is often a tablet, a smart phone or a smart TV. The lab technicians take the pictures which are immediately accessible for the analysts. The pictures are attached to a device and their locations are registered. The analysts tag the results and can give new ideas for the next photo shoot. The documents (photo, video, sound) are saved on the hard disk of the device where they have been taken with the additional tags from the analysts. The devices of the analysts have access to the data through a network connection. None of the data are moved from one device to another. At the end of an experiment, the local data are indexed on the devices of the experimenters and the experiment is closed. Each item of an experiment is named based on the name convention of the laboratory and data can be found based on a regular expression on the names.

3.2 Protocol analysis

In computer networking there are different types of rules which are used for data transmission from one device to another device on the network such as Wi-Fi or Bluetooth; these rules are called protocols. There are two main types of protocol which are used to spread information on internet such as TCP and also UDP used for transmission of data in the form of data packets that are called datagrams. These two protocols are completely responsible for major networking in the field of information technology.

In the field of computer networking, protocol analysis is used for different processes and devices to decode the data which are transmitted in the form of packets from start towards end. In this domain, the terms header and trailer are more suitable and meaningful. These general concepts have to be refined in our business domain. Therefore, our goal is finally to provide the content of the headers and the trailers. To achieve this goal, we follow our three step methodology and first we describe through interaction diagrams the exchanges of our nominal case study. The UML language is used as the interaction diagram notation. We consider that each person works with only one connected device for the simplicity of the description and we use the role of these people in the diagram to point out the given device.

As shown previously, the protocol description has three main phases. In the initialization phase, the devices are prepared for communication. The experiment phase corresponds to the main part time when the pictures are taken, saved and exposed. The closeout phase finishes the experiment, closes the connections and saves the experiment metadata.

3.2.1 Initialization phase

After the creation of the new experiment in our local Cloud via an application service, an analyst changes the Wi-Fi settings of his device and transforms it into a hot spot which is available for the other members of the experiment. Another analyst or a technician lab (called Partner on the following figure) can then approach his device near to the device of the first analyst and the communication occurs via radio frequencies. The information of the Wi-Fi access point is transferred via Near Field Communication (NFC). The second device then looks for the defined connection to the local hot spot, registers it and receives an IP address. The access point allows all members of the experiment having access to the local network. This entry point is open by the main analyst of this experiment.

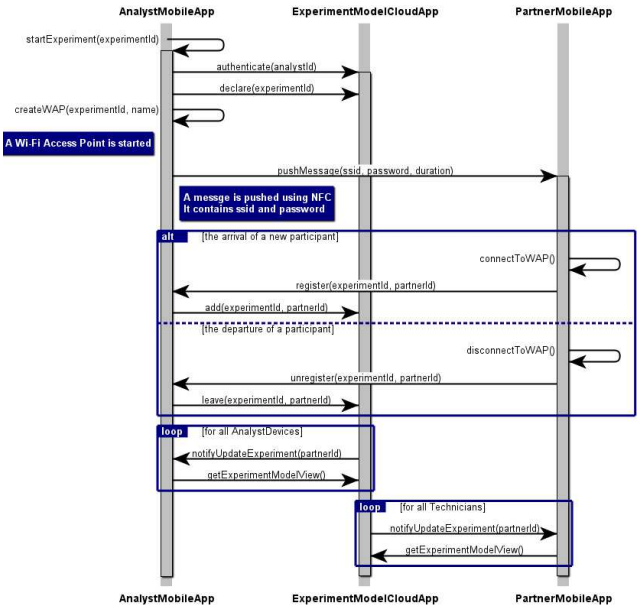

Figure 1 First phase of MEXP protocol

The new participant is then registered in our experiment model which is managed through a REST service. In consequence, all participants of the current experiment are notified about a change into the experiment model. The arrival of a new partner can occur during the duration of the experiment. If he follows the steps of the first phase [Figure 1], then all participants will know the new presence of an analyst or a technician lab.

Figure 1 also depicts the departure of a partner from an experiment. As we have just explained, the departure of an analyst or a technician lab is recorded in the experiment model through a REST service which is deployed in our local cloud. Next, all the participants of that experiment are notified about the change. This interaction diagram is a classic interpretation of a Model View Controller pattern (MVC) which is implemented in the world of mobile distributed systems. The data model stays available as long as the first analyst does not close the experiment and the Wi-Fi access stays available.

3.2.2 *Experiment phase*

The experiment phase occurs when the connection is ready to use. The technician lab starts to take pictures or videos and the new documents are automatically remotely accessible for all the analysts. The following figure shows as alternatives the three kinds of potential request. First, an analyst can ask for a picture to be read (or other kind of document: video, sound). For a given picture, an analyst can ask for a picture to be deleted from the local gallery of a mobile device which is or was being used by a technician lab. Finally, he can ask for a new picture based on a previous one which is named as parameter [Figure 2]. The original format is considered as divided into four quarters (TopLeft, TopRight, BottomLeft, BottomRight). The new photo will be linked to the previous one by a tag value. During this phase, the first analyst who has opened the experiment, has to maintain his access point available.

Tagging is essential for document accessibility. Tags establish logical reading order and provide a means for indicating structure and type, adding alternative text descriptions to pictures, videos and sounds and substituting text. Several tags are already defined in this application. They attach the opinion of the analysts to a validated document. An untagged document will display the words "No Tags Available" as a document of no use to the team members.

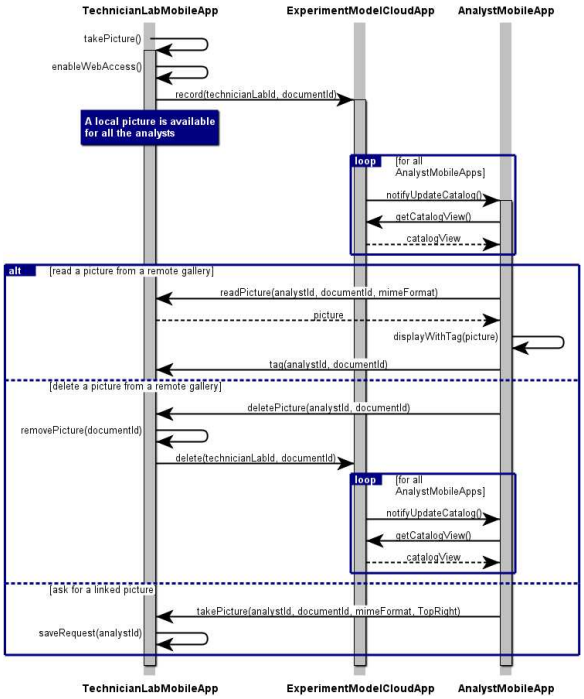

Figure 2 Second phase of the MEXP protocol

The picture and the tagged values are saved on the mobile devices where the pictures are. Therefore, the picture access is managed by a REST API deployed over this mobile device. Conflicts can occur when several analysts send requests which change the state of the pictures. For instance, an analyst sends a delete request and another analyst sends the same kind of request. The business algorithm indicates that the orders are considered on a first-come, first-served basis and will therefore not consider export priorities as a condition for selection. In that case, the first request will be treated successfully and the following one will fail with a response about the origin of the failure. Other kinds of conflict will be considered in the same manner; for instance, a delete request followed by a demand on a more precise picture. Due to network delay timestamp of requests may not observe the causality of events [24]. This involves the use of clock vectors for all the exchanges.

3.2.3 *Closeout phase*

The duration of an experiment is declared at its creation. Therefore, when this duration is achieved, a message is sent to all the partners of the current experiment, and then a first quick report is displayed on each mobile device about its contribution. Next, all the connections are closed and the Wi-Fi Access Point is stopped.

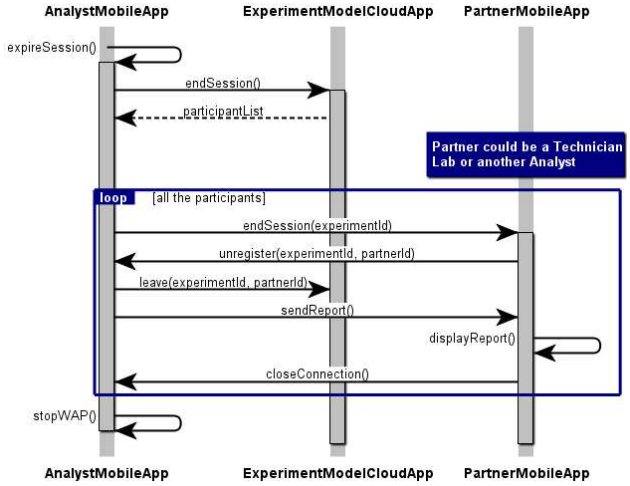

Figure 3 End of experiment session

To sum up, by the end of an experiment all the pictures [Figure 3], videos and sounds are tagged and saved on the devices. The consultation of the data is now possible by all the members of the research team. Furthermore, all the mobile devices are also available for a new experiment session with new records.

3.3 Protocol design

As we explained in section 2.2, the Swagger specification provides a way to describe our API using a specific JSON or YAML schema that outlines the names, order, and other details of the API. This means the formal expression of all the communication interfaces and also the data model used such as the description of what is an experiment in the cloud tier. The Swagger files are coded by reading the previous interaction diagrams. Next, different tools consume the Swagger file to generate the interactive API documentation and also code skeletons and stubs.

3.3.1 *Design process*

The writing of the formal description is progressive after the reading of each interaction diagram. In a first reading, the methods are translated into network requests by the definition of network verb, an exchange format, a list of parameters and potential exceptions. From the interaction diagrams [Figure 1], [Figure 2], [Figure 3], the names of the stimuli are preserved but new names appear such as the data types for the signature of the exchanges. A deeper reading of the interaction diagrams involves adding documentation about the element of the specification. The word "documentation" hides a complex task for the designer. It contains not only technical information but also the version of each sensitive element, its author. It also covers more technical concepts such as the selection of a protocol scheme, the uniform resource location (URL) for all the previous requests and of course, the description of all potential responses (code, status, message).

The next step of the design relates to the definition of the body of the requests and the responses. New data types are added to the specification and the relationships are expressed to simplify the whole description. The Swagger language contains basic data type which can be used at the lowest level of type. It also contains type constructors which allow array, enumeration and map to be expressed. This design step finishes by the definition of the headers of each exchange. Other features can be added such as a layer on the top of a multi-layer construction; for instance, the security aspect is often required in a context where the exchanges are responsive to external perturbations.

As an example, a part of the design of the first interaction diagram is given [Figure 4]; it provides 16 request declarations and response

```
swagger: '2.0'
info:
  version: "1.0.0"
  title: Mobile Experiment eXchange Protocol API
  description: The protocol defines the exchanges
between three main components; a Cloud application
which maintains the data model of an experiment, a
mobile application
  contact:
    name: Cyril Dumont, Fabrice Mourlin, Laurent Nel
    url: http://www.leuville.con
    email: cyril.dumont@leuville.com
schemes:
  - http
  - https
host: mexp.api
basePath: /mexp/v1
paths:
  /experiment:
    post:
      summary: an analyst starts an experiment
      description: |
        Launch a new experiment.
        Required query param of experimentId
      parameters:
        - name: experimentId
          in: query
          description: Identifier of the experiment
          required: true
          type: number
          format: double
      responses:
        200:
          description: Successful response
          schema:
            title: Acknowledgement
            type: boolean
        400:
          description: Bad request
        403:
          description: Forbidden
          schema:
            title: Origin of the failure
            type: string
        406:
          description: not acceptable
          schema:
            title: experimentId
            type: number
        409:
          description: Conflict
          schema:
            title: reason
            type: string
  /experiment/wap:
    post:
      summary: an analyst creates a Wi-Fi acc. point
      description: a Wi-Fi access point is created
      for the current experiment
      parameters:
        - name: experimentId
          in: query
          description: Identifier of the experiment
```

Figure 4 Swagger representation of Figure 1

when it is mandatory. A reader could at first believe that this document is just a documentation of the first interaction diagram. As we explained, it is much more than a textual description. The YAML format allows a concise and structured definition of the MEXP protocol. The first 20 lines are a declarative area where the main references are published: the title, the version, the authors and particularly the contact in case of requests. We have versioned our API because it helps to prevent invalid requests from hitting updated endpoints. It also helps smooth over any major API version transitions as we continue to offer old API versions for a period of time. The version needs to be in the URL to ensure browser explore capability of the resources across versions (the key basePath).

A list of protocols is included but, in our case it is an indicative information and not a restriction. The keys host and base path indicate that all the URLs will start with `scheme://mexp.api/mexp/v1` where scheme is either http or https.

The key called "paths" introduces the list of endpoints which compose the MEXP protocol for a given element: the *AnalystMobileApp*. In [Figure 4], each endpoint has a description, a verb, few parameters and a list of potential responses. Each parameter is declared precisely with a type, a direction and qualifiers. Each response has also its code, a description and the structure of its body if it exists. After the reading of this part of the design, a big step is achieved towards an implementation. It is missing the glue code between the input and the output: this is the business part

3.3.2 REST conventions

The business part of our project is on the access management of resources distributed over a set of mobile devices. It supports project construction rules about its life cycle, name convention and open standard respect such as the REST architecture constraint. This means the application of RESTful principles, that were first introduced by Roy Fielding in his dissertation on network-based software architectures. The key concept involves separating the MEXP API into logical resources (*Experiment*, *Record*, *Partner*). These resources are manipulated using HTTP requests where its methods or verbs have a specific meaning. We have reused the business jargon because it appears first in the URLs and next into the data models.

Once the main resources have been defined, the useful actions have been identified and how those map to MEXP API. RESTful principles provide strategies for handling CRUD actions using HTTP methods mapped which are very common such as:

`POST /experiment` in [Figure 4] this creates a new experiment.

`POST /experiment/wap` in [Figure 4] this creates a new Wi-Fi access point.

The relations exist within resources: experiment and record, RESTful constraints provide useful guidance. An experiment in a lab consists of a number of records. These records are logically mapped to the experiment endpoint as follows:

`GET /experiments/12/records`: this retrieves a list of records for experiment #12

`DELETE /experiments/12/records/5`: it deletes record #5 for experiments #12

There are actions that do not fit into the world of CRUD operations (Create, Read, Update, Delete). Therefore, the action is restructured to appear like a field of a resource (/partner/report) [Figure 3]. This

works because this action does not take parameters. In our example, it displays a report on the partner device and this action is mapped to a Boolean activated field and updated via a PATCH to the resource called partner.

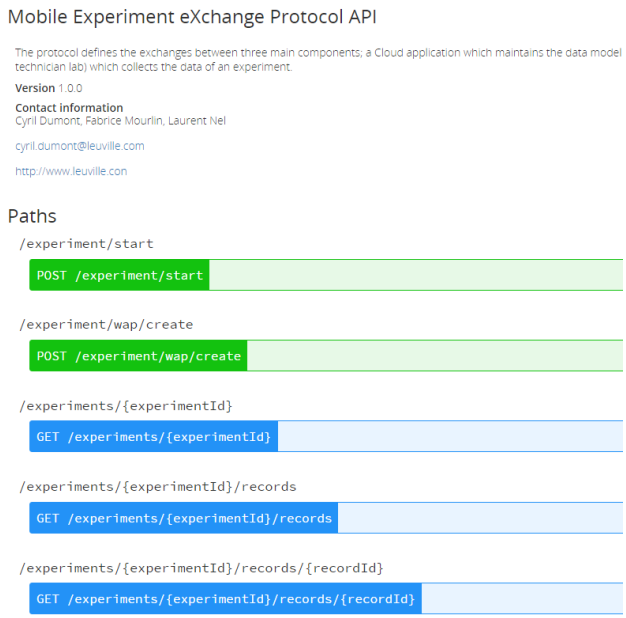

Figure 5 Interactive documentation of MEXP protocol

All the updates and creations return a resource representation [Figure 1], for instance an experiment or a record whose format is a JSON representation because the requests make modifications to fields of the underlying resource (an experiment, a record, etc.) that were not part of the provided parameters. This avoids the MEXP consumer having to send another request again for an update of his representation, and have the MEXP API return the updated representation as part of the response. The MEXP API does not wrap the responses in envelopes. There are a couple of justifications for doing this. First it makes it easy to include additional metadata or pagination information; some REST clients do not allow easy access to HTTP headers, and JSONP requests have no access to HTTP headers. However, new standards have appeared such as the Cross Origin Resource Sharing (CORS) and the Link Header from RFC 5988. They are in line with unwrapping and the removal of any envelope.

Finally, one of our strongest constraints is about the respect of HATEOAS rules [25] (Hypermedia as the Engine of Application State). The HATEOAS constraint decouples client and server in a way that allows the server functionality to evolve independently. There are a lot of mixed opinions as to whether the MEXP API consumer should create links or whether links should be provided to the MEXP API. RESTful design principles specify HATEOAS which roughly states that interaction with an endpoint should be defined within metadata that comes with the output representation and not be based on out-of-band information, although the web generally works on HATEOAS type principles and the Swagger documentation helps us to know about the resources and even about the expected formats. Therefore, we have decided that the resources (experiment, record, partner, etc.) should expose links to navigate across each other. These links are provided in the body of all the responses.

Additional information can be added to the previous specification by introducing examples when possible and then improving the end user document.

4. MEXP Implementation

A project development follows a precise project plan and our technical constraints are taken into account progressively. In the implementation phase, the use of a tool chain allows a reduction in development cost. The Swagger tool provides source code pieces, but for the service part, these are only skeletons. Furthermore, we use multiple technologies and their composition needs to combine code pieces and the tests of distinct sensors: Wi-Fi and NFC.

4.1 REST architecture

REST is the architecture style we have chosen for designing networked mobile applications. The idea is that a simple HTTP client is used to make calls between mobile devices and the cloud. REST is not a standard. There will never be a W3C recommendation for REST. And while there are REST programming frameworks, working with REST is so simple that we can often code with standard library features in languages like NodeJs, Java. The key constraint is that the mobile services and mobile clients must both agree on the media used, which in the case of the MEXP protocol is JSON. The URLs have been specified previously together with the links between resources.

Today, the development of REST services and clients is possible for all kinds of platforms: workstation, mobile device cloud, television, etc. The current version of the project uses a cloud part and a mobile device part. Therefore, the architecture has two main kinds of components: some are built for mobile devices and others for a cloud platform such as Amazon or Google. The distinction between an analyst and a technician lab is operated through the use of a mobile account and an authentication component. When a partner is logged, depending on his mobile account, he will have a role name. This role enables components and disables the other ones.

4.2 REST Implementation

The analysis of our protocol has highlighted three phases [section 3.1]: the initialization phase when the Wi-Fi access point is configured and then exchanged through NFC with the other devices of the same experiment; the experiment phase concerns all the MEXP exchanges of videos and pictures during the experiment; and the closure phase concerns the end of the experiment.

4.2.1 Service Side

In a software architecture, client/server is a program relationship in which one program (the client) requests a service or resource from another program (the server). In mobile distributed systems, these roles can evolve along the runtime and a requestor can expose a service for a subsequent request. In the MEXP protocol, all the mobile devices play both roles during an experiment.

4.2.1.1 Initial configuration between devices

NFC can be used in different situations: we use it after turning on a Wi-Fi access point when an analyst starts an experiment. We have focused our attention on NFC Data Exchange Format (NDEF) data attached to a tag. We transfer access point data through a tag. We have developed an Android NFC Tag. We push NDEF messages between 2 devices using the peer to peer principle. This works in the following way: one of the tablets registers the NDEF message to be pushed and as soon as the other tablet enters the field of the other tablets this message is exchanged. This tag has a type based on a record of strings: a value for the Service Set Identifier (SSID),

another one for the password, following by the creation date time and the duration of this access point.

This exchange of data can occur from the creation of the experiment until the expiry date or if the first analyst stops the access point prematurely. In parallel with this potential exchange of configuration data, other requests occur about the analysis of the experiment data, but these use REST exchanges.

4.2.1.2 REST service implementation

From the formal descriptions described in section 3.3 and Figure 5, the Swagger toolchain generates artifacts. It contains a code generator for the service skeleton generation. It is powerful when we create an API from scratch. The management of changes is not as simple as the toolchain is because it changes all the previous generated classes for new ones. We can minimize the manual intervention by using Spring Framework [26] and its aspect oriented programming library. Thus, the business code is injected at load time by a weaver and the definition of point cuts. We use the delegation principle and add method call code into generated classes which invoke our business code [27]. This technique comes from the definition of Portable Object Adapter in Corba library [21].

In the business part, additional classes are built to manage a whole experiment session. The classes are packaged into OSGi components (Open Service Gateway initiative) and loaded at runtime by the use of an Apache Felix instance [28]. This OSGi server is used in an embedded system on mobile devices and also on standard workstations. This allows us to load components at runtime and ensures that the application can easily evolve over the time. By the end of the experiment, the components are unloaded and the OSGi server stops. The garbage collector frees the used memory.

4.2.2 Client Side

There is a client over NFC and another one over Wi-Fi depending on the phase of the experiment. Both components play complementary roles during an experiment. The first allows a durable connection, the second serves to access data of the current experiment.

4.2.2.1 How to belong to an experiment?

When we develop an Android NFC tag reader, the first thing we want is our application to be notified when we get near a NFC tag. For this purpose, we use an intent filter. We have used a type of filter, so we have used a mime type. In other words, when a NFC tag NDEF is discovered and it has a mime type text/plain, then our mobile application will be started. Filtering with intents works if our mobile application is not in foreground. If our mobile application is running in foreground, it will not be notified, if move our mobile device near an NFC tag. In this case we have used a different technique called NFC Foreground dispatch.

Once we have created our filters, we have to interact with the NFC component in the mobile device. For this purpose, we use *NfcAdapter*, provided by Android SDK. Using this class, we can check whether the NFC is supported by the mobile device or whether the NFC is turned on or off. Once we know how to handle a NFC tag, we want to read the tag content. There are several types of content defined in NFC specifications; among them are: Media-type, Absolute URI, etc. Each type has it is own payload. In other words, a NFC NDEF data is composed of a Message. A message can contain one or more records. Each record is made by a header and a payload which contains the information about the Wi-Fi access point. When these data are read, then the device is configured as a Wi-Fi client of a local network during the experiment.

4.2.2.2 REST requestor

When a mobile device is connected to the network, its user interacts with the other partners of the experiment through the use of our business protocol MEXP. After creating an API in section 3.3 what we need is a client side code to access the MEXP API. There are plenty of supported languages for Swagger and we have selected Java language. The Swagger code generator needs a YAML swagger file as the input which is used to represent the MEXP API. After generating the Java client side code, we have as many client parts as there are entities in our MEXP protocol. To sum up, we have three main entities: *TechnicianLabMobileApp*, *Experiment-ModelCloudApp, AnalystMobileApp*. Each part is imported in the code where requests are created. By using these three classes we have been able to configure all the requests. But we regret all the difficulties in isolating each client part at the code generation phase. The name convention is essential especially when some data structures play a similar role in a different container. For instance, in the cloud application and in the mobile application we use a class called *Experiment*. It has appeared essential to review our design and rename some structures depending in which component they will be. So there will not be any type intersection between components.

After this first version of the project, it appears necessary to add a security layer based on role management. When the client part of the *AnalystMobileApp* entity is imported, all potential requests can be built, even those which do not correspond to the caller (for instance *TechnicianLabMobileApp* instead of *ExperimentModel-CloudApp*). This could be implemented by an enrichment of the request header and the use of API keys. Other security strategies can be applied by the use of other protocols such as oauth2 for instance.

Customization of the Swagger generation is not as simple as presented in their documentation. The easy changes concern the modifications on the type mapping between Swagger structure and Java classes. However, it becomes really complex to change the code factory and its output structure. Our first results are not conclusive and this was not our project objectives.

4.3 Test phase

Unlike ordinary tests that typically run on a single computer and do not interact with other machines, mobile distributed tests consist of multiple parts that work on different mobile devices and interact with each other.

4.3.1 Unit test

After the development of mobile REST services, it seems common practice to apply unit tests over them. The use of a graphical tool allows us to compare the payload of the responses and the headers. But the volume of tests makes it necessary to automatize this test phase. Several test runners exist in the REST world, but their Boolean combiners are limited and the validation of payloads involves new class development. We have selected the REST-Assured because its API development is more powerful than the others [29], particularly for writing a test case and check response status and JSON data.

As an example, we write the following test very much as we do in jUnit [Figure 7]. We also verify JSON specific elements very easily and many other details. For the tests, an embedded web server is launched, then all the tests are executed in a standard life cycle project. The test reports are saved as log files in the log directory of the embedded web server. The reading of these files allows validation of each REST service. After this complete validation of the REST API, our challenge is to test our previous scenarios [Figure 1], [Figure 2], [Figure 3].

```
package test.unit.mexp;
import static com.jayway.restassured.RestAssured.expect;
import groovyx.net.http.ContentType;
import org.junit.Before;
import org.junit.Test;
import com.jayway.restassured.RestAssured;

public class ExperimentTest {
  @Before
  public void setUp(){
     RestAssured.basePath = "http://mexp.api:8080";
  }

  @Test
  public void testGetExperiments(){
     expect().statusCode(200)
        .contentType(ContentType.JSON).when()
        .get("/mexp/v1/experiment/12345088723");
    }
}
```

Figure 7 Unit test case for REST service

4.3.2 Integration test

The integration tests are built on unit tests by combining the units of code and testing to see that the resulting combination functions correctly. But at some point we also have to make sure that the whole system works with integration tests. We have tried to use the rest-assured library as previously. This library provides a kind of DSL for testing REST APIs. We wrote our tests with JUnit and use mockito-like syntax to send a request and test the response. But this did not feel like the right solution for integration tests, because we always had to configure the requests with real Java code within the test method. We have preferred a solution where we can simply configure the request using something like annotations and just execute the test after the request is sent.

Another limit was the use of asynchronous services in MEXP. For handling asynchronous services, we always have two options: callbacks or polling. The test creation of those services is not as easy as it appears. For polling it is easier because we can loop the request code; but the results do not look right from our expertise. For callbacks we have to open a server, again within our test method as before. It seems there are no suitable solutions for asynchronous services; even REST-Assured cannot handle these services very well. We have chosen *ElasticSearch* because it is particularly convenient for implementing integration tests for distributed components based on an asynchronous message exchange pattern [30]. This framework contains a client which provides all the necessary support to embed an *ElasticSearch* node and use it to implement integration tests. It creates a node in-memory node. This can be done using the class *NodeBuilder*. It uses the default configuration of *Elasticsearch*. We override configuration entities by using the method *setSettings* of the class *NodeBuilder*. After implementing our integration test, we create data structure for these tests. The test runner executes the test suites and finally drops all data. r. All these operations are executed using the indices managed by the client API.

To implement reproducible integration tests, we need to base them on data sets that are initialized before tests and cleared. Integration tests are simply implemented using the *ElasticSearch* client instance managed by the current test suite. We notice that the component to test must be correctly designed to inject this instance in them. The following code describes a sample integration test for a component that interacts with *ElasticSearch* to build. There are three embedded *ElasticSearch* nodes that will be used by integration tests. The names are preserved as shown [Figure 6]. By the end of its execution, we observe by reading the resulting log file on the server that the causality of the interaction is preserved in comparison with the sequence diagrams.

```
package test.elasticsearch.mexp;
(...)
public class FirstPhaseTest {
  private static Node analystDevice;
  private static Node technicianLabDevice;
  private static Node experimentModelCloud;
  private static Client client;

  @BeforeClass
  public static void setUpClass() throws Exception {
     // Initialize ElasticSearch
     startupEmbeddedNode();
     // Create index and types
     createIndex("mexpIndex");
     // Initialize suite context
     IntegrationTestContext currentContext
        = IntegrationTestContext.getCurrentContext();
     currentContext.setClient(client);
  }

  @Test
  public void testLoadMetadata() throws Exception {
     IntegrationTestContext currentContext
        = IntegrationTestContext.getCurrentContext();
     Client client = currentContext.getClient();

     ElasticSearchMetadataLoader loader
        = new ElasticSearchMetadataLoader();
     loader.setClient(client);
     loader.setIndexNames(
        Arrays.asList(new String[] { "mexp" }));
     loader.initialize();

     List<TargetEntityType> types = loader.getTypes();
     Assert.assertEquals(2, types.size());
     (...)
  }
}
```

Figure 6 integration test for MEXP protocol

5. Case Study

We have organized a functional test with a set of twelve tablets from the same factory (Google's HTC-made Nexus 9) and a set of REST web services. The devices are distributed amongst the students of an undergraduate degree class. The teacher played the role of analyst and all the students played the role of technician lab. The topic of the Biology lesson was plant pigments and photosynthesis. Therefore, two lab activities are prepared: one on plant pigment chromatography, and the second one on measuring the rate of photosynthesis.

For the design of the first experiment, the students separate plant pigments using an organic solvent such as a mixture of ether and acetone. They deposit by rolling a coin over a spinach leaf about 15 times to make a heavy green line over the paper chromatography. The tablet is used to catch a set of pictures throughout the experiment. After the experiment, the students analyze their results, compute the reference front and answer a quiz.

For the design of the second experiment, the students measure the rate of electron excitation when light hits chlorophyll. Thus, they prepare one tube which contains all the solutions used in the reaction except 2,6-dichlorophenol-indophenol (DPIP). Since the tube contains chloroplasts, it will be green; the students will use this

tube to calibrate the spectrometer. The students prepare other tubes which will be experimental, and will contain either boiled or unboiled chloroplasts. Then, they expose the tubes to light and make an initial spectrophotometer reading, followed by readings at 5, 10, and 15 minutes. Again, the tablet is used to keep snapshots of this experiment. After the experiment, the students draw in the approximate shapes of the curves of the light reactions of photosynthesis over the time and answer a short quiz.

5.1 Deployment phase

The hardware architecture consists of two kinds of computer: servers which belong to a cluster and form part of a cloud; and mobile devices which are directly used by the students and a teacher. Therefore, we have highlighted three main actors: the analyst (played by the teacher), the technicians lab (played by the students) and a cloud application. The first two are deployed over mobile devices and the last component is deployed over the cloud. To sum up, there are two kinds of components: a mobile application with two facets depending on the role of the end user and a cloud application.

This deployment is automated by the use of a tool such as Apache maven. For the set of mobile devices, we iterate the same deployment task through Android Device Bridge (adb) access.

5.2 Nominal scenario

This scenario has been described in sections 3.1 and 5. All NFC-enabled devices have the ability to transfer data to each other simultaneously. For this scenario, the teacher launches the mobile application and successfully tries to connect as an analyst. He creates an experiment called "lab1". This automatically starts the Wi-Fi access point and enables his NFC tablet. Then, all other students of this experiment can quickly touch his tablet together to automatically add each one as technician lab on this experiment. Their identification on the tablet can occur after the Wi-Fi configuration or before but the role choice is essential. Then the first experiment starts on chromatography. The students take photos of their experiment. They work at their own pace and the photos become available for tagging on their mobile device. It means that REST services are exposed after taking pictures.

Concurrently, the device of the teacher displays the list of all technician labs. By clicking on one of them, the list of experiments of the given tablet is displayed, another click on the current experiment displays all the photos. The HATEOAS principle allows the teacher to navigate into all the student resources as it was its local resources. Then, after selecting one picture, he displays it on his own mobile device (Get request with “image/png’ as Accept header). He has a selection of options depending on the accuracy of the picture and what it shows. In this case, the picture is validated by the teacher and a request is sent to the student’s device to save it with additional information (id of the teacher, date, rank, etc.). This process continues until the end of the lab experiment and the stopping of the Wi-Fi access point.

The second experiment is on chromatography. The same process happens again where the teacher first creates a new experiment called “lab2”. This configures his tablet into hot spot and the students have to exchange the configuration of the Wi-Fi access through the same process. This Wi-Fi access point is visible outside the classroom but it is secured by a secret key which is unknown by the students. This limits intrusive accesses. The short duration of its validity is also a guarantee of the proper conduct of the lab activity. The students do not have access to the previous experiments and take new pictures of the second lab experiment. The teacher browses the pictures of their students to validate or not their results for the final score. By the expiration date of the Wi-Fi access, all resources are browsed.

5.3 Measurement and results

Apart from the functional tests, we wanted also to measure the limits of our distributed system. We have used Apache JMeter for stress and load-testing of REST services [31]. The main service of the MEXP API under test processes the data structures that are uploaded to a certain URL. If the upload process was successful, a URL pointing to a resource containing the processing result is returned. The resulting resource is not available immediately, processing takes a while. So polling is used to retrieve the resource once it is available. Our goal was to measure the time it takes to upload the data structure linked to a picture, process it and download the resulting picture in one test run. Running such a test with multiple users concurrently should give us a fair impression of the throughput capabilities of the system.

We have starting by writing a test plan for our previous scenario using the on-board capabilities of JMeter. Our test plan describes a series of steps JMeter will execute when run. We have defined an extension of *AbstractJavaSamplerClient* overriding *runTest(JavaSamplerContext)*. Within that method we use *HttpClient* to perform the upload and poll requests. Once the processing result is successfully retrieved by a poll request, all the header and content information is stored in an instance of *SampleResult*. The latter is returned by the overridden test sampler method for further processing by JMeter. In this way we have obtained a common JMeter result measurement functionality as shown [Figure 8].

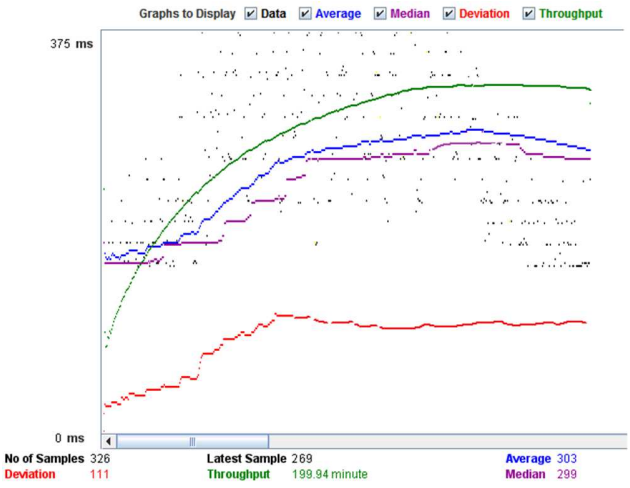

Figure 8 Stress test of MEXP Protocol

This results show that the throughput measurement is now reasonable. Because the number of partners is often less than 20, the stress limit is never achieved. Therefore, we guess our solution might be of interest for other groups such as researchers in a lab, artists in a dance rehearsal or trainers in a zoo etc.,

6. Conclusion

Our first result is our three step approach for the construction of a mobile distributed system. We have shown how essential is the construction of an open API specification with a dynamic test client. This approach is detailed through our case study on remote access over a local network. We have built a mobile distributed system and validated it in accordance with our software rules. Of course API is never going to be completely stable. Change is inevitable. What is important is how that change is managed. We

have shown that a well-documented specification can be good practice for many API lifecycles. It comes down to what is reasonable given the consumers of the MEXP API in a future version. We have shown that a security layer should be applied on the top of the MEXP protocol. We have explained how our API prototype has been validated together with the semi-automated process for the delivery. Another result shows that the calibration of the services is enough for the application domain we want to look for. This current expertise confirms the view that machine to machine (M2M) provides a response to a real need and our pragmatic approach is a direct solution.

As a future work, we suggest to enrich the MEXP protocol with new selection modes. So, with these new urls, a user could define match rules based on the injection of new tag definition on the data sources. These tags would be defined regarding the experiment tracking: event (time, period), accuracy (iteration, duration), … This will enforce the role of the experts and catch meta-information for the next queries.

7. References

[1] A. W. C. R. C. S. H. S. S. E. R. C. J. C. G. Sarah Adel Bargal, «Image-based Ear Biometric Smartphone App for Patient Identification in Field Settings.,» *VISAPP (3) 2015: 171-179,* pp. 171-179, 3 2015.

[2] K. K. K. L. G. V. &. X. S. X. Zhu, Migration to open-standard interorganizational systems: network effects, switching costs, and path dependency, 515-539.: Mis Quarterly, , 2006.

[3] R. Milner, Communication and concurrency, New York etc: Prentice hall, 1989.

[4] M. D. Hill, «What is scalability?,» *CM SIGARCH Computer Architecture News,* vol. 18, n° %14, pp. 18-21, 1990.

[5] E. &. L. G. Newcomer, Understanding SOA with Web services, Addison-Wesley, 2005.

[6] K. Petrova, «Mobile learning as a mobile business application.,» *International Journal of Innovation and Learning,* vol. 4, n° %11, pp. 1-13., 2006.

[7] J. H. Christensen, «Using RESTful web-services and cloud computing to create next generation mobile applications,» chez *the* 24th *ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, 627-634, 2009.

[8] G. &. P. L. Gehlen, «Mobile web services for peer-to-peer applications,» chez *Second IEEE Consumer Communications and Networking Conference*, 427-433, 2005.

[9] D. E. D. K. G. L. A. M. N. N. H. F. .. &. W. D. Box, «Simple object access protocol (SOAP) 1.1.,» Wiley, 2000.

[10] R. T. &. T. R. N. Fielding, «Principled design of the modern Web architecture,» *ACM Transactions on Internet* Technology, vol. 2, n° %12, pp. 115-150, 2002.

[11] P. Van de Put, Professional IOS Programming, John Wiley & Sons., 2013.

[12] J. T. T. &. B. T. Louvel, Restlet in Action: Developing RESTful Web APIs in Java, Manning Publications Co.., 2012.

[13] M. &. D. M. Aldinucci, «Skeleton-based parallel programming: Functional and parallel semantics in a single shot.,» *Computer Languages, Systems & Structures,,* pp. 179-192, 3 33 2007.

[14] J. J. I. &. B. G. Rumbaugh, Unified Modeling Language Reference Manual,, The. Pearson Higher Education., 2004.

[15] G. &. B. W. Pahl, Engineering design: a systematic approach., Springer Science & Business Media., 2013.

[16] M. L. F. &. A. D. Petychakis, «Adding Rules on Existing Hypermedia APIs,» chez *he 24th International Conference on* World *Wide Web*, 1515-1517, 2015.

[17] B. M. A. K. &. R. A. S. A. Erfianto, «Modeling context and exchange format for context-aware computing.,» chez *SCOReD 2007. 5th Student Conference*, 1-5, 2007.

[18] C. Pautasso, «RESTful web services: principles, patterns, emerging technologies. In Web Services Foundations,» chez *Web Services Foundations*, Springer New York, 2014.

[19] M. B. R. I. B. B. C. M. &. H. M. Juric, «Comparison of performance of Web services, WS-Security, RMI, and RMI–SSL.,» *Journal of Systems and Software,,* vol. 79, n° %15, pp. 689-700, 2006.

[20] K. &. B. J. Wiegers, oftware requirements, Pearson Education., 2013.

[21] I. &. S. D. C. Pyarali, «An overview of the CORBA portable object adapter,» *StandardView,* vol. 6, n° %11, pp. 30-43, 1998.

[22] V. Stirbu, «A restful architecture for adaptive and multi-device application sharing.,» chez *First International Workshop* on *RESTful Design*, 62-65, 2010.

[23] C. Kankanamge, Web services testing with soapUI., Packt Publishing Ltd., 2012.

[24] L. Lamport, « Time, clocks, and the ordering of events in a distributed system,» *Communications of the ACM,* vol. 21, n° 17, pp. 558-565, 1978.

[25] O. S. L. &. S. K. Liskin, «Teaching old services new tricks: adding HATEOAS support as an afterthought,» chez *The Second International Workshop on RESTful Design*, 3-10, 2011.

[26] J. &. A. S. Arthur, «Spring framework for rapid open source J2EE Web application development: a case study.,» chez *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Network*, 90-95, 2005.

[27] M. &. C. L. Abadi, A theory of objects, Springer Science & Business Media., 2012.

[28] W. Gédéon, OSGi and Apache Felix 3.0., Birmingham: Packt Publishing., 2010.

[29] F. (. T. w. S. B. I. P. S. B. Gutierrez, «Testing with Spring Boot. In Pro Spring Boot,» Apress, 107-120, 2016.

[30] C. &. T. Z. Gormley, Elasticsearch: The Definitive Guide., O'Reilly Media, Inc, 2015.

[31] E. H. Halili, Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites, Packt Publishing Ltd., 2008.