Micro Project

Chantal Taconet, Denis Conan and Sophie Chabridon

Revision : 545
## Contents

1 Introduction to the case study 3

2 VlibTour use cases 3
   2.1 Use cases: management of tours and POI 4
   2.2 Use cases: lobby room 6
   2.3 Use cases: management of locations 7
   2.4 Use cases: emulation of the management of visits 8

3 Extrafunctional requirements 10

4 Software Architecture 10
   4.1 The components of the system 10
   4.2 Class diagrams 12

5 Microproject progress 14
   5.1 Micro Project, labs and article links 14
   5.2 Expected POC 14
   5.3 Deliverables 14

A Priority, preconditions and postconditions of use cases 16
   A.1 Lobby room high priority use cases 16
   A.2 Tour Management high priority use cases 16
   A.3 Location Management high priority use cases 17
   A.4 Visit Emulation Management high priority use cases 17
1 Introduction to the case study

In order to develop the use of Paris VLib system for bike tourism, the city of Paris aims to propose VlibTour, a new mobile application. With this application, group of tourists such as families and friends will visit famous places of interest (POI) of Paris by bike. The participants choose a tour made of a sequence of POIs; participants take VLib to travel from place to place; and thanks to their mobile phones, participants see the map of the tour with the location of the other members of their group, find the best VLib bike stations to smoothly pick and leave VLib (without any additional time to wait for available bikes and available empty places).

The city of Paris asks for a proof of concept of the server side of the system (called the POC or the demonstrator in the sequel). The first selection of a set of eligible proposals will be made after a demonstration defense.

The modelling elements of the next sections are provided as an initial version. The model of your solution may be different. You will be asked to highlight the differences, if any, in the report and provide updated diagrams if necessary. However, your solution should not depart too much from these initial elements. And some elements will be prepared during specific labs.

Figure 1 illustrates the application user interface. A group of tourists (Joe, William and Averell) are using the system. Those tourists have selected "The unusual Paris" tour instance made up of three POI ("Musée Grévin", "Pyramide du Louvres" and "Les catacombes de Paris"). They are actually going to the last POI with different paths. In this example, one user visualises the placement of all the participants.

2 VlibTour use cases

The use cases of the application are presented separately because they address several systems. Each system is a candidate to be a component of the application that may be independently deployed. These use case diagrams are a first step in defining the API of these components.

On all the use case diagrams, the higher priority use cases are in green on the figures. These are the use cases to be implemented for the demonstrator\(^1\). Some high priority use cases are detailed in appendix A.

---

\(^1\)Other use cases will be implemented for the final version of the system (read: not in this project), by the tender selected by the city of Paris.
2.1 Use cases: management of tours and POI

Figure 2 presents the use cases of the system that manages the description of tours proposed by VlibTour. It enables tourists to select a tour; and enable tour operators to create tours.
A travel agency that acts as an operator of the system can prepare some tours, etc. on behalf of future clients.

Figure 2: Use Case Diagram — VLib — management of tours and POIs
2.2 Use cases: lobby room

Figure 3 presents the use cases of the system that manages groups of participants\(^2\). Group of tourists are created for the duration of a visit. The creation of a group is initiated by one of the tourist. The group should have already selected a tour.

![Use Case Diagram — VLib — Lobby Room](image)

We suggest to use the functionalities of this system/component in a scenario. One of the tourist, through their tourist application, enters the lobby room and asks for the creation of a group and the joining of this group. The identifier of the group is computed with the data provided by this tourist. For instance, the identifier of the group is the concatenation of the name of the tour and the name of the tourist. The role of the lobby room is to create the group communication system that is dedicated to this new group; the lobby room returns a **login** and a **password**. Afterwards, the tourist can connect to the group communication system—i.e. knowing the identifier of the group + the name of the tourist + the **login** + the **password**, the tourist application can connect to the group communication system.

The identifier of the group is transmitted to the members of the group of the visit through a communication means that is external to VLibTour, e.g. by email.

Each member of the group can then call the lobby room in order to join the group by providing the identifier of the group. The lobby room returns a **login** and a **password**. Afterwards, the group member can connect to the group communication system of the visit—i.e. knowing the identifier of the group + the name of the member + the **login** + the **password**, the tourist application can connect to the group communication system.

\(^2\)Group of participants are symbolised as lobby rooms, which explains the name of the component of the system.
2.3 Use cases: management of locations

Figure 4 presents the use cases of the system that manages the distribution of the locations. Each participant publishes his/her location. Each participant receives location of other participants in the group.

![Use Case Diagram — VLib — management of locations](image)

This use case diagram corresponds to the usage of the group communication system. We suggest to use the functionalities of this system/component in a scenario. Each tourist, through their tourist application, connects to this system (group communication system) and shares their location with the members of the group of the visit by using the use case « publish location ». Furthermore, each tourist subscribes to the group communication system to receive locations by using the use case « subscribe to location information ». The group communication system is then responsible for notifying tourist applications when there is newly published location: this is the use case « notify the location of a participant ».
2.4 Use cases: emulation of the management of visits

Figure 5 presents the use cases of the system that manages the visit of groups of tourists.

![Use Case Diagram — VLib — emulation of the management of visits](image)

We suggest to use the functionalities of this system/component in a scenario. We speak about “emulation” because we propose the validation of the solution with a scenario of tourists visiting Paris: The set of POIs (Points Of Interest) and the graph of positions of the tourists is defined. The scenario is then as follows:

- The visit of a group of tourists is defined as a graph of positions and a set of POIs. Each tourist of the group has their own path of positions from POI to POI. The visit of a tourist is then a sequence of paths, the end and the beginning of each path being a POI. The tourists of a group wait each other at every POI and take some time to visit the POI.

- Periodically, tourists obtain their current position from the visit emulation system. They can also know the position of the next POI of the visit.

- Moving forward is “stepping” in the current path, and “stepping” to the next POI when the visit of a POI is terminated. The scenario ends when the visit of the POI is terminated.

A regular usage of this server is as follows:

- **getCurrentPosition**: to know where a user is;
- **getNextPOIPosition**: to know the position of the next POI, i.e. the end of current path of a user;
• **stepInCurrentPath**: to move to the next position in the current path of a user. When a user is at the end of their path, `stepInCurrentPath` returns the same position;

• **stepsInVisit**: this call is used when a user is at their POI in order to know the position of the next POI in the visit. When a user is at the end of their visit, `stepsInVisit` returns the same position.

Use case “search for the arrival bike station” will be discussed during the labs on the REST technology.
3 Extrafunctional requirements

Additionally to the functional requirements, the City of Paris adds some extrafunctional requirements.

In the context of the project, we ask you to take in consideration those requirements for the report. In the report, we ask you to analyse the architecture of the application with regard to two quality attributes chosen among the following ones.

1. Scalability
   The system should be able to handle up to 3000 groups of tourists at a time, and handle up to $3000 \times 10$ tourists. In the future the system could also be used by other European cities.

2. Security
   The system is able to protect data against unauthorized access. Data provided by users, such as their location, are considered as private and should be used only for the purpose of the VLibTour app by authorized entities.

3. Interoperability
   The client applications can be written in any language. The APIs and interactions between the components are well defined.

4 Software Architecture

4.1 The components of the system

A possible architecture of the system is presented in Figure 6. It shows candidate components of the system.

The distribution of the components is not specified in this diagram—i.e. every component could be deployed on a separate node.

Each project may produce slightly different architectures. In particular, some components may be grouped, others may be splited. The placement of the components on client/server computer are mentioned indicatively.

Most of the connections between the components on the server side are not modelled in this diagram. This diagram does not include the technical components that could be added to meet the software quality attributes: e.g. for fault tolerance.

**Tour Management** The tour management component is used by the tour operator to prepare the available tours proposed to the tourists. It is also used by the tourists to discover available tours. This component implements the features presented in the tour management use case diagram of Figure 2.

**VisitEmulation** This component is able to follow a group of tourists for a visit, in particular it updates the next POI for the visit. The visit emulation service is a service offered by VLib for testing purpose. The methods of the API are implemented and you have to “encapsulate” it into a REST server component. This component implements the features presented in use case diagram on the management of the emulation of the visit (cf. Figure 5).
These two components are co-located if they use the same RabbitMQ broker. If so, the lobby room system creates the group communication systems on demand (one per group).

Figure 6: General architecture of the system

**Lobby Room** The lobby room is where tourists have access to the system for creating a visit and the corresponding group of participants. The main function of a lobby is to create a group for a visit, and to enable a tourist to join an existing group. Joining a group/visit means going to have access to the corresponding visit group communication system. This component implements the features presented in the group management use case diagram of Figure 3.

**Visit Group communication system** As its name indicates, the visit group communication system is the system responsible for implementing reliable multicast, that is reliable broadcast to only the participants of the group constituted for a visit. If necessary, communication could also fulfil FIFO, or causal, or even total order.

**Location management** This component is able to distribute location information among the participants of a visit. This component implements the features presented in the location management use case diagram of Figure 4.
4.2 Class diagrams

Figure 7 depicts the class diagram as seen by the tour management component.

Figure 7: Class Diagram of the Tour Management entity

Figure 8 depicts the class diagram as seen by the VisitManagement component.
A next POI belongs to the set of POIs of the tour

At all times, a group takes part to at most one tour

A tour is a sequence of POI, thus being an ordered set

arrivedAtPOI => currentLocation == gpsLocation of next POI

arrivedAtPOI => forall p in participants: p.arrivedAtPOI

Figure 8: Class Diagram of the Visit Management entity (recall that this component may exist or not in your solution)
5 Microproject progress

5.1 Micro Project, labs and article links

Some parts of the system will be designed and implemented during the labs.

- The TourManagement component will be prepared during the JavaEE lab. As a first tour sample, create "The unusual Paris" tour instance associated with a sequence of three POIs having the following attributes:
  - name: "Musée Grévin";
    description: "Musée de cire sur l’histoire de France";
    gpslocation: 48,871799 - 2,342355;
    duration: 60 minutes;
  - name: "Pyramide du Louvres";
    description: "La pyramide du Louvre est une pyramide constituée de verre et de métal, située au milieu de la cour Napoléon du musée du Louvre à Paris";
    gpslocation: 48,860959 - 2,335757;
    duration: 20 minutes;
  - name: "Les catacombes de Paris";
    description: "Labyrinthe éclairé dans une ancienne mine de calcaire avec des millions de squelettes macabres entassés";
    gpslocation: 48,833566 - 2,332416;
    duration: 60 minutes;

- The VisitEmulation component will be prepared in the REST labs.

- The LobbyRoom and the Group Communication System component will be prepared during the AMQP labs.

Some presented articles are linked to the quality attributes mentioned in Section 3.

5.2 Expected POC

It is possible that you will not have finished the whole system at the end of the project. What we ask is that some of the components are finished and tested and some of the components are interconnected. A minimum operational demonstrator should be provided. Implement and test step by step (i.e. component by component).

Remember that we want a proof of concept concerning middleware and distributed architectures, not user interfaces! We ask for simple clients, i.e., test clients with no interactions and with predefined users and messages are absolutely sufficient.

Your implementation has to be original. Some variants on the subject are encouraged.

5.3 Deliverables

The results of the micro project will be:

**Report** A small report (max. 10 pages)
- Describe your proof of concept (POC)
- Show the structural and architectural design patterns. With the material in the course “Introduction to middleware through design patterns”, identify the design patterns involved in the application. For each design pattern, draw a picture presenting it and give adequate explanations.
- Present the architecture of the POC (highlight the differences with the proposed architecture if any)
  - Highlight the design choices
  - Highlight technology choices
  - Highlight data model choices
- Provide all the necessary information to test your POC (i.e., minimum user manual, it may be just: “refers to the README”)
  - Present your architectural propositions to face two of the extrafunctional requirements (see Section 3).
  - Explain the encountered difficulties and all information you will judge necessary.
  - Feel free to present ideas not yet implemented in the proof of concept but that may help you to be selected by the city of Paris!

**Defense** An oral presentation of 15 minutes including a demonstration.

- Don’t present the microproject subject (we know it and you don’t have time for it!), presents your added value
- Discuss architecture choices to handle the chosen extra functional qualities
- Prove with a demo: explain how the demo shows the parts that have been implemented, and the integrations of the different components.
- Be prepared to answer individually to any question concerning your POC and the CSC7321 course

**Sources** All the material (report, slides of the oral presentation, program SOURCE files) has to be uploaded in moodle before the deadline and in one archive. The archive will contain a root directory with the name of the students who worked on the project (e.g., nameA-nameB.tgz archive contains nameA-nameB/ directory). Be careful:

- Do not include files that can be generated (e.g. .class);
- Use ascii7 characters only in the name of files (no accent, no white etc.);
- Include the maven pom.xml used to automate the generation of classes and also to launch the demonstration; Even, if you develop your application with an Integrated Development Environment (IDE), make sure that it can be run as a standalone application (we will test it with maven only);
- Provide a good quality code (for instance verified with checkstyle and findbugs).

Good work!
We detail below some use cases with their priority, input, output, preconditions and postconditions.

**A.1 Lobby room high priority use cases**

Create a group and join it

- input: group identifier (GID), user identifier (UID)
- output: URL to connect to the group communication system
- precondition: GID ≠ null ∧ GID ≠ empty string ∧ UID ≠ null ∧ UID ≠ empty string ∧ GID is unique ∧ UID is unique
- postcondition: GID ≠ null ∧ GID ≠ empty string ∧ UID is unique ∧ UID ≠ null ∧ UID ≠ empty string ∧ UID in the group ∧ URL ≠ null ∧ URL ≠ empty URL

Concerning the uniqueness aspects of identifiers, for the sake of simplicity, we propose to not manage it, but to consider that the identifiers that are provided by the client applications are unique. Of course, client applications can make use of the facilities provided by middleware technologies when they manage the uniqueness of identifiers: for instance, the annotation `@GeneratedValue` with the attribute `strategy=GenerationType.AUTO` of JAVA EE entity beans, or the class `java.util.UUID` with RFC 4122. In addition, for those who wish to add access control through the use of passwords, you may be interested in the JAVA library Passay at [http://www.passay.org/](http://www.passay.org/) for generating passwords. Please, do not hesitate to point to suggestions or solutions that could be interesting in the context of this micro-projects. As a consequence, the implementation that you propose may not check that identifiers are unique; this does not avoid the management of possible conflicts in resource instantiations such as instanciating objects that are semantically equal.

Join a group

- input: GID, UID
- output: URL to connect to the group communication system
- precondition: GID does exist ∧ UID ≠ null ∧ UID ≠ empty string ∧ UID not already in the group
- postcondition: UID ≠ null ∧ UID ≠ empty string ∧ UID is unique ∧ UID in the group ∧ URL ≠ null ∧ URL ≠ empty URL

**A.2 Tour Management high priority use cases**

Select a tour

- input: GID, UID, identifier of the tour (TID)
– output:
– precondition: GID does exist ∧ UID in the group ∧ TID does exist
– postcondition: The tour of the group GID is TID

A.3 Location Management high priority use cases

Subscribe to location information
– input: GID, UID, notification handler
– output: subscription identifier (SID)
– precondition: GID does exist ∧ UID in the group ∧ UID not already registered to location notification
– postcondition: SID associated with notification handler for the corresponding GID and UID

Publish location
– input: GID, UID, current position
– output:
– precondition: GID does exist ∧ UID in the group ∧ current POI ≠ valid GPS position
– postcondition:

Notify the location of a participant
– input: notification handler of the UID, GID and UID of the participant whose location is notified, position
– output:
– precondition: valid notification handler³ ∧ GID does exist ∧ UID in the group ∧ GID of participant to be notified = GID of the participant whose location is being notified ∧ UID of participant to be notified ≠ UID of the participant whose location is being notified
– postcondition:

A.4 Visit Emulation Management high priority use cases

The specification of these use cases is not presented in this document, but the implementation is provided. The API is composed of the public methods of the class VisitEmulationServer.

³The validity of a notification handler is system/middleware dependent.