POLITECNICO DI TORINO

III FACOLTÀ DI INGEGNERIA

UNDERGRADUATE THESIS

Data storing with an external web database for e-learning activities in Second Life

AUTHORS: Cristian Moral Martos
Albert Bellonch Llargués

TUTOR: Andrea Giuseppe Bottino

April 2011

Abstract

To this day, technology, and more specifically Internet, is present in almost all the areas of our life, either in the professional or the personal sphere. Nowadays, one of most global and important example is the use of the social networks, like Facebook or Twitter, that have become part of everyday life of millions of people around the world, not only to share photos, videos, messages or events, but also like an incredibly prized source of information.

During these last years, this technological progress has also made appearance in the field of the education. Indeed virtual learning, also called e-learning, is becoming more and more popular and common not only in specific courses of distance learning, but also as support to primary and secondary school and obviously in university and college. This type of learning is characterized by the use of multimedia means that include photos, videos, sound and interactive activities that can be provided by many different platforms.

In our case, we have chosen as multimedia platform the virtual network Second Life that owns all these features, bringing them to their ultimate form as it looks like a 3D game. Indeed, our project is part of greater project of the London Metropolitan University that aims at giving some of their courses, or at least part of them, through this platform, because of its visual appeal, its flexibility, its scalability, its simplicity and its easy access, beside it's free.

This document goes through different abstraction levels to start describing the most general issues involved in this field, i.e. Second Life and e-learning, until the concepts can be applied to a case study that fits perfectly within the framework of the e-learning through Second Life, and riches it by implementing some features that are basic but essential.

Contents

List of Illustrations

List of Diagrams and Tables

Chapter 1

Introduction

1.1 Objectives

Our objective is to create a system able to support e-learning activities in Second Life. This project conforms our final thesis and has been possible thanks to the collaboration with Alan Hudson, professor of the London Metropolitan University and one of the leaders of Second Life at it, carrying out many projects in the mentioned university. Our collaboration is centered in the use of Second Life for learning purposes through e-learning activities. Even if the own platform of Second Life offers mostly all the requirements needed to carry out e-learning activities, as it is a real time and multimedia platform, the problem stems from the fact that Second Life doesn't give the possibility of storing data about the actions done or the activities carried out by the users. This supposes a big problem to develop e-learning activities, as there is no way to recover some important information even from the users and from the activities, like if a user has the permission to assist an activity, which users have assisted to an event or the result of an evaluation of the knowledge acquired by the user, between others.

That's why our system aims at covering these gaps of Second Life related to the storing, analysis and management of data. As Second Life doesn't offer any structure to store data, we have used an external web dataserver where data will be stored and have set a communication channel between Second Life and it. Moreover this, we have designed a user-friendly web application that eases the management and the access to the stored data to any user, as it is really intuitive.

To accomplish these purposes, we have tried to apply all knowledge acquired during our career to create a scalable, durable, easy-to-use, easy-to-manage and easy-to-expand system, according to the requirements given by Alan Hudson but also providing ourselves some other ideas or improvements.

1.2 Structure

This document will be divided mostly into two parts. The first one is the one related to the investigation we have done to center the topic and give some sort of background of it. For this, we have searched, analyzed and summarized all the information we have considered important for the project. This information deals with 2 main issues: Second Life and e-Learning. The first one aims at presenting Second Life, its main features and functionalities in order to justify its suitability as platform for e-learning activities. For this, it's been essential to introduce the concept of e-learning and its features to check if those can be satisfied by Second Life, at least most of them.

The second main part of this document deals with our project, which is presented as an improvement that added to second Life, will allow using it as a platform to carry out e-learning activities. This part aims at explaining as much as possible why we have designed our system as we have done, how we have done it, which problems have we met and how have we solved them. The solution presented can be also divided into two parts, as the same project consists of two parts: Client side and Server side. The first one refers to the application which is directly visible and used by the final user, that's to say the infrastructures and coding in Second Life, and the web application to access and manage the data stored in the database. The other issue is the one related with the architecture, design and internal implementation of the server and the database, which are not visible to the users, but are responsible of the most of the logic of the system and its functionality.

The organization of this document is also a reflection of the working path we have followed. So, at the beginning, and according to the requirements wanted from Alan Hudson, we have had to learn something about Second Life, its philosophy, functionalities and features. After this we have looked for some information about e-learning and its features and needs. All of this helped us to design a system solving the problem raised fitting the solution to the requirements and the information collected as much as possible.

Chapter 2

Second Life

2.1 Description

Second Life is a virtual 3D world developed by Linden Labs, with Philip Rosedale at the head, launched on June 23rd 2003, which is accessible via Internet by anyone all over the world since it's free and multilingual. As data of use, during the last quarter of 2010 the users have spent around 105 million hours in Second Life, there was an average of repeat logins of around 795 thousands per month and its virtual surface rose up to 2080 km^2. (1)

This virtual world has characteristics of both social media and a massively multiplayer online game, even if the concepts of winner and looser don't apply in this case.

For using it, users only have to download and install a client software called Viewer. This program enables the users to interact with the other users of Second Life. For instance, users can explore, meet other users, socialize, shop, participate in individual or group activities, work, create and trade virtual objects, buildings or services with one another, or travel throughout the virtual world. As its name says, Second Life tries to simulate as much as possible the real life, and that's why it becomes essential to have money in it as base element to exchange for objects or services.

2.2 Viewer

Second Life is based on a three-dimensional modeling which base is a set of simple geometric shapes, which can be combined by the user to create virtual objects. Moreover this, some functionality can be added to these virtual objects by using some scripts, written in Linden Scripting Language (LSL). Both objects and its functionality become visual for users thanks to the Viewer.

2.3 Users

In Second Life, we can distinguish the physical users from the virtual users. On the one hand, we have the physical users, called Residents, who are the users of Second Life. On the other hand, there are the avatars, which are the virtual characters allowing the Residents to interact each other into Second Life.

2.3.1 Appearance

One of the most important features related to avatars is that their appearance can be fully customizable by the residents. These ones can not only choose if they want their avatar to look like humans, animals, vegetables, mineral or a combination of them, but also personalize any physical trait, as eye's color, hair's color or type of physical constitution among others. This allows the users to decide if their avatar will resemble themselves as they are in real life or, on the contrary, will look like someone absolutely different. (2) Even if a single resident may be using only one avatar at a time, the appearance of this one can be changed as many times as the resident wishes. In fact, it's been demonstrated that avatar appearance was very important to users, evolving over time and that's why users often change or update their avatar appearance by purchasing items like clothes, hair or body shapes in Second Life stores. (3) The choice of the appearance is directly related to the identity that people wish to present. This way, we can identify four types of users according on user's aims: the realistics, the ones who intended to show their ideal self or their fantasized self and the role-players.

2.3.2 Communication

Another essential feature related to avatars in Second Life is communication. Indeed avatars can communicate via one-to-one chat, group chat, global instant messaging (known as Instant Messaging, IM) or voice. Chatting is used for localized public conversations between two or more avatars. This means that the conversations are visible to any avatar within a given distance in the virtual world, more specifically within 20 meters for normal chat, 10 meters if the avatar whispers and 100 meters if it shouts. (4) In these cases, the avatars must communicate through the channel 0, defined as the *"Public channel*" which is directly listened by all the avatars within the defined distance. Nevertheless, avatars can also speak through other channels. For this, in the chat box, the resident must only write */nºchannel* just before the text. This type of communications is directed to objects as it must be a script which listens in this channel, as all the channels different from 0 are only listenable via script.

There is another option that allows communicating to all the objects present in a region. This type of communication is only directed to objects as the channel 0 cannot be used, so every object in the region has to have a script listening to the same channel in order to listen the communication.

IMs are used for private conversations, either between two avatars, or among the members of a group, or even between objects and avatars. Unlike chatting, IM communication doesn't depend on the participants being within a certain distance of each other. In case of using voice chatting, you can also chat with everyone within 60 meters or have a private oral conversation with another avatar. (5)

Another type of communication is given by Understanding Dialogs, which allows a direct interaction between the message and the avatar, as the first one gives a message and some options and the second one have to choose one of them.

Finally, the last type of communication focuses on the prims that form an object. As we will see later, these prims are linked one to each other, and between them to send *link messages* that are only listenable by the prims that are linked into the same object.

2.3.3 Actions

The actions that can be done by the avatars in second Life can be divided into two groups. The first one tries to reproduce the most of the movements that can be done by humans in real life, that’s to say walk, run, touch, take and drop objects, sit, unsit, jump and duck. With these set of movements, the avatars are allowed to move in the virtual world as a human would do in a real world. The second set of movements is the one which groups the surrealistic actions that cannot be done by humans. For instance, avatars can fly by themselves or teleport to other places. The reason of adding these two fantastic movements is to increase to the maximum the mobility of the avatars easing as much as possible the access and knowledge of any place in Second Life by everyone.

2.3.4 SLurls

In case of teleport, Second Life has established virtual addresses for every place in the platform. These addresses, called SLurls, look like the URLs of the webpages and behave as them, because they act as a hyperlink that allows the avatars to teleport from one place to another directly. These SLurls carry themselves both the name and the coordinates (3, as the virtual world is three-dimensional) of the region. An example of SLurl is http://maps.secondlife.com/secondlife/Here/128/128/44, which points at the region *Here* with the coordinates *(128, 128 ,44)*. The use of these SLurls is absolutely intuitive, as provided that the viewer is opened and the resident is logged, it's enough to click on the SLurl to teleport the avatar to the destination.

2.4 Objects

As the entire world of Second Life is created by its residents, the creation of objects is very common in Second Life. In fact all the residents can create objects with structures that go from the most simple to the most complex.

2.4.1 Prims

Every single object created in Second Life is based on some simple structures called prims (short for primitives). These prims are objects with basic and simple 15 shapes, like a cube, a triangle, a pyramid, a sphere, a cone or a cylinder. (6)

Illustration 1. Some samples of prim shapes

All of them are fully customizable not only in terms of measure, but also in orientation (both vertical and horizontal), color, transparency and so on. In fact, by default all the prims are plywood-looking, as we can see in Illustration 1, but even this is customizable as the resident can change the material a prim is made of, choosing one of given by default or acquiring a new one from other avatar r from a shop in Second Life.

2.4.2 Linking

In order to create more complex objects, Second life allows to link different links one to each other so it looks like a single object. The only restriction regarding the linking of prims is that it is not permitted to link more than 255 prims. But the advantage of linking prims is not only the look but also that the linked object behaves as only one, which is very useful because they can be moved in unison and because it's possible to associate different script to each prim that can easily communicate with each in order to reach the desired effect or action. In fact residents can link not only some prims to form a single object but also they can link different objects each other creating a single structure that gives, as for prims, a single behavior for all of them and simple way to communicate each other (e.g. a door that opens when a button is clicked).

Moreover, there's another way to obtain a desired shape. When a resident wishes to create an object with an organic shape which is too difficult to create linking basic prims, textures can be created outside of Second Life using a 3D modeling software and after uploading the obtained texture to Second Life in order to apply it to a prim. This type of prims are called sculpties. (6)

2.4.3 Sandbox

Although prims are free in Second Life, resident cannot build object wherever they want. Indeed, there are two ways to build objects. Firstly, there are some parcels of lands, called sandbox, where avatars can practice and build objects for free. The only restriction of these places cannot be created indefinitely as they are "cleaned" every few hours, that's to say that every that has been created on the sandbox is clear and returned to the creator. In fact, avatars can store as many objects as they want, carrying virtually them, in their personal inventory. The second option is to own a land or to have permission from the owner of the land. In this there's no restriction in terms of time.

2.4.4 Permissions

Another important feature of objects in Second Life is their protection. In effect, every object has some individual permission that defines what can do or not everyone in the virtual world. The first type of permission is the one of 'Move', which sets who's allowed to move a created object through the world. The second is the 'Modify' permission, defining who can change or edit the object. Associated to this permissions, there's the 'Next Owner Modify', in whose absence any avatar, except the current one, no one can create derivative works. (7)After this, there's a 'Copy' permission to allow or not someone to copy the object. In case of the 'Transfer' permission, it gives an avatar right to give or sell an object to another avatar. If an item is not transferable, the owner cannot sell or give it away. The transfer permission only applies to the owner as he's the only one who can initiate a transfer. Finally there are other permissions like ‘Delete’ and ‘Take and Return’ that are called pseudo-permissions because they depend on inherited permissions. For example, an avatar can never delete an object unless he’s the owner. Some of these permissions, like ‘Move’, ‘Modify’, ‘Copy’ or ‘Transfer’ can be also set to groups allowing that all the members of the group inherit the permissions of it. (8)

2.5 Economy

Since Second Life intend to simulate the real life, it’s not a big surprise that this virtual world has its own economy and currency, called Linden Dollar (L$). (9) The economy in Second Life is based on the trading of goods and services between residents through their avatars. As every resident is able to create objects that can be decorative or functional, it only has to be another resident interested in them to begin the transaction. This way, like in real life, we have sellers and buyers of many types of virtual goods (e.g. buildings, vehicles, devices of all kinds, animations, clothing, skin, hair, jewelry, flora and fauna, and works of art) or virtual services (e.g. facilities to dance, paint or sit).

Apart of these activities, residents can also obtain L$ thanks to a weekly stipend given by Linden Labs. Residents have this stipend if they a premium account, which is not free, or if they have registered in Second Life in the very beginning of Second Life.

Like all the real currency, Linden Dollar has a currency exchange, that’s to say it’s possible to buy and sell Linden Dollars, paying or earning respectively with the local currency of the resident. For this activity, Linden Lab has created a platform called LindenX (10) which defines the rates of exchange, which are fluctuant, and allows performing the above transactions. Nevertheless, Linden Dollars don’t’ have an intrinsic value as a form of currency (11), namely it doesn’t have a real value and consequently, as indicated in the Terms of Use, situations like the loss of Linden Dollar from the database or an alleged bankruptcy would not imply, in any case, a compensation for the residents. (12)

2.6 Policy of use

Although Second Life seems to reproduce in every way our real society, the fact is that in terms of laws, politics and government the lack is almost complete. Indeed, Second Life residents mostly do not have a government. In part this is enabled by the fact that there is also no physical damage, and in principle no possible theft of property (excluding virtual content), nor is there war on a large scale, other than between military groups or other role players. Thus, many of the functions of government are not required.

At this point, any potential conflict is solved depending on its seriousness. At the lowest level, the owner of the land is the one who has the last word, and every resident can set the rules he or she wants in his owned parcel and can eject or ban any resident he or she wishes to, with or without cause. Afterwards, if the problem reaches larger dimensions, the ultimate authority is Linden Lab, which is the real owner of Second Life and all his lands and contents. In these cases are applied the rules indicated in the Terms of Use.

2.7 Uses and Applications

Due to the flexibility of the platform and to his similitude with our real society, practically every single activity can be done in Second Life.

2.7.1 Marketing, Advertising and Trade

Probably one of the most common and popular area which has a representation in Second Life is the one of Business, Marketing, Advertising and Trade in general. Second Life offers a great, cheap, visual, easy-to-access and worldwide platform to test, evaluate and sell real products or services. In fact, it's really to fins in Second Life entire lands owned and managed by big companies from all around the world and from all kinds of areas, going from cars to sportswear to software or even hostels. (13)

Illustration 2. Toyota land in Second Life

Illustration 3. Adidas land in Second Life

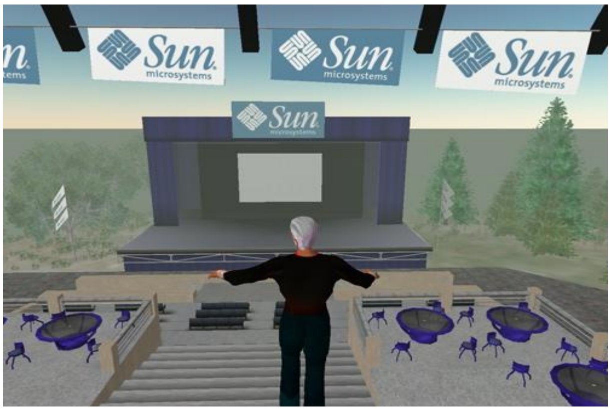


Illustration 4. Sun Microsystems land in Second Life

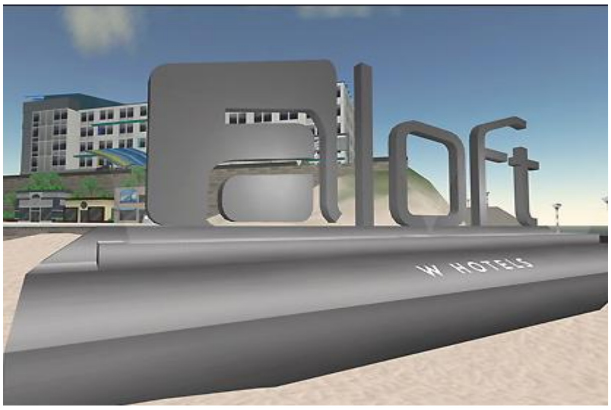


Illustration 5. Starwood Hotel in Second Life

2.7.2 Information means

But the profit-making companies are the only ones that are represented in second Life. Another area present in second Life is the one related to the information. Televisions, Radios, Newspapers and News Agencies have their own lands where they not only advertise themselves and their programs, but they even retransmit some of them via this virtual world in streaming.

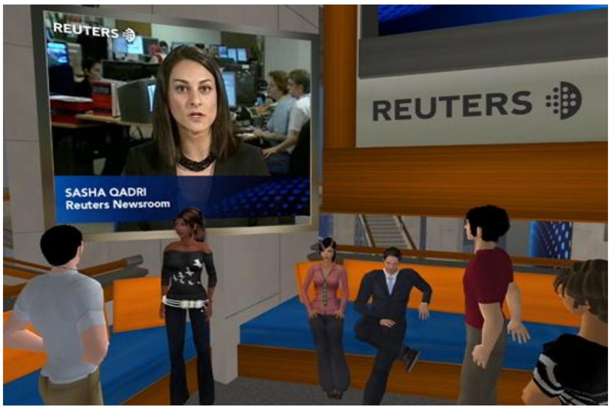


Illustration 6. Reuters land in Second Life

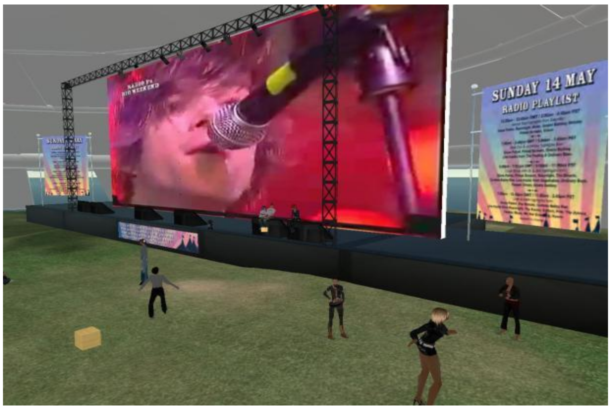


Illustration 7. Retransmission of the BBC in Second Life

2.7.3 Science

Besides this, Second Life is a very useful tool in Science field, as it is used for Scientific Research, collaboration, and data visualization. (14)

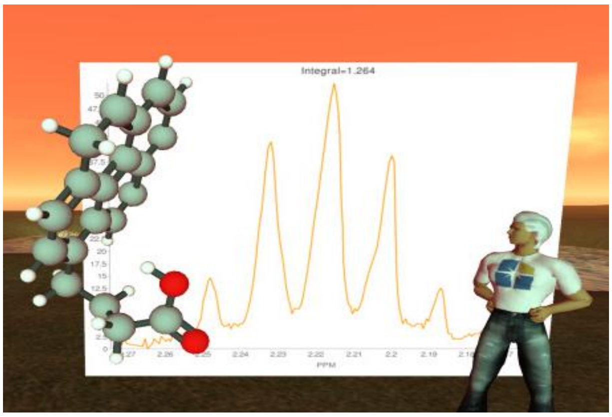


Illustration 8. 1-Pyrenebutanoic acid together with its NMR spectrum in Second Life

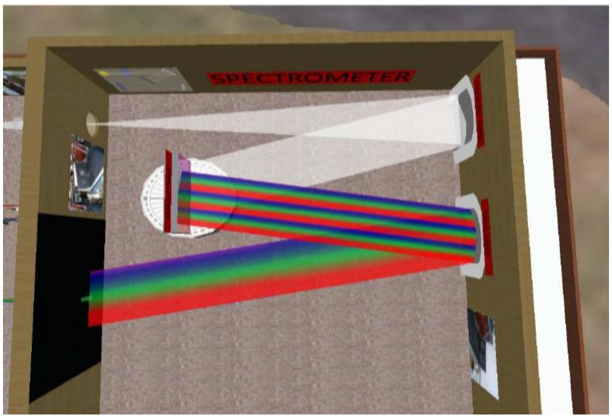


Illustration 9.Glow Discharge Spectrometer in Second Life

2.7.4 Leisure

By the way, we can’t forget that one of the first and most important goals of Second Life is entertain and amuse the members of this social virtual network. That’s why there also lots of lands dedicated to the leisure offering games, dance, art, sports, etc.

Illustration 10. Lego land in Second Life

Illustration 11. Avatars playing Wii in Second Life

Illustration 12. Major League Baseball stadium in Second Life

2.7.5 Religion

Obviously, religion is not an exception and is also present in Second Life. Christian, Muslims, Jewish and so on have created churches, mosques and temples in order to allow their followers to practice their faith virtually.

Illustration 13. Mosque in Second Life

Illustration 14. Church in Second Life

2.7.6 Diplomacy

After this, maybe one of the most surprising presence is the one of the diplomatic representation. Indeed there are some countries, like The Maldives, Sweden or Estonia which have used or are using Second Life to open there virtual embassies in order to give information and support to their citizen and also to promote their culture and country. But this is not the only given application to Second Life by politicians. Second Life has become in some cases one the means used to carry out political campaigns or political meetings. (15)

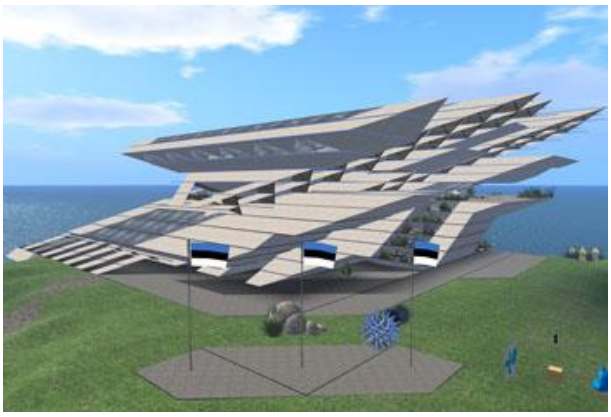


Illustration 15. Estonia embassy in Second Life

2.7.7 Education

Finally, the field which is growing faster and has a good future perspective is education, through e-learning. In fact, Second Life has all the means and infrastructures required to teach, learn, assist to seminars or debates, work in group… and all of this with the support of the new technologies and the ease and comfort of being to do so at any place with a computer connected to Internet. Currently, lots of university and colleges have or are planning to have virtual campuses in Second Life where they could give information to prospective students, teach courses or lessons virtually, perform practical works using objects created specifically for this or simply meet with other students, teachers or professionals to discuss about a topic, a homework or a project. Moreover, the fact that Second Life is free is a key point as neither the education institutions (except maybe to buy a parcel to set the campus) nor the students have to invest any money, and so there are no economic or budgetary barriers to carry out as many projects as needed to cover as many fields as possible. (16)

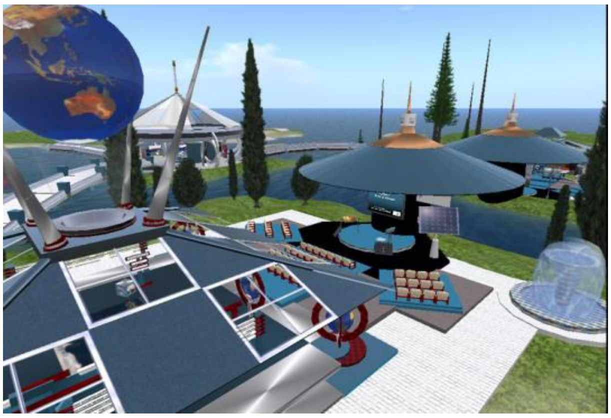


Illustration 16. Solar Home of the Polytechnic University of Madrid in Second Life

Chapter 3

E-Learning

3.1 Description

As its name says, e-learning, abbreviation for Electronic Learning, groups any form of teaching and learning which is supported by electronic means, that's to say by using systems of communication and information like computers connected to internal or external networks but also media means like, for instance, CDs, DVDs or satellite TV.

Essentially, by e-learning we mean the use of a computer and a network which allow transferring skills and knowledge. For this, typically are used some media (image, animation, streaming video and audio.) to simplify and clarify the process of teaching and learning. Another feature is that the e-learning can be self-paced, and so the student organizes the time according to his availability or to his/her need to understand the contents. In other cases, the study is led by an instructor which imparts his lesson through a virtual classroom.

Afterwards, we are going to study the most significant characteristics of e-learning describing its differences with traditional teaching modalities, and analyzing its most general advantages and disadvantages. As we will see, the strong point of e-learning does not lie in its technical dimension, like the platform used, but rather in the control and meaning assigned to a series of variables (17) such as:

- Presentation of the contents
- Role of the teacher
- Role of the students
- Communications between teacher and students (both synchronous and asynchronous)
- Tools used and their application in the teaching process
- Didactic strategies used
- Attention paid to organizational features
- e-activities provided…

3.2 Distinctive features of e-learning

Even if there are many ways to perform e-learning, all of them have some aspects that common. These identify the model and set down a sort of guidelines to establish an e-learning system. Some of the features that must be taken into account are:

- Learning lead by computer
- Use of web browsers to access the information
- Protocols of connection between professor and students, that can be separated in time and/or space.
- Use of different tools for communications, both asynchronous and synchronous
- Use of multimedia and digital materials
- Protocols to store, maintain and manage all the materials in a web server
- Learning must be flexible
- Learning must be supported by tutorials of the professors
- Combine individual study with collaborative activities
- Learning must be eminently interactive in all senses
- Use of protocols TCP and HTTP to ease to the students the access to the learning materials or to the resources.

3.3 E-learning vs. Traditional teaching

Even if the objective of both is the same, the steps to achieve are not always the same. In fact, each model is based on different procedures and skills, and each one has its own philosophy that guides the learning to one method or another. We can identify 8 main aspects that illustrate the differences and similarities between the two models.

3.3.1 Organizational model

First of all, e-learning is based on a different organizational model. On the one hand, e-learning allows the individual learning, as all the resources al electronically and therefore accessible individually by every student. This leads directly to another feature of e-learning, which is the unlimited number of students that can participate in a course, due to the unlimited access capability to the resources.

3.3.2 Learning model

After this, we can talk about the learning model, since e-learning inherently involves an active and personal process of construction of knowledge. Indeed, every student has access to the didactic material and the professor as in traditional education, but the flow of the learning process changes. In fact, the student has to study all the lessons by himself with the support of the material, and the professor becomes only a physical and human support to clarify or expand what the student has learned by himself previously, but stops being the one who leads the lessons and explains all its contents.

3.3.3 Material

This new model is achievable thanks to the material associated to e-learning. This material is composed, as in traditional teaching, of written notes about the lessons, but also allows the use of multimedia in the form of audios, photos, videos… This way, the material can be combined as wanted and becomes highly dynamic and interactive, allowing the student to understand better the contents combining theory and practice at each point of the process. Moreover, as all the material is in electronic form, it is accessible everywhere and every time the student wants or needs.

3.3.4 Rhythm

This flexibility in the access to the material is the base of another difference compared to traditional learning related to the rhythm of study. In fact, an e-learning student can study the syllabus of the course at his/her own pace. This way, the students are allowed to spend more time in the topics they find more difficult, and to speed on those that they understand easier. This solves one of the most important problems that can be found in the traditional teaching: the difference of federate of the students. This difference is bad both for the students that advance faster and those who do it slower. In the first case, the students don't use all their skills because the rhythm of the lessons must be lower in order to allow the slower students to follow the lessons. As a result, these students don't improve as much as they could and so they waste of their capabilities and see their possibilities limited. On the other hand, student that need more time to understand the contents must carry a higher rate than normal in order to be allowed to follow the lessons. This causes a great stress on them that prevents them from understanding well the contents or, at worst, causes they fail to keep the pace of the class being inevitably doomed to failure. That's why this model of learning is called *just-in-time learning*.

3.3.5 Flexibility

This temporal flexibility is not the only one provided by e-learning. The spatial flexibility is another important feature to take into account e-learning allows the students to consult the material and advance almost everywhere, as the only need is, in most cases, a computer with connection to the Internet. This makes a big difference to the traditional education because the student are no longer forced to move to the study centers, resulting in savings of time which can be used for studying. In some cases, for examples for students that are working, this temporal and spatial flexibility is the one that allows them to continue their studies because otherwise it would impossible for them to reconcile their work and studies.

3.3.6 Communication

Another point to consider is the communication. In the traditional teaching, the communication is linear and almost always between the student and the professor. However, in e-learning, the communication is much more open and interactive, as students communicate also between them, for example through chats or forums, where each one expose its opinions allowing to take place a debate of ideas between the students and the professor. Moreover, the use of the technology allows using not only synchronous communication (talking in real time with the professor or other students) but also asynchronous, which allows students raise their doubts at any moment to everybody so that the response is useful to all.

3.3.7 Resources

Nevertheless, owing to the relative novelty of this new way of teaching, most societies are not yet ready, in terms of structural and organizational resources, to make this learning model the main one in all the levels, or at least in some of them. However, this lack is being reduced more and more over time. As an example, in the United States, 51% of the rural school districts and around 72% of the suburban and urban school districts are currently offering on-line courses, which results in around a quarter of the secondary student population of the whole country is using, nowadays, this model of learning. (18)

3.3.8 Experience

At last, we can point out that the experience in this model of learning is far away from the experience gained over the last decades. This hinders its larger implantation because it implies that both the institutions and the teachers must update their methodology, resources and planning and improve their skill to adapt themselves to the new technologies. Besides, students must also change their methodology of study, which can cause also reluctance from them. In many cases, ones or others are reluctant to those changes, what slows down the process of spreading and implantation.

Below we sum up the presented features and compare them to those of the traditional teaching:

Feature	e-Learning	Traditional teaching
Organizational Model	-One application can be used by many students at the same time. -Usually done individually, but without giving up the accomplishment of group activities.	-The number of student is restricted as the resources are limited (classrooms, blackboards…). -The teaching is always carried out in groups.
Learning Model	-The knowledge is acquired in an active way, as the student is who leads the learning process and so develop his/her own ideas from the beginning.	- The knowledge is acquired in an passive way, as the professor is who leads the learning process and the student listens and then is supposed to develop his/her own ideas.
Material	-Use of multimedia material: more dynamic, more attractive and more interactive.	-Normally based on printed documentation presented by the professor.
Rhythm	-Each student can study at his/her own pace. -Students study when and what they want or need (*just-in-time* and *just-for-me* training).	-The time and manner of study are marked by the teacher. -All the students must follow at the same pace, which is defined by the professor.

Flexibility	-Students study when they want. -Students can study wherever they want.	-Temporal rigidity. -Developed in a fixed time and in specific classrooms.
Communication	-Open communications (broadcast). -Share of knowledge. -Synchronous and asynchronous communication. -Facilitates the participation of all the students.	-Linear communications. -Communication only between the students and the professor. -Only synchronous communications.
Resources	-In many cases, there are no the necessary structural and organizational resources to carry it out.	-There lots of organizational and structural resources, so its implantation is quite easy.
Experience	-New model: little experience. -Reluctance of institutions, teachers and students.	-Model used for decades: very much experience. -Known by teachers, students and institutions.

Table 1. Differences between e-learning and traditional teaching

3.4 Advantages and disadvantages

As all the models, e-teaching present both advantages and disadvantages. Many of the advantages and disadvantages associated to e-learning stem more from subjective popular belief than from objective investigations. Some of these advantages are:

- Makes available for the student a great amount of information.
- Eases the update of the information and of the contents.
- The information is more flexible, regardless of time and space where are located the student and the professor.
- Allows the delocalization of the knowledge.
- Increases and promote the autonomy of the students.
- Propitiate a just-in-time and just-for-me education.
- Offers several tools for communication both synchronous and asynchronous, both for students and professors.

- Allows using multimedia in the learning.
- Promotes a learning individual and in group at the same time.
- Promotes the interactivity with information, with the teacher and with the other students.
- Allows to store in the data servers what, when and how have done the students.
- Save money and time because students and professors don’t have to move.

On the other hand, the disadvantages are listed below:

- The professor has to spend more time to prepare all the materials, which must be totally self-contained and absolutely clear and concise.
- Students and professors must have and minimum level of technological knowledge.
- Students must have ability to study by themselves.
- The quality of the education must decrease if the ratio professor-student is not adequate.
- Students must spend more time and effort to achieve the goals.
- Has to face the reluctance from institutions, professors and students to change the model.
- Impose loneliness and lack of physical references.
- Depends on a computer with connection to the Internet, and the speed of this that must be fast.
- Can appear problems of security and authentication of the student.

Obviously, some of these disadvantages would disappear as experience is gained.

Chapter 4

Enabling e-learning activities in Second Life

According to the information exposed before, Second Life seems to be a perfect platform to carry out activities of e-learning, and that's why our case of use centers on it.

Inherently, Second Life provides all the features needed in e-learning activities, as it is a multimedia platform. However, there are other features, rather related to management and control issues that are not implemented. Our objective is to fill some these gaps to allow teachers and managers to carry out activities of e-learning. Some of these shortcomings are related both with the students and the activities. On the one hand, it's necessary to be able to identify the students, to know when they have assisted to an activity or to test their knowledge to evaluate their progress. All these features carry the need of storing data. As Second Life doesn't offer any mechanism to store data, we have implemented an external web database with which communicate our structures of Second Life and which stores those data. Moreover this, this web dataserver has been improved providing some features as creating the e-learning activities that will be associates to a room in a defined moment in Second Life, consultation of data stored by users through a webpage with user interface… and all of this with a login system.

4.1 Starting the project

The first step we have done to start our project is related with the configuration and setting-up of the tools. Later, in point 4.6 we detail these tools, their functions and why we have chose them.

As we will mention later, we have used a Subversion control system in order to track changes and set up a common repository for both developers, which is moreover used directly from Eclipse. In fact downloading the code of our project is as easy as getting the Subversion files from the root, as we detailed below:

1. Click File > Import.

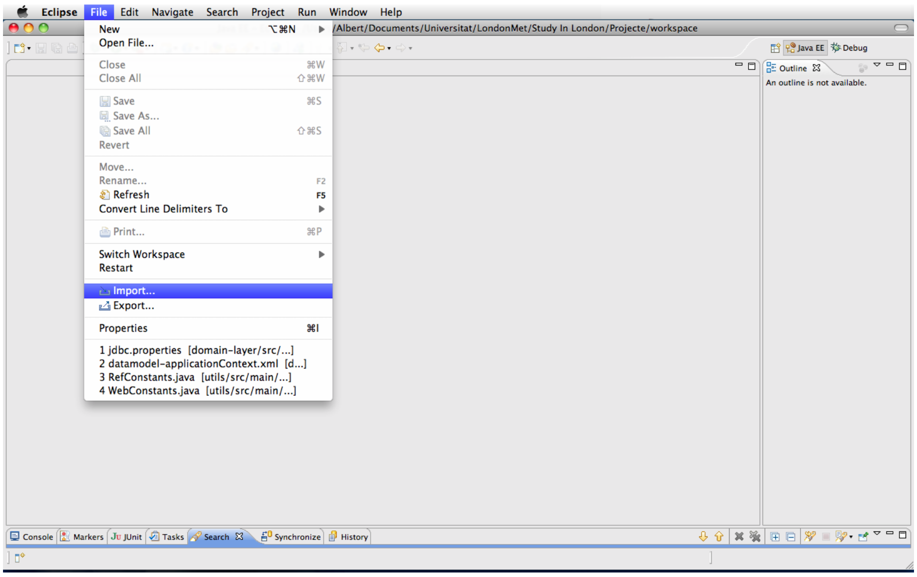
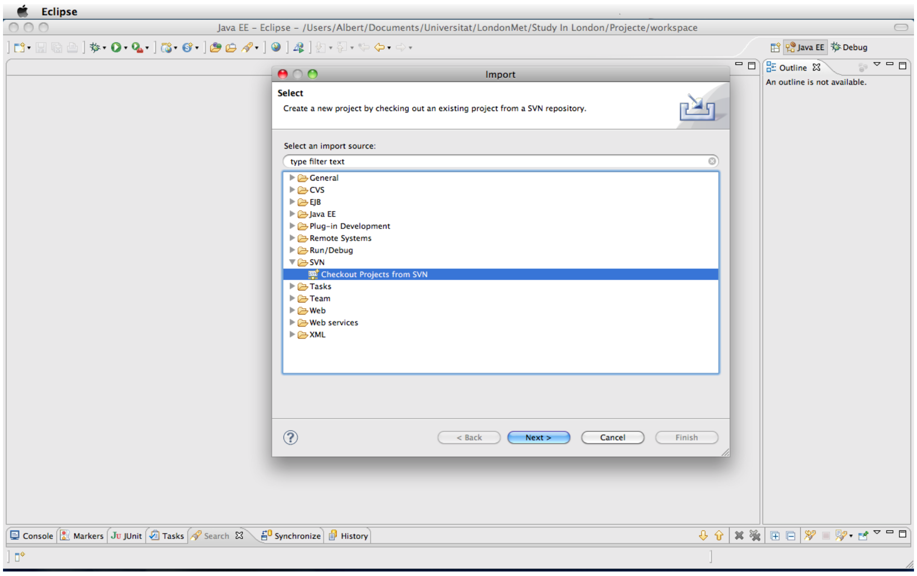

Eclipse
File
Edit
Navigate
Search
Project
Run
Window
Help
/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace
New ⌥⌘N
Open File...
Close ⌘W
Close All ⇧⌘W
Save ⌘S
Save As...
Save All ⇧⌘S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... ⌘P
Switch Workspace
Restart
Import...
Export...
Properties ⌘I
1 jdbc.properties [domain-layer/src/...]
2 datamodel-applicationContext.xml [d...]
3 RefConstants.java [utils/src/main/...]
4 WebConstants.java [utils/src/main/...]
Java EE
Debug
Outline
An outline is not available.
Console
Markers
JUnit
Tasks
Search
Synchronize
History

Illustration 17. Starting the project: Step 1

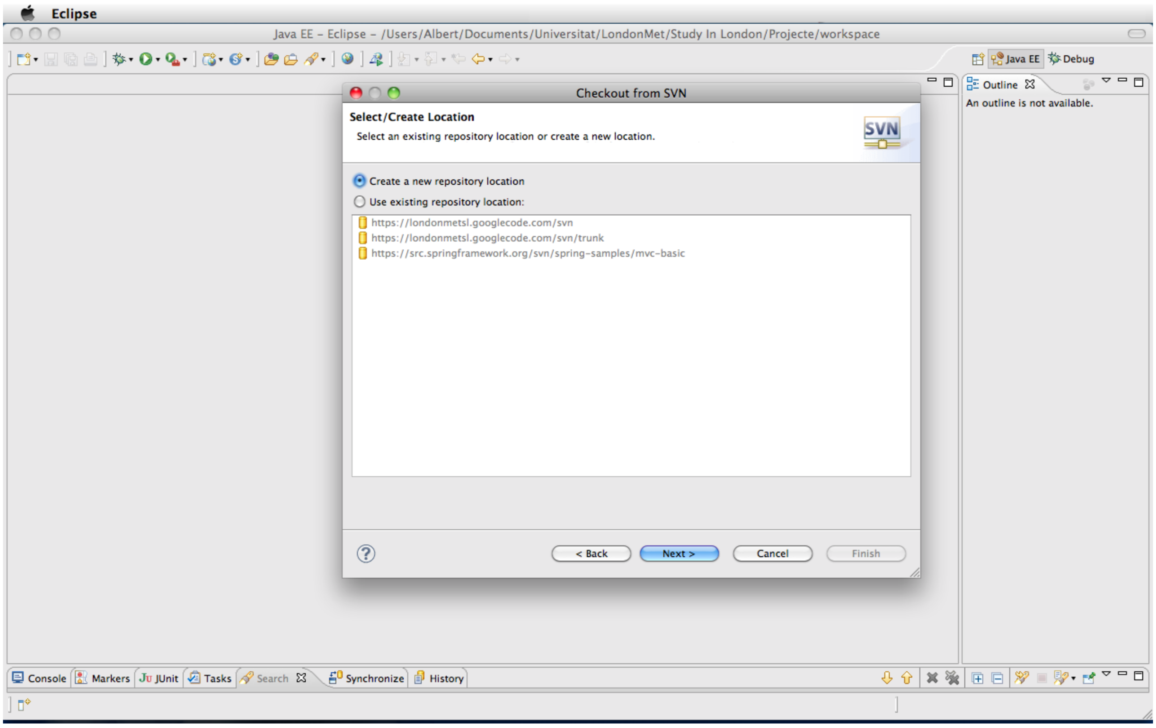
2. Select **Checkout Projects from SVN**.



Create a new project by checking out an existing project from a SVN repository.

Illustration 18. Starting the project: Step 2

3. Select to create a new repository location.



Eclipse
Java EE - Eclipse - /Users/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace
Java EE
Debug
Outline
An outline is not available.
SVN
< Back
Next >
Cancel
Finish
Console
Markers
JUnit
Tasks
Search
Synchronize
History

Checkout from SVN

Select/Create Location

Select an existing repository location or create a new location.

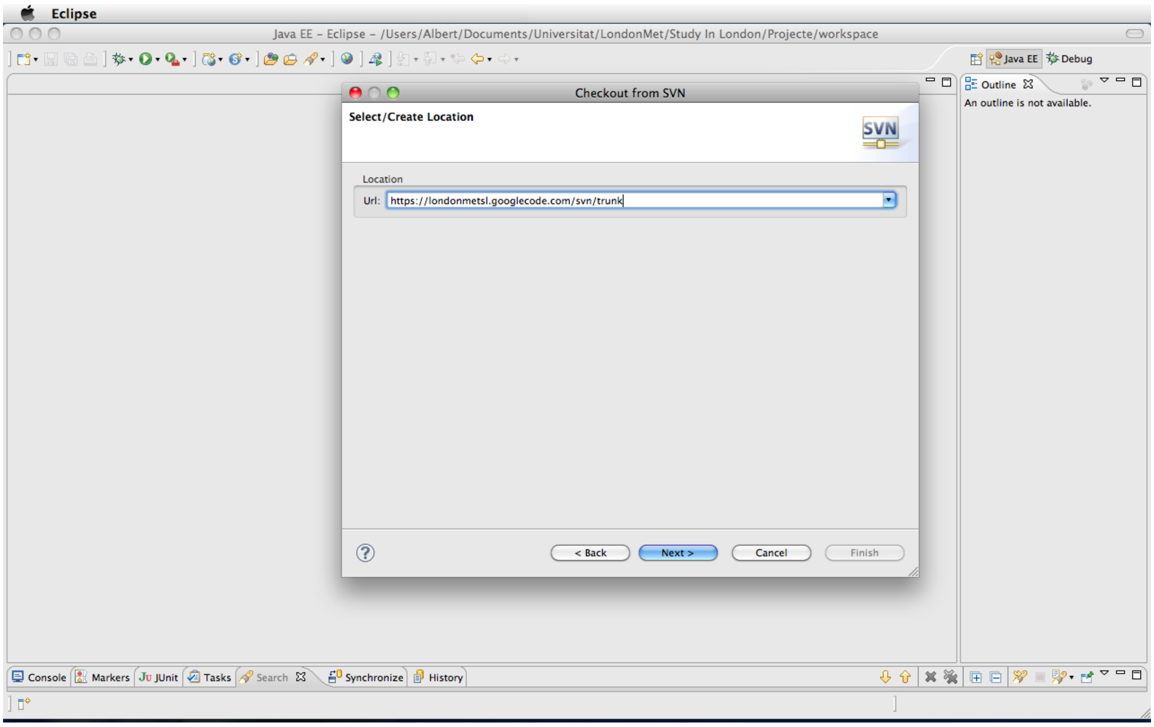
● Create a new repository location

○ Use existing repository location:

- https://londonmetsl.googlecode.com/svn
- https://londonmetsl.googlecode.com/svn/trunk
- https://src.springframework.org/svn/spring-samples/mvc-basic

Illustration 19.Starting the project: Step 3

4. Copy **https://londonmetsl.googlecode.com/svn/trunk** as the URL.



Eclipse
Java EE - Eclipse - /Users/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace
Java EE
Debug
Outline
An outline is not available.
SVN
< Back
Next >
Cancel
Finish
Console
Markers
JUnit
Tasks
Search
Synchronize
History

Checkout from SVN

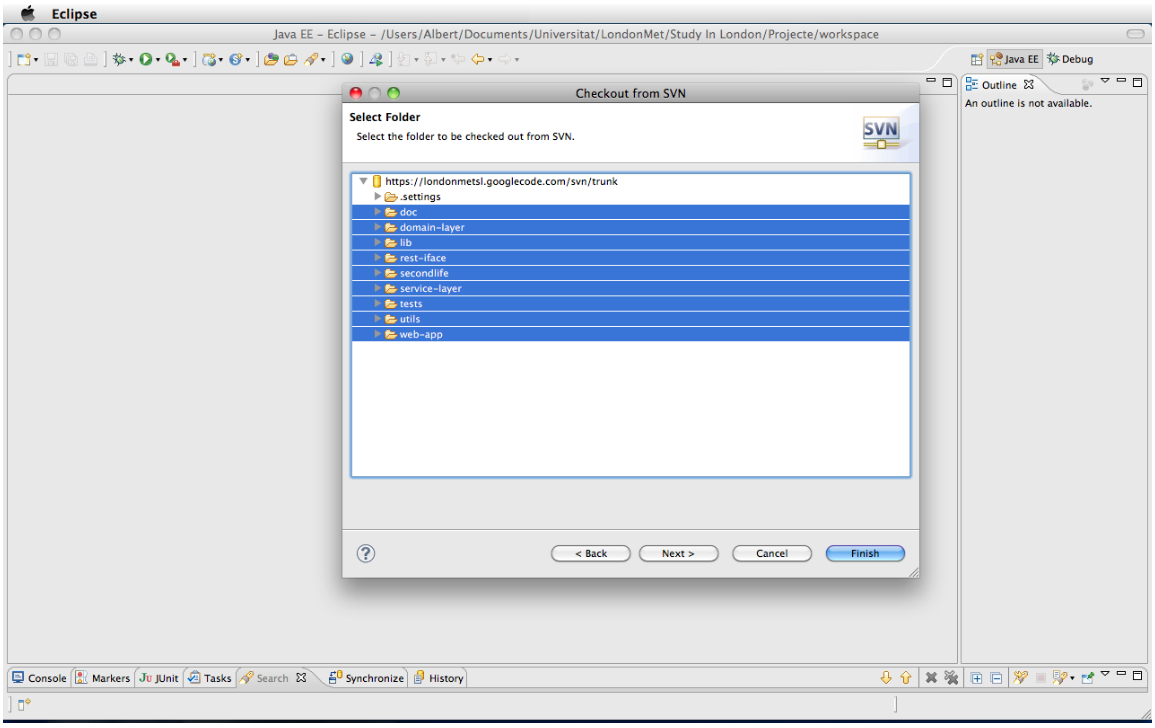
Select/Create Location

Location

Url: https://londonmetsl.googlecode.com/svn/trunk

Illustration 20. Starting the project: Step 4

5. Select the nine projects (not the root), as it is more interesting to have one project unit per each project.


Eclipse
Java EE – Eclipse – /Users/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace
Java EE
Debug
Outline
An outline is not available.
< Back
Next >
Cancel
Finish
Console
Markers
JUnit
Tasks
Search
Synchronize
History

Checkout from SVN

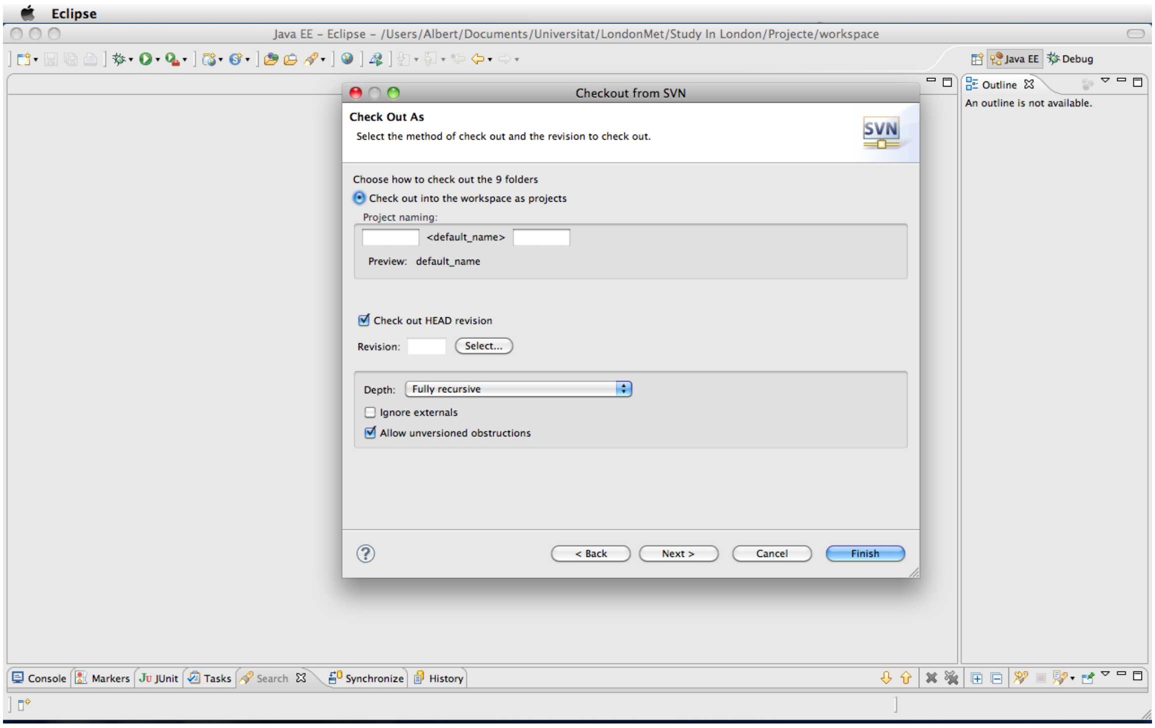
Select Folder

Select the folder to be checked out from SVN.

```
▼ https://londonmetsl.googlecode.com/svn/trunk
  ▶ .settings
  ▶ doc
  ▶ domain-layer
  ▶ lib
  ▶ rest-iface
  ▶ secondlife
  ▶ service-layer
  ▶ tests
  ▶ utils
  ▶ web-app
```

Illustration 21. Starting the project: Step 5

6. Select to check out as nine different projects, at the HEAD revision.


Eclipse
Java EE – Eclipse – /Users/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace
Java EE
Debug
Outline
An outline is not available.
Select...
< Back
Next >
Cancel
Finish
Console
Markers
JUnit
Tasks
Search
Synchronize
History

Checkout from SVN

Check Out As

Select the method of check out and the revision to check out.

Choose how to check out the 9 folders

● Check out into the workspace as projects

Project naming:

<default_name>

Preview: default_name

☑ Check out HEAD revision

Revision:

Depth: Fully recursive

□ Ignore externals

☑ Allow unversioned obstructions

Illustration 22. Starting the project: Step 6

7. Finally click **Finish** and wait for the code to be downloaded.

Note: *Apart from we two developers, for the moment only Alan Hudson (from London Metropolitan University) has access to view and commit to the Subversion code. In case of need to extend it, please contact us.*

4.2 Architecture of the system

The system we have designed has been optimally structured in order to be easily expandable and understandable. In order to do so, each part of it is clearly delimited. We can split the whole environment in two ways: by client and server, or by application tiers, all of them visible in Illustration 29**.**

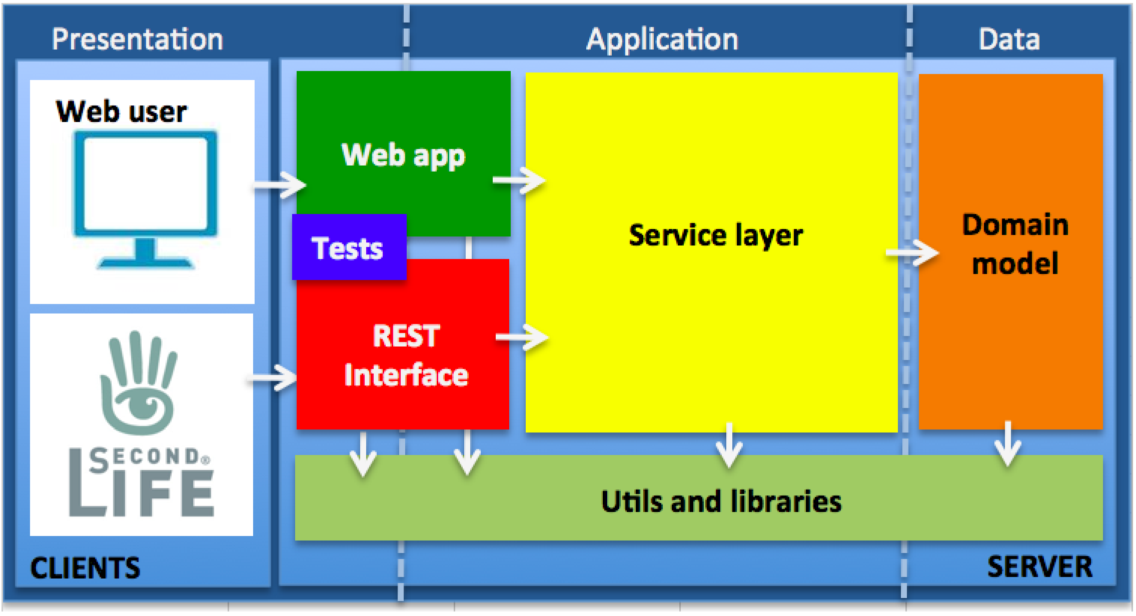


Illustration 23. General architecture overview

The client and server way of understanding our environment is pretty straightforward, and understandable even if we have no notions of application architecture. On the one hand, we have a set of clients (19), one independent from each other (in Illustration 29, the left light blue panel). Each client communicates with the server at its own way:

- **Web user:** Accesses the server through a web page visible via a browser. The main purpose of the web user is to get information (the whole meetings list, the activities for each meeting, the location list) and manage it (creating, modifying and deleting meetings, activities and locations).

- **Second Life:** Accesses the server through a specific interface, either sending or receiving data. The main purpose of this interface is to collect data from the Second Life environment (people in a place for a specific moment, answers to the tests, the time instant when someone gets in or out, etc.).

On the other hand, we have a **server**. From the clients' view, the server acts as a black box, which computes, stores and retrieves all the information. These operations work transparently for the clients, which simply know how to communicate with that server in order to send and receive data: clients ignore how the server deals with all that data.

4.3 Client Side

The client side has been developed in two main parts. The first has consisted on building the infrastructures needed to carry out a prototype in Second Life for an e-learning lesson. We have to point out that these infrastructures are completely functional, what means that they haven't been designed to have a good esthetic but only to do what they have to do. As all the functionalities are fully related to the server, all the development has been realized in parallel with the development of the server. The second part, instead, consists on a web user application to interact with the server through a user-friendly interface.

Below we are going to detail the entire process we have done in the client side until the end of the project.

4.3.1 Registration in Second Life and creation of avatars

Obviously the first step was registering on Second Life[1]. This consists on giving some personal details (that will not be shown to other residents) and choosing your avatar. Indeed you have to choose its appearance according to the different models proposed (6 males and 6 females) and its full name (you can put the name you want but the surname must be chosen from a given list).

[1] https://join.secondlife.com/

After doing this, we had to choose a password that, together with the full name of the avatar, were our sign in data to access Second Life through the Viewer.

4.3.2 Construction of the infrastructures

As we needed a permanent land to create our buildings, we were not able to use any sandbox, as it would have erased our creations from time to time. That's why our supervisor invited us to the group of the university (London Metropolitan University group), which owns a great parcel of land where we were able to construct our buildings permanently, located in the region of Naylor with the coordinates (66, 152, 31). This location can be reached with the SLurl http://maps.secondlife.com/secondlife/Naylor/66/152/31.

Illustration 24. Free parcel of LondonMet's land

As our objective was to create a prototype of an e-learning room, we have done, to begin, a very simple single room to develop the functionalities. For this we have used, as explained in Chapter 2, simple prims that we have linked between them to form more complex structure.

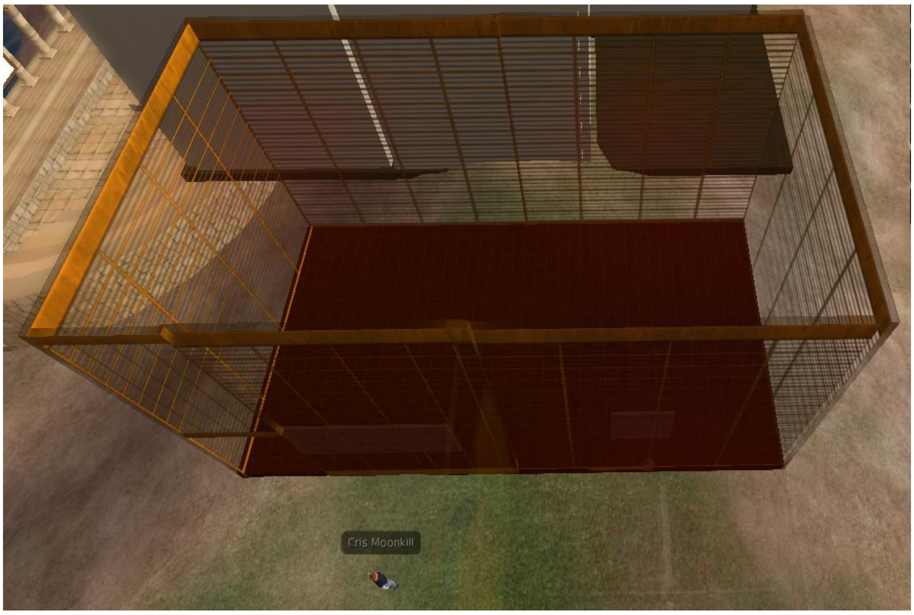


Illustration 25. Simple room

As the project has progressed, this room has changed to meet our needs, until reaching its final aspect which the next one:

Cris Moonkill

Illustration 26. Final room

The client side system focuses its functionality on four activities: registration of new students, access to the e-learning room, control of presence of the students and test of the knowledge of the students.

4.3.3 Registration of students

One of the most important features to take into account in order to realize e-learning activities is the identification of the participants. It's essential to have some data about the professors, students, or any other type of assistant not only to identify them, but to be able to assign the activities, tests, assistances, etc they might have done to them.

As the first objective of the whole project is to attract as much students as possible into the land of the London Metropolitan University in Second Life, we've tried to simplify as much as possible the process, automating all the steps we were able to. In case of the registration, as we only collect personal data from the avatars that are public, we have created a simple post where the avatar only have to touch a button to register itself in the system. The post gives also some information about the success or not of the registration.

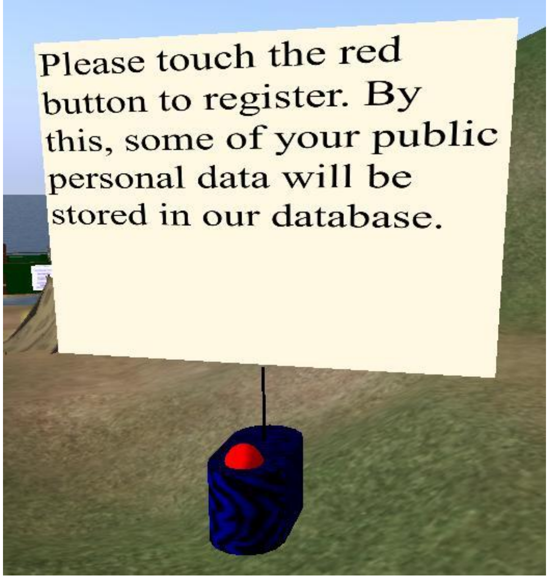

Please touch the red button to register. By this, some of your public personal data will be stored in our database.

Illustration 27. Registration point

To be more specific, the registration point collects 4 personal data from avatars:

- Identification number in Second Life
- Name of the avatar
- Language of the avatar
- Birthday of the avatar

The first three data, considered basic information, can be collected directly from the avatar using LSL functions from the library:

- *llDetectedKey(0)* returns the identification number of the avatar in *S*econd Life
- *llDetectedName(0)* returns the name of the avatar
- *llGetAgentLanguage(id)* returns the language of the avatar

For the birthday, the procedure is a little bit different, as the birthday is not considered basic information, so it's necessary to make a request to the dataserver of Linden Labs. This procedure, as many in LSL, works is an event-driven function divided in two parts. At first, we make a request to the server indicating the id of the avatar and the information wanted:

- *llRequestAgentData(id, DATA_BORN)*

This function doesn't return the information requested, but the id of the query. The value is given by an event produced by the server of Linden Labs which is handled by the function *dataserver (key queryid, string data)* which is identified by the id of the query done by the script, and the value of the information requested. The function *llRequestAgentData* allows also to recover if the avatar is online, its name, its rating in the virtual world and information about payment.

After this, we send the collected data to our web dataserver in order to store them. This sending is also divided in two parts.

The first part consists on coding the information we are going to send not only for security but also to simplify its interpretation in the server. We have chosen the format JSON because of its simplicity of use and its high presence in many technologies, as it's derived from Javascript, which interested us very much as our web dataserver has been developed in Java. In fact, JSON is natively supported by the Internet protocols as the media type *application/json*. Its main function is to serialize and transmit structured over a network connection thanks to a simple and defined schema, which is quite similar to the XML one. (26) As an example, the 4 information collected, after their codification under the JSON scheme, look like:

```
{"id": "8560c937-6e5c-42cd-9d65-39e69f0ca27e",
"name": "Cris Moonkill",
"birthday": "2010-07-19",
"language": "en"}
```

Illustration 28. Registration data coded with JSON scheme

The second part corresponds to the sending of this serialized data through the Internet. For this, we did HTTP Request indicating that the content of the request (HTTP MIMI Type) was JSON text. That's why the method of the requested was POST, as it is used to submit data which has to be processed to the identified resource. The HTTP Request is implemented in the library of LSL through the function *llHTTPRequest(string url, list parameters, string data).* Between the list of parameters, apart from the method and the mime type, it's possible to indicate the maximum length of the sent data and if the request must be done through a secure channel, that's to say with HHTPS protocol with SSL. In response to this request, our dataserver sends an HTTP Response which is handled by the script through the function *http_response(key request_id, integer status, list metadata, string body)*. This response contains important information about the success or not of the request. In our case, we have decided to use the status to identify, in all the requests done in the client side, if the request has been successful or failed. In the first case, the server sends the status code 201. Otherwise, we have grouped all the possible errors, either caused by the user or by the server, with the code 304, but it's very easy to give more details about the error and adapt the feedback message. It's enough to add more branches *else if* into the *http_response* of the script. Both codes come from the standardized HTTP Response codes (21) for HTTP Request. According to this standardization, the meaning of these codes is:

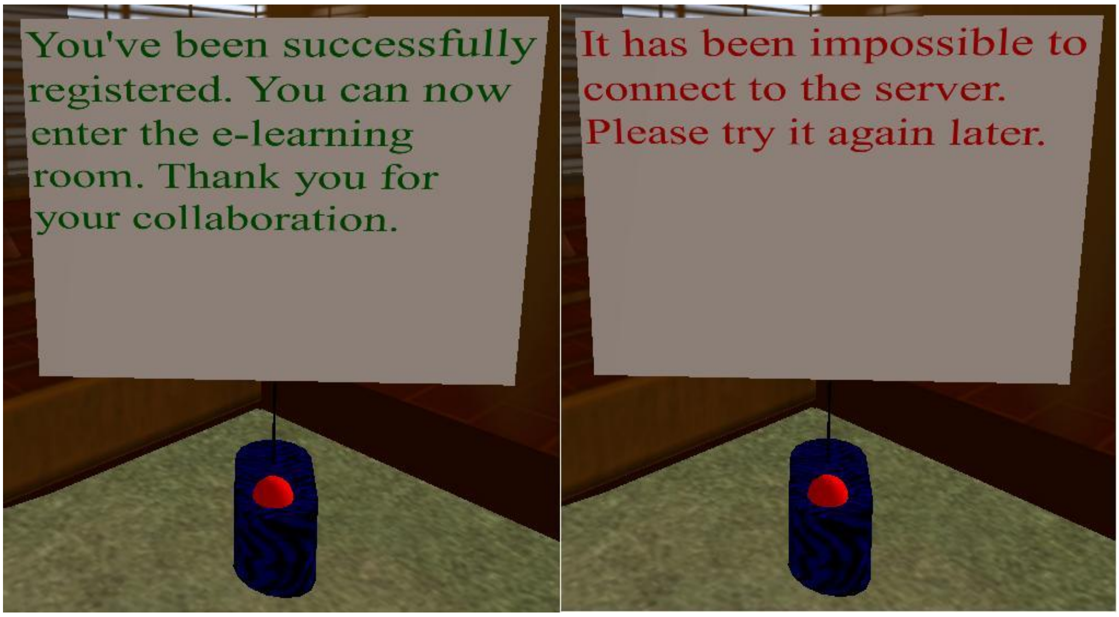
- **201: "*Created***

  > *The request has been fulfilled and resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the entity of the response, with the most specific URI for the resource given by a Location header field. The response SHOULD include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. The origin server MUST create the resource before returning the 201 status code. If the action cannot be carried out immediately, the server SHOULD respond with 202 (Accepted) response instead."* (W3C)

- **304: *"Not modified***

  > *If the client has done a conditional GET and access is allowed, but the document has not been modified since the date and time specified in If-Modified-Since field, the server responds with a 304 status code and does not send the document body to the client."* (W3C)

Thanks to this, depending on status code received, we are able to know in Second Life if the request has been successful or not, and give a feedback information to the resident who, in case of fail, is aware that his/her avatar must, for example, redo the registration process a little bit later. For this, the button, which has collected and sent the information to server, must be linked to the screen. In this manner, the button sends a linked message, which is only received by the prims that are lined together. This can be done thanks to a function from the library of LSL, *llMessageLinked(integer linknum, integer num, string str, key id)*, where *linknum* indicates to which prims we want to send the message, *num* is an identification code and both *str* and *id* are the fields containing the message. In our case, we only have used the identification code using 0 for the success and -1 for the fail. In case of distinguish the different errors, it will be very useful to use the fields *str* and *id*. On the other hand, the screen has also a script which receives this message with the handler function *link_message(integer sender_num, integer num, string str, key id)* which receives exactly the same parameters sent by *llMessageLinked*, and then, according to the identification code received, shows one or other message.



You've been successfully registered. You can now enter the e-learning room. Thank you for your collaboration.

It has been impossible to connect to the server. Please try it again later.

Illustration 29. Feedback texts when success and fail

In order to prevent unwanted behaviors of the screen, we must configure some of its features. This process will be done in the handler *state_entry* of the state *default*, which is launched every time the state is entered. The features will be customized through the function *llSetPrimMediaParams(integer face, list params)* and the ones listed below:

- **PRIM_MEDIA_AUTO_SCALE**: Sets whether auto-scaling is enabled. Auto-scaling forces the media to the full size of the texture. Its value will be *FALSE*.
- **PRIM_MEDIA_FIRST_CLICK_INTERACT**: Sets whether the first click interaction is enabled. Its value will be *FALSE*.
- **PRIM_MEDIA_PERMS_CONTROL**: Sets the permissions mask that control who can see the media control bar above the object. Its value will be *PRIM_MEDIA_PERM_NONE*, which denies all the permissions to all the users.
- **PRIM_MEDIA_PERMS_INTERACT**: Sets the permissions mask that control who can interact with the object. Its value will be *PRIM_MEDIA_PERM_NONE*, which denies all the permissions to all the users.

4.3.4 Access to the room

Once registered, the avatars are allowed to enter the room. The objective of this access control is not to ban the entry to avatars outside the system but to have information about the users that assist the e-learning activities. This will allow e-learning professors and managers to have some feedback to improve their methods and activities according to the profiles of these users.

As in the registration point, the access control has been made automatically in order to ease, and thus promote, its use. For this, the user has only to touch a button at the entrance of the room. In this case, there are two data that are collected and sent to the server to verify the identity:

- Identification number in Second Life
- Location of the room

The first one is collected in the same manner as in the registration point. Regarding the location, this must be set statically into the script. The function of the location is to identify which room is accessing a certain avatar, which is essential because, at every moment, in a room can take place only one activity and therefore, knowing the place and the time (known directly by the server when it receives the request), the avatar can be assigned univocally to one activity (that be discharged previously through the webserver). As in the registration point, both information are sent using the JSON scheme.

Once the request has been sent, the server verifies if the avatar is registered on the system and if there is an active activity in the room at the moment, after which it sends an HTTP response to Second Life. As in the previous case, this response contains a status code indicating if the access is allowed or not, and depending on the value of this, a linked message is sent to the door (the access button and the door must be linked) indicating if it must open or not. When the avatar wants to exit the room, there's another button that must be touched by the avatar in order to open the door. However, in this case there's no request to the server, because if the avatar is into the room, it means that it has been allowed before to enter. So, touching the exit button always implies the opening of the door. In both cases, either when entering and exiting, the door closes itself automatically five seconds after it opening (this value is modifiable in the script).

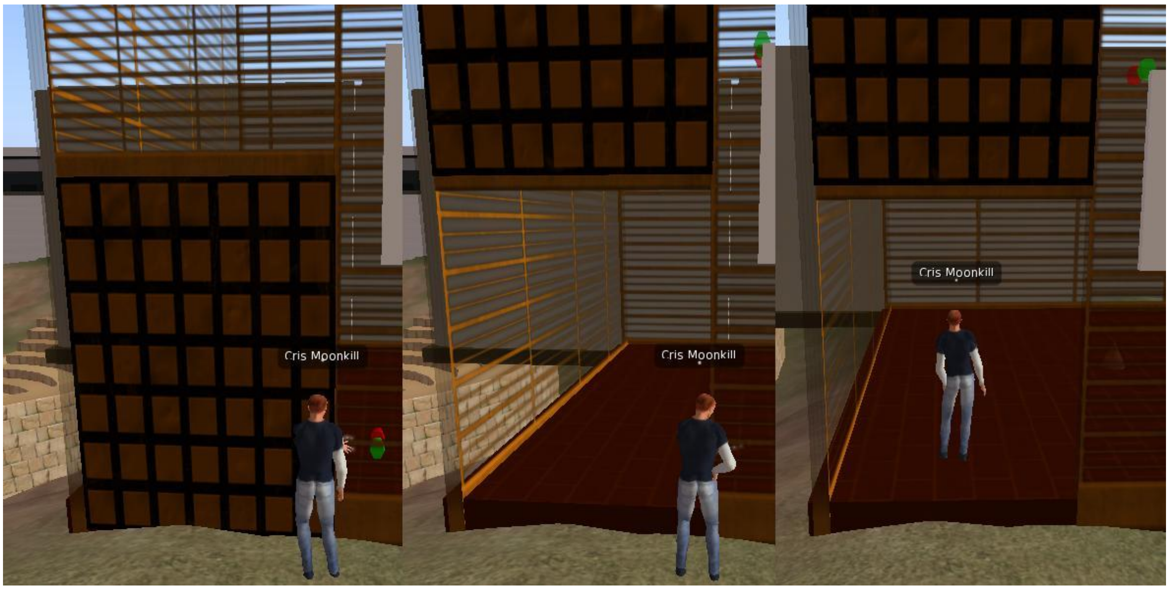


Illustration 30. Steps to enter the e-Learning room

4.3.5 Control of presence

This step is the only one that is absolutely transparent for the users. Its function consists on scanning the room every so often in order to detect which avatars are present in it. By this, the system knows exactly who has attended an activity, when has an avatar arrived to the room and when has he exited, and so on.

This scan is done cyclically by a set of fixed objects acting like scanners. In order to cover all the room, we have put as many objects as needed to cover all the surface of the room. In fact, the objects scan up to a radius of 96 meters from them. That's why combining some objects in a correct position we can cover the entire surface, except the corners which are not accessible by the circles of scanning, as we can see in the illustration 7.

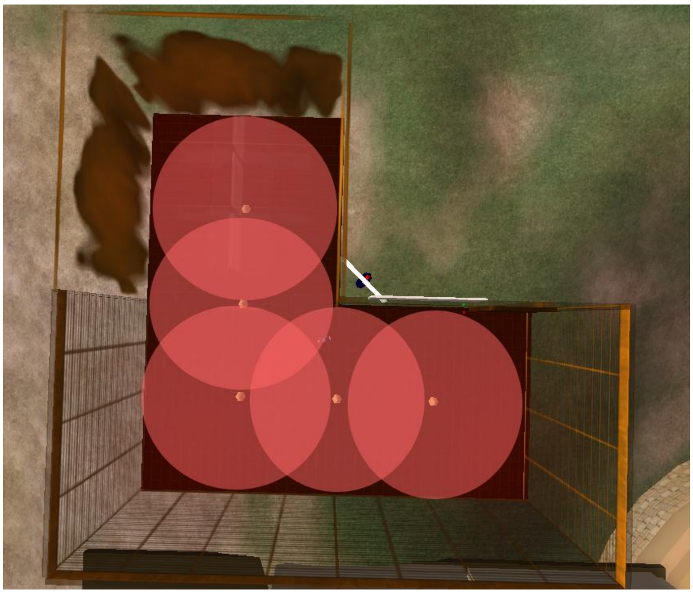


Illustration 31. Room with zones of scanning in red

These sensors work in a collaborative way using a master/slave scheme. Indeed, there's one object which acts as master, and the others as slaves. This model allows using as many sensors as wanted and so, if in a future the shape of the room changes or other rooms are added, it's enough with creating new objects that cover the new surface and link them to the others. Moreover this, the scanning runs in cascade, namely objects scan their zone one after another. This way, the first one, which is the trigger of a scanning process, is the master which begins a scan at regular intervals. This is done with the function *llSensorRepeat(string name, key id, integer type, float range, float arc, float rate)* which scans within a range of *range* every *rate* seconds.

While scanning, the master activates a handler function, *llListen(integer channel, string name, key id, string msg)*, which allows it to listen to the last slave through a random public channel known by both and asks for the first slave to do a scan of its zone through another random public channel known by both, through the function *llSay(integer channel, string msg)*. This slave, which must have set previously a handler function (*llListen*)to listen on this public channel, scan its zone (using the function *llSensor* as asked and spreads the request to the next slave, attaching to it the avatars it has scanned in its zone, through a third random public channel known, as always, by both. This second slave updates the list of avatars received by its predecessor with the avatars scanned in its zone. The process continues as this until the requests arrives to the last slaves, which, after scanning its zone, has the final list of avatars detected by all the slave sensors. This list, which is sent to the master, is finally updated by this last with its detections and sent to the server. It's important to highlight that all the channels used, both for sending or receiving information, are chosen by the manager of the scripts. The only issue to consider is that in all the sensors, the receiving channel must match with the transmitting one of the previous sensor, and the transmitting one must match with the receiving one of the next sensor. As the last slave sensor communicates with the master, we can consider that the next sensor of this last slave sensor is the master, so the sensors form closed circle. In our example, we have used these channels to allow the communication between the sensors:

Sensor	Receiving channel	Sending channel
Master	150	151
Slave 1	151	152
Slave 2	152	153
Slave 3	153	154
Final Slave	154	150

Table 2. Public communication channels used between the sensors

At the beginning, we have considered to link all the sensors in order to use the LinkedMessage that are private and so not visible to third parties. However it was impossible because the scan was done without any control, which caused an uncontrolled duplication of ids that overflows the system and forces the script to raise an error and stop until it is reset. We think that as all linked prims behave as the same object, when one scans all the others do it also, but this leads to an infinite loop of scanning as every sensor incite the others to scan and vice versa.

In order to solve the problem of privacy, we decided to encrypt the communications. For this, we have chosen the RSA method because it is one the most well-known and used currently, and above all because LSL has library functions that allow to figure up all the necessary calculations. This method consists on 5 steps:

1. Choose randomly two distinct prime numbers[1] p and q.
2. Compute $n = pq$.
3. Compute $\phi(n) = (p-1)(q-1)$
4. Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e,\phi(n)) = 1$, i.e. e and $\phi(n)$ are coprime.
5. Determine $d = e^{-1} \bmod \phi(n)$

[1] A prime number (or a prime) is a natural number that has exactly two distinct natural number divisors: 1 and itself

At this point, the encrypted message *c* is obtained computing:

$$\mathbf{c = m^{e} \bmod n}$$

And the decrypted message *m* can be obtained computing:

$$\mathbf{m = c^{d} \bmod n}$$

Once *n, d* and *e* have been calculated, *c* and *m* can be obtained directly without the function *llModPow(integer a, integer b, integer c)* that returns exactly a^b mod c. Nevertheless, in practice this method cannot be used in our prototype for two main reasons. On the one hand, in our case we need to send text messages, but RSA only encrypt numbers. For this, we were obliged to encode the text into numbers using the ASCII codes of every character. This was done with the functions *llBase64ToInteger(string str)* combined with *llStringToBase64(string str)* and *llIntegerToBase64(integer number)* combined with *llBase64ToString(string str)* that converts a text into an 64-bit ASCII code and vice versa respectively. And precisely, the use of these functions is the cause of the impossibility of using RSA, because the time needed to encode the text into a number takes too much time and, evidently, the longer is the text, the longer is the time needed. On the other hand, the number return by these functions is really high, therefore the operation of modular exponentiation is exaggeratedly expensive and it may even cause and overflow when the number that represents the text is too high.

After talking with our tutor, we have decided to do open communications between the sensors, because the information is not committed and because in general terms, Second Life is a secure place where people don't sniffs information to misuse it.

Both of scanning function act like triggers that must be caught then. There are two situations treated in the same way for both functions that must be considered: when the sensor detects someone in its zone and when it doesn't detect anyone. In the first case, the event is catch by the handler function *sensor (integer num_detected)*, which contains the list of avatars detected by the sensor. On the other hand, when there are no avatars detected, the event is catch by the handler *no_sensor()*. If this occurs, the sensor directly propagates the received list to the next sensor.

As we can see in illustration 7, there are some places that belong to more than one zones of scan, and so if there are avatars in these places, they are scanned more than one time. In these cases, the avatar will appear more than one time in the final list, but this will be treated be the server that will ignore the duplication of avatars in a same list.

Later, diagram 1 illustrates briefly the process of scanning carried out by multiple sensors with a scheme of master/slave in cascade.

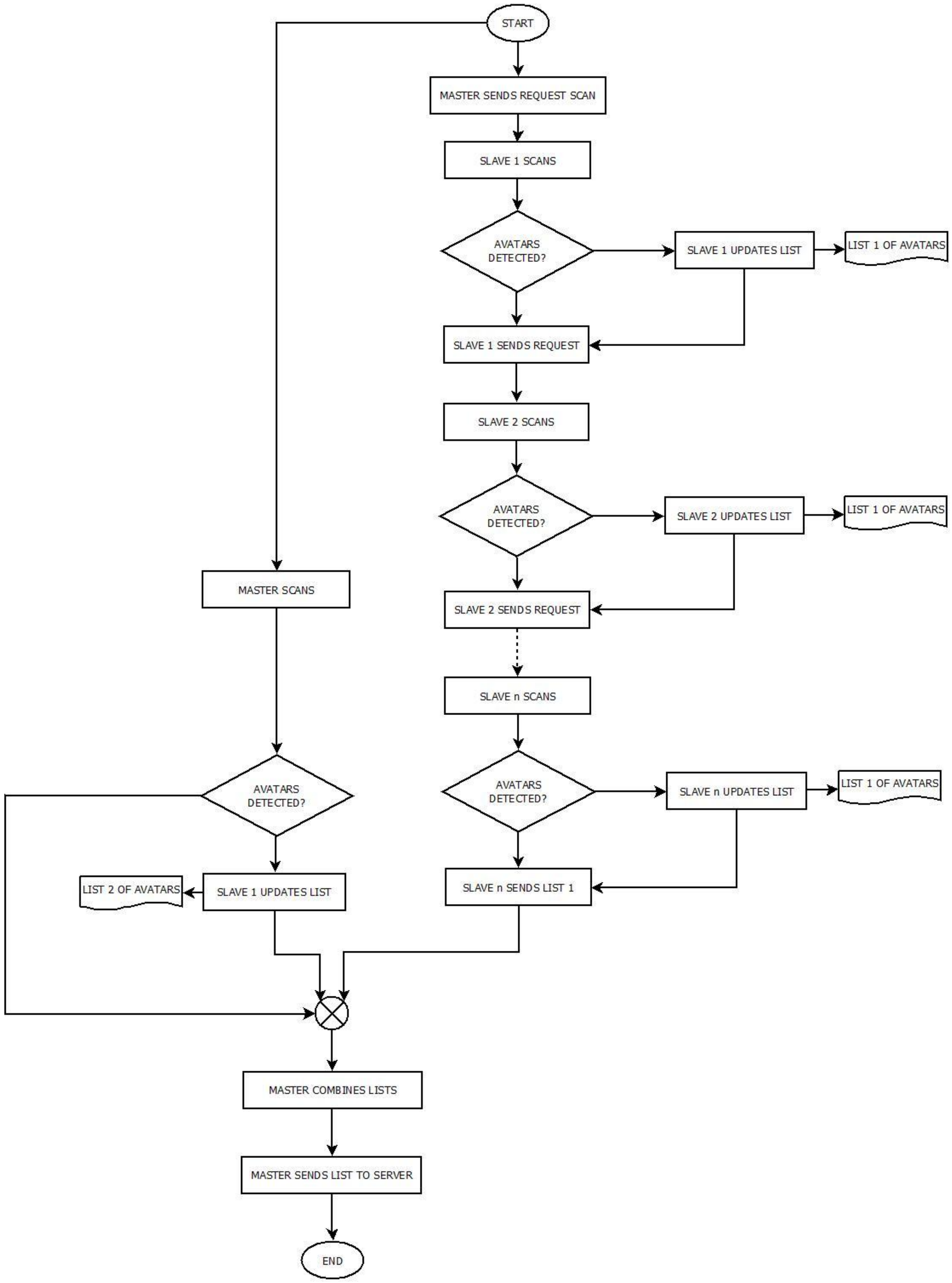


Diagram 1. Process of scanning with n sensors

4.3.6 Knowledge test

Finally, the last feature we have implemented is a station for doing tests. The main idea is that this station will allow the professors or managers of the activities to test the knowledge of the assistants either before or after the activities.

The tests works as a typical test: every question has 3 different answers, and only one of them is correct (see Illustration 31). Obviously, as the residents are using a computer with access to the Internet (and therefore to all the information contained in it) these tests don't pretend to serve as exams, but they can be very useful to give a feedback to the managers in order to adapt the activities or to evaluate if the users have obtained the desired knowledge from the activity.

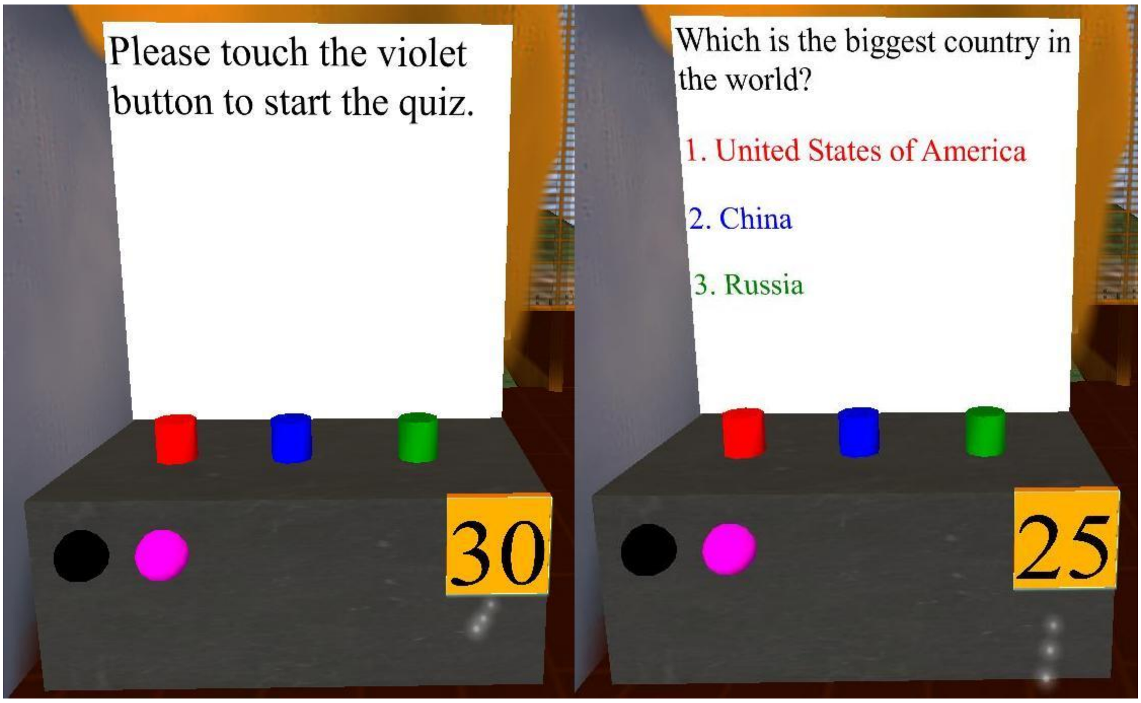

Please touch the violet button to start the quiz.
30
Which is the biggest country in the world?
1. United States of America
2. China
3. Russia
25

Illustration 32. Test station

This object is the one that interacts the most both with the server and with the avatar. The process begins when the avatar touches a button that runs a script that sends a request to the server with 2 data coded according to our well-known JSON scheme: location and avatar id in Second Life. As in the access control, the location allows, beside the time reference, to identify univocally which activity is attending the avatar. The identification of the avatar has, as always, a double function: verify if the avatar is registered on the system and associate the answers given to the avatar.

Once the system has checked the id, it sends an HTTP Response indicating the success or the fail of the request. However, in this case the request is done with the method GET, so we cannot use the status codes. Instead of them, the server sends within the body a message identifying the success or the error. In the last case, the error is sent into a simple tag *<error>* similar to that used in CSS, that fits perfectly the message and makes this easily locatable. An example of error is:

<error> alreadyAnswered </error>

On the other hand, if the avatar is registered in the system and the location exists and is active, the HTTP Response contains in its body all the questions and its corresponding answers associated to the activity. For this, we have also used tags to identify every one of them. To be more specific, we have used 3 types of tags: *<num_questions>, <questionX>, <answerXY>.* The first one contains the number of questions, the second contains the question number X and the last contains the answer Y (Y goes from 1 to 3, as every question has 3 possible answers) to the question X. Below, in Illustration 34, we show an example of the message received from the server.

In order to ease the process, the station has 3 buttons to answer the questions, and each ones has the same color as every possible answer, so it's really intuitive. As in the registration point, the stations have a screen that shows not only the questions and its answers, but also some feedback messages to guide the avatar through the test. So, again we must link all the objects forming the station in order to allow them to communicate privately. Indeed all the objects (buttons, screen, timer) send linked messages that are caught only by the script of one the linked objects which is the responsible of all the communications with the server and all the transition needed during the test. Because of there are some actions that make requests, and therefore receive responses, and as within a state the handler function *http_response* catches all the responses, we have decided to create different states so that every state will receive its response without confusion. This scheme is also very useful and secure because it prevents some actions to be done depending on the state. These are the states we have used and their functionalities:

- Default
  - Sets the location
  - Sets the timer to answer every question
  - Transit to the other states (at the end, all the states return to this state) depending on the linked message received

- Reset
    - Sends a request to the server indicating that the avatar hasn’t finished the current test (sends avatar id in Second Life and location, with JSON scheme)
    - Makes the screen show the initial message
    - Clear the timer
    - Jumps to the Default state

- Start
    - Sends a request to the server with the avatar id in Second Life and the location, asking for the questions and its answers
    - Catches the response of the server
    - If there’s been an error, it sends a linked message (there are different codes depending on the type of error) to the screen, which shows a feedback message
    - If there’s no problem, it extracts and organizes into a list the questions and its answers
    - Jumps to the Update Screen state

- Update Screen
    - If there are still questions to answer, it sends with a linked message the next question and its answers to the screen
    - If there are no more questions to answer, it sends a linked message with a code indicating that the test has finished
    - Jumps to the Default state

- Send Answer
    - Depending on the button touched by the avatar, it sends a request to the server indicating the id of the avatar in second Life, the location, the number of the question and the number of the answer selected.
    - Catches the response of the server
    - If the request has been successfully served, it jumps to the Update Screen in order to show the next question
    - If the request has failed for any reason, it asks the screen, through a linked message, to show a message error.

<num_questions> 5 </num_questions>

<question1> Which is the biggest country in the world? </question1>
<answer11> United States of America </answer11>
<answer12> China <answer12>
<answer13> Russia <answer13>

<question2> Which is the current Spanish League 'La Liga' winner?</question2>
<answer21> Real Madrid CF </answer21>
<answer22> FC Barcelona <answer22>
<answer23> Valencia CF <answer23>

<question3> Which In which World Wars did the United Kingdom take part? </question3>
<answer31> Only the World War II </answer31>
<answer32> Both World War I and II <answer32>
<answer33> None <answer33>

<question4> Which is the theorem which relates the squared sides of a triangle with its hypotenuse? </question4>
<answer41> Pythagorean theorem </answer41>
<answer42> Euclid theorem <answer42>
<answer43> Rouché-Capelli theorem<answer43>

<question5> Who is Bill Gates? </question5>
<answer51> Apple's co-founder </answer51>
<answer52> Microsoft's CEO <answer52>
<answer53> Microsoft's co-founder <answer53>

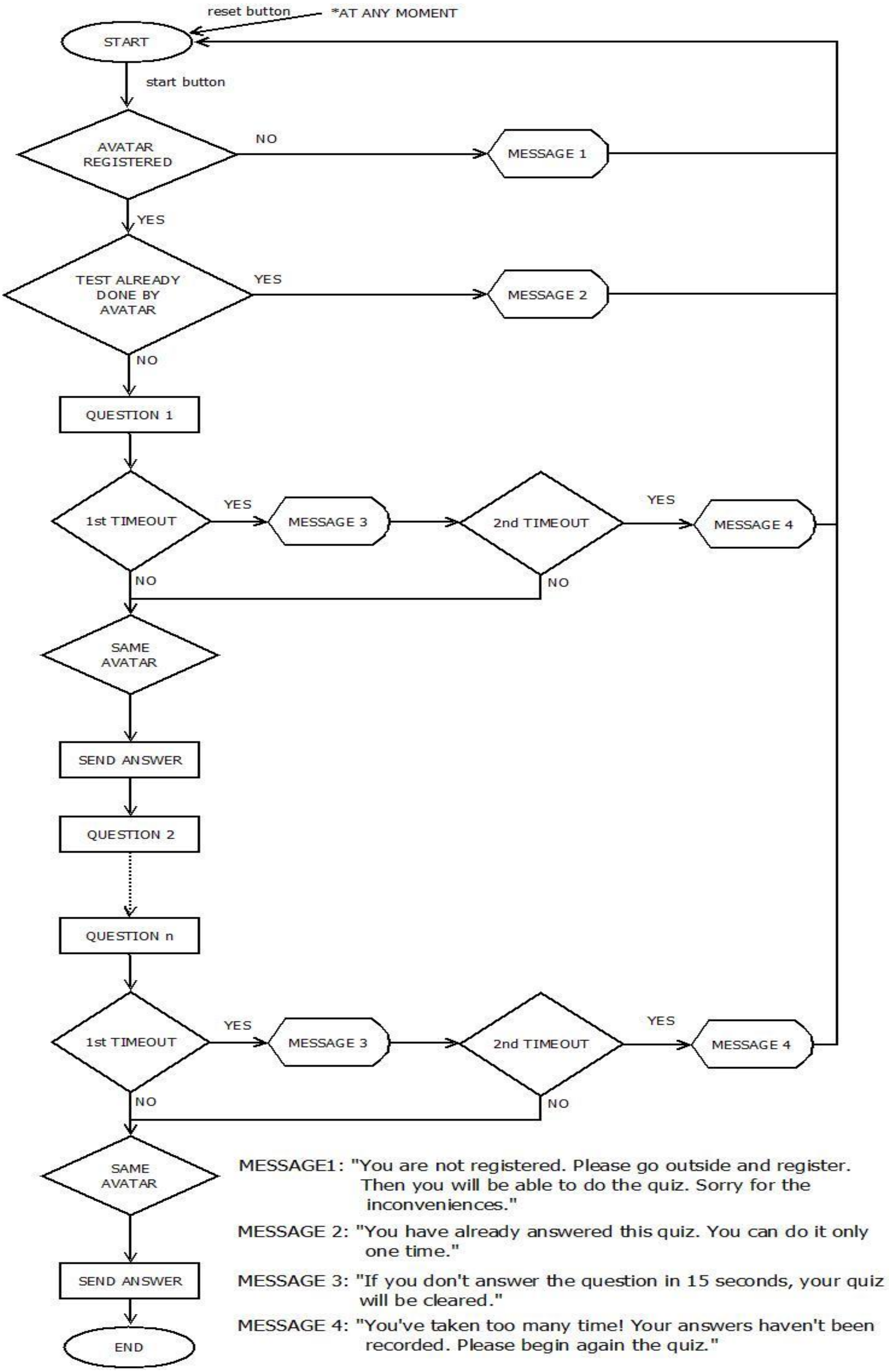
Illustration 33. Questions and answered received with a tagged scheme

In addition, we have implemented some control features. One the one hand, we have established a maximum time to answer a question. For this, every station has a counter that makes a countdown, currently of 30 seconds, and that becomes red when the middle of the countdown is reached. If the avatar doesn't answer before the end of the countdown, the test is automatically reset. To help the avatar, we have added a counter in the station that shows how many seconds remain before the restart. Moreover this, at the middle of the countdown, the avatar receives a message warning of the end of the countdown. On the other hand, we have implemented that when a test is started by an avatar, it must be done totally by this avatar. So, if during a test another avatar answers a question, the test is also automatically reset. Finally we have defined that the same test cannot be done more than one time by the same avatar. So, the station will not run the test if an avatar has already done it. We also have included a Reset button in the station that allows the avatar to stop and restart a test. By doing this, the test appears like undone for the avatar, so it has to redo it from the beginning, even it has answered some questions before. This button allows redoing only tests that haven't been finished by the avatar. The whole process has been summarized with the Diagram 2.

As we have told, the different parts of the station communicate via linked messages, and these allow to transit from one state to another. Below we list in the Table 3 the codes used in these linked messages to identify the aim of them:

CODE	FUNCTION
0	Reset the test, both in Second Life and in the server.
1	The avatar is asking to begin the test.
2	The avatar has asked the current question. The answer must be sent to the server.
3	The screen must be updated because a question has been answered.
4	The test has finish, so the screen must show the corresponding message.
5	The timer has reached the limit time.
6	The avatar is trying to do a test that has already done before (a test can be done only one time per avatar).
7	An avatar is trying to answer a question of a test began by another avatar.
8	There is an error when trying to connect to the server.
9	The avatar is no registered on the system.
10	The timer must be triggered as a new question has appeared on the screen.

Table 3. List of codes used in linked messages



MESSAGE1: "You are not registered. Please go outside and register. Then you will be able to do the quiz. Sorry for the inconveniences."

MESSAGE 2: "You have already answered this quiz. You can do it only one time."

MESSAGE 3: "If you don't answer the question in 15 seconds, your quiz will be cleared."

MESSAGE 4: "You've taken too many time! Your answers haven't been recorded. Please begin again the quiz."

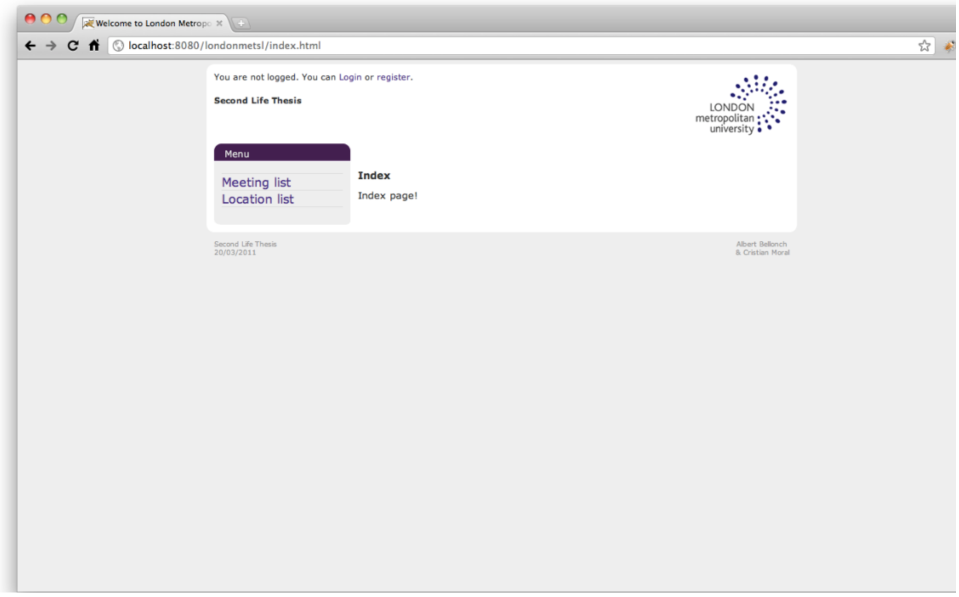
Diagram 2. Process of testing

4.3.7 Web User Application

In this section we will see the functionalities that, for now, are implemented in the system.

4.3.7.1. Register

1. Go to the root page (**http://[domain]/londonmetsl**)



Welcome to London Metropo
localhost:8080/londonmetsl/index.html
Menu
Meeting list
Location list
LONDON metropolitan university

You are not logged. You can Login or register.

Second Life Thesis

Index

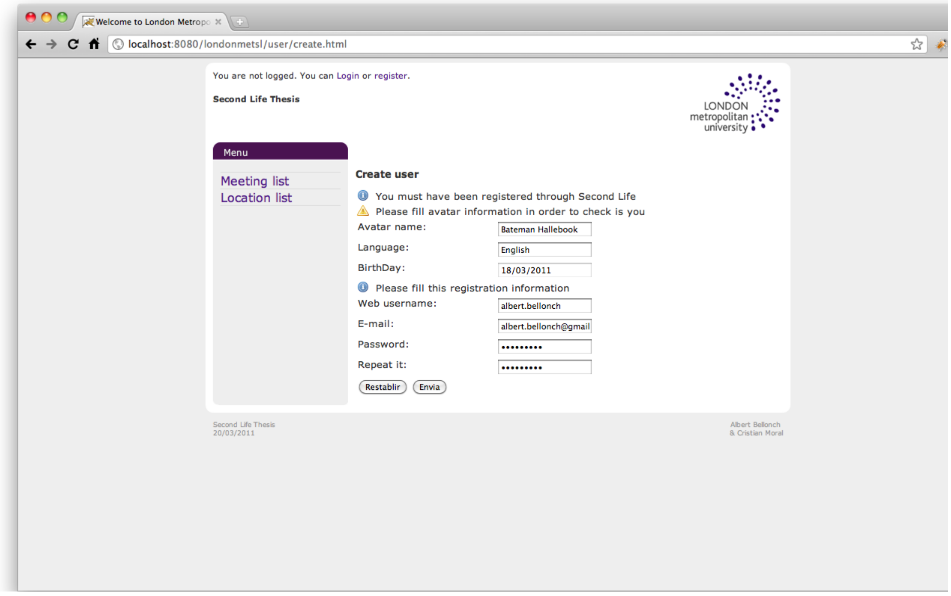
Index page!

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 34. Register in Web User Application: Step 1

2. Click on **register** and fill the fields with your data.



Welcome to London Metropo
localhost:8080/londonmetsl/user/create.html
Menu
Meeting list
Location list
LONDON metropolitan university
Restablir
Envia

You are not logged. You can Login or register.

Second Life Thesis

Create user

You must have been registered through Second Life

Please fill avatar information in order to check is you

Avatar name: Bateman Hallebook

Language: English

BirthDay: 18/03/2011

Please fill this registration information

Web username: albert.bellonch

E-mail: albert.bellonch@gmail

Password: •••••••••

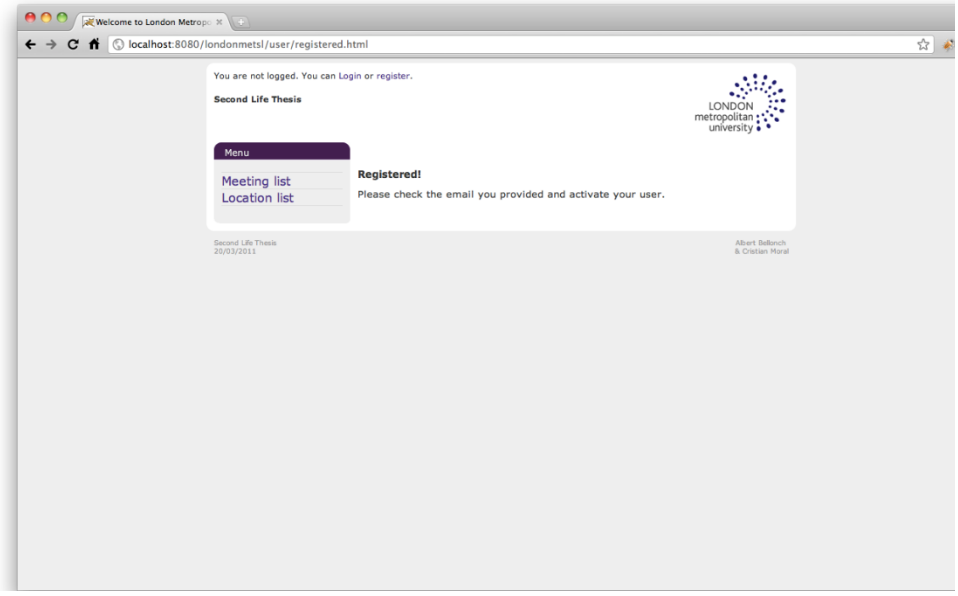
Repeat it: •••••••••

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 35. Register in Web User Application: Step 2

3. Click the **send** button. You will receive a mail with an activation URL.


Welcome to London Metropo
localhost:8080/londonmetsl/user/registered.html
Menu
Meeting list
Location list
LONDON metropolitan university

You are not logged. You can Login or register.

Second Life Thesis

Registered!

Please check the email you provided and activate your user.

Second Life Thesis
20/03/2011

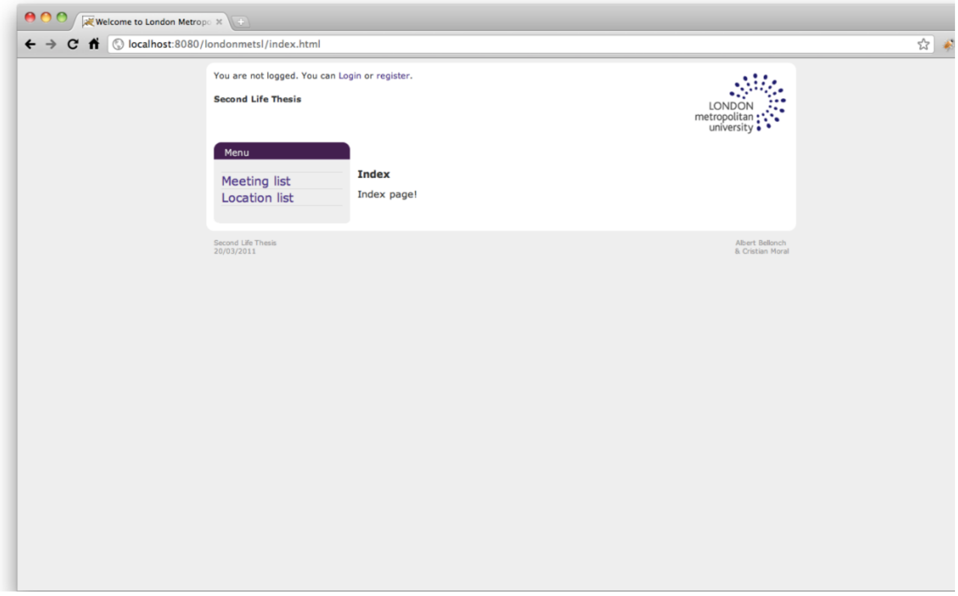
Albert Bellonch
& Cristian Moral

Illustration 36. Register in Web User Application: Step 3

4. Once accessed to that link, you will activate the new account.

4.3.7.2 Login

1. Go to the root page and click **login**.


Welcome to London Metropo
localhost:8080/londonmetsl/index.html
Menu
Meeting list
Location list
LONDON metropolitan university

You are not logged. You can Login or register.

Second Life Thesis

Index

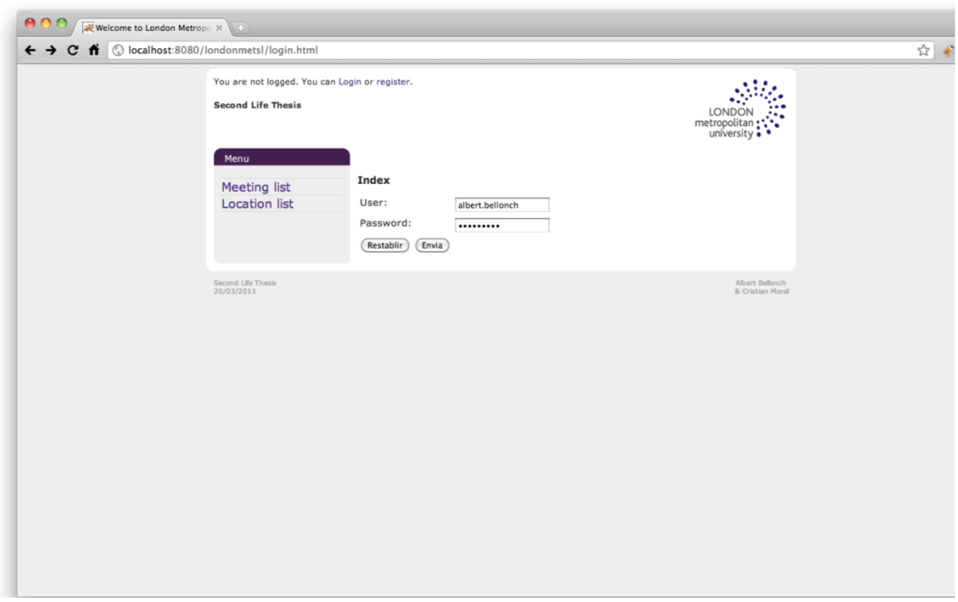
Index page!

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 37. Login in Web User Application: Step 1

2. Fill with your username and password and click **send**.



localhost:8080/londonmetsl/login.html

You are not logged. You can Login or register.

Second Life Thesis

Index

User: albert.bellonch

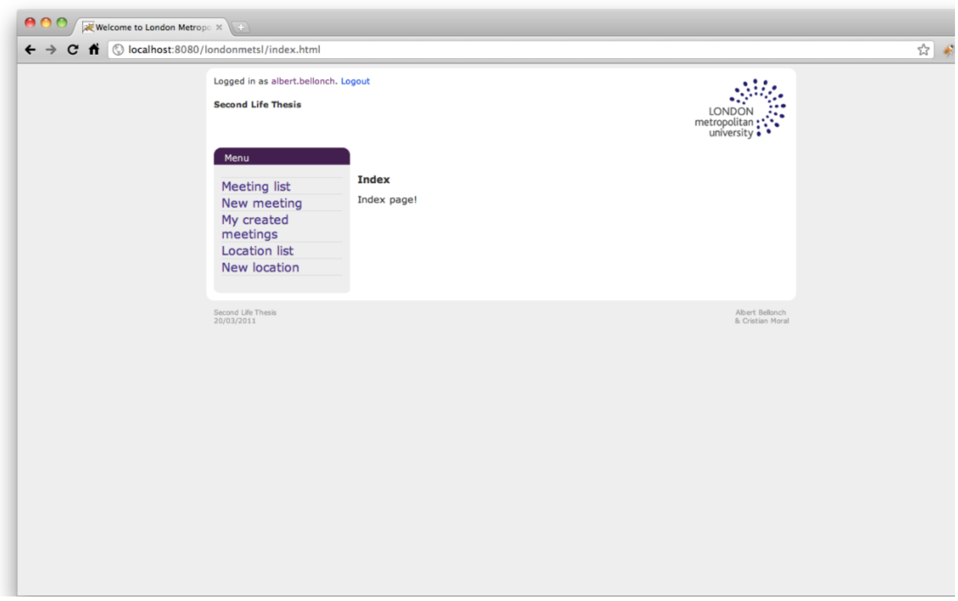
Password: •••••••••

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 38. Login in Web User Application: Step 2

3. You will go to the index page.



localhost:8080/londonmetsl/index.html

Logged in as albert.bellonch. Logout

Second Life Thesis

Index

Index page!

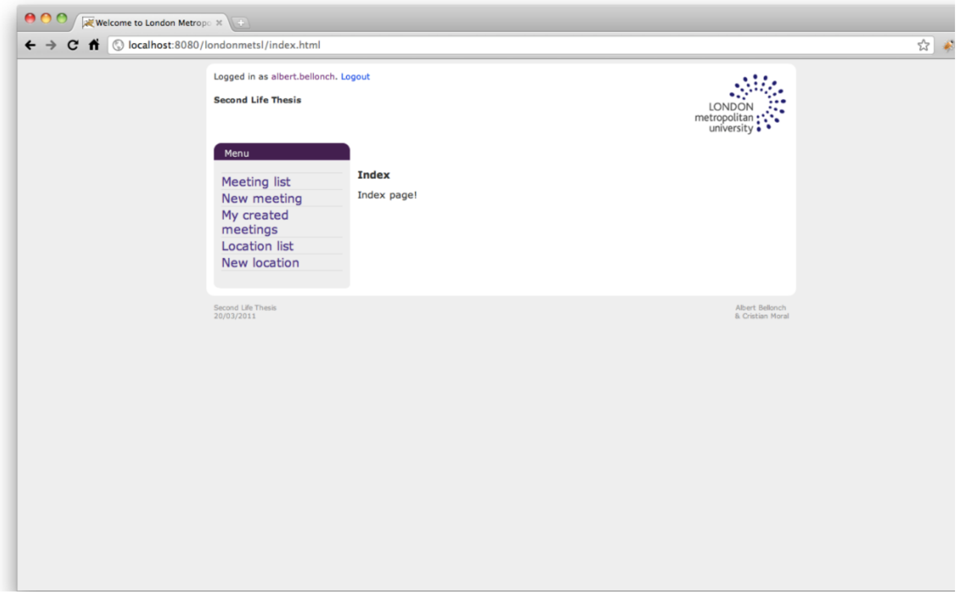
Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 39. Login in Web User Application: Step 3

4.3.7.3 Logout

1. Go to the root page and select **logout**.



Welcome to London Metropo
localhost:8080/londonmetsl/index.html

Logged in as albert.bellonch. Logout

Second Life Thesis

Index

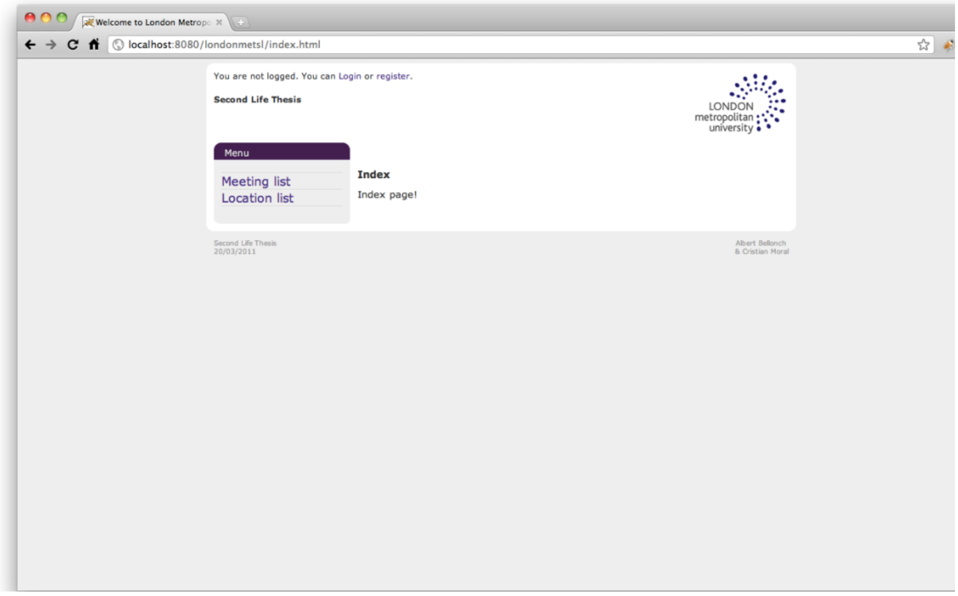
Index page!

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 40. Logout in Web User Application: Step 1

2. You will be redirected to the main page.



Welcome to London Metropo
localhost:8080/londonmetsl/index.html

You are not logged. You can Login or register.

Second Life Thesis

LONDON
metropolitan
university
Menu
Meeting list
Location list

Index

Index page!

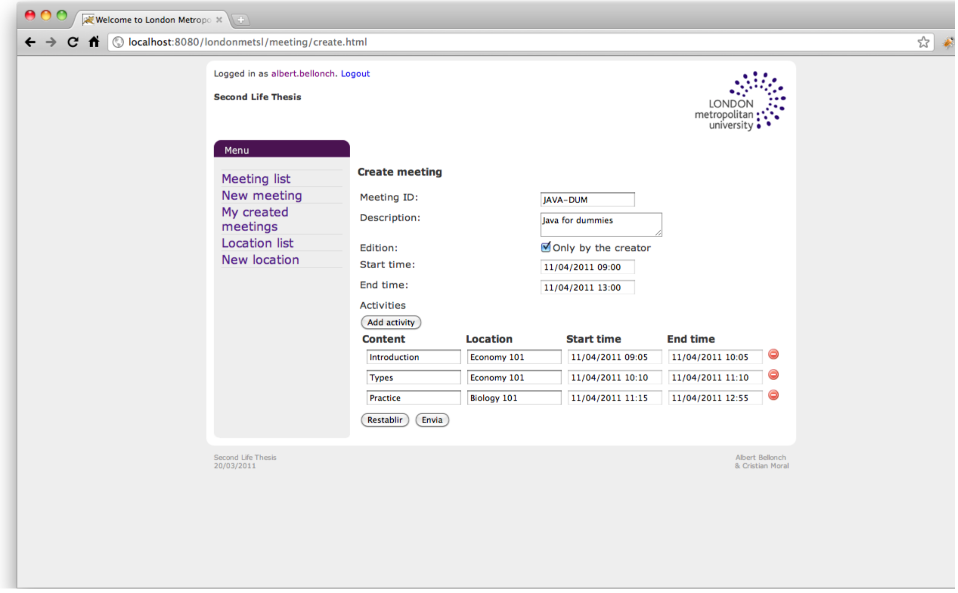
Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 41. Logout in Web User Application: Step 2

4.3.7.4 Meeting creation

1. Go to **new meeting** (from the main page, click the second link in the left) and fill the data.



Logged in as albert.bellonch. Logout

Second Life Thesis

Create meeting

Meeting ID:	JAVA-DUM
Description:	Java for dummies
Edition:	☑ Only by the creator
Start time:	11/04/2011 09:00
End time:	11/04/2011 13:00

Activities

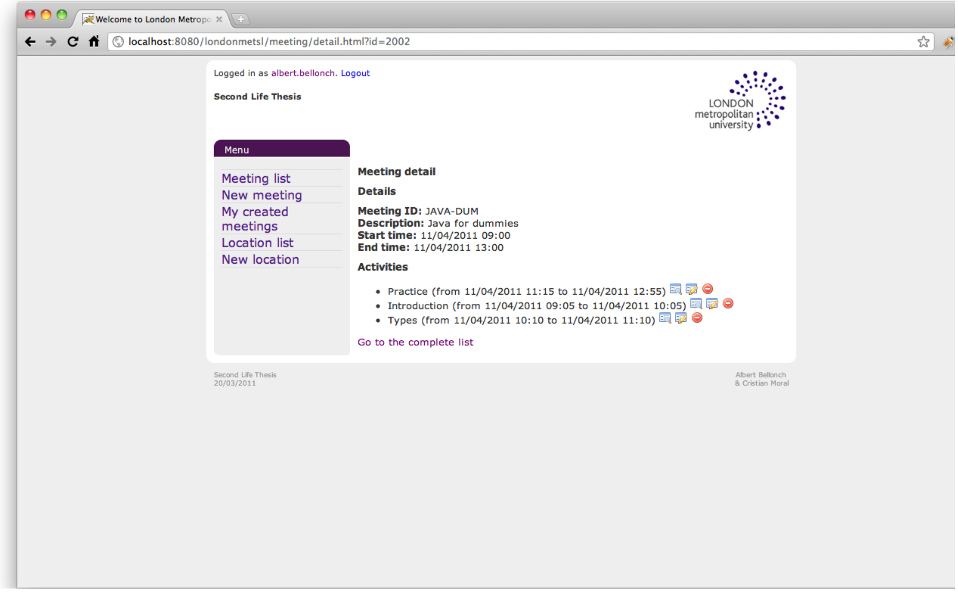
Content	Location	Start time	End time
Introduction	Economy 101	11/04/2011 09:05	11/04/2011 10:05
Types	Economy 101	11/04/2011 10:10	11/04/2011 11:10
Practice	Biology 101	11/04/2011 11:15	11/04/2011 12:55

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 42. Meeting creation in Web User Application: Step 1

2. Your meeting will be created, and its detail will be shown.



Logged in as albert.bellonch. Logout

Second Life Thesis

Meeting detail

Details

Meeting ID: JAVA-DUM
Description: Java for dummies
Start time: 11/04/2011 09:00
End time: 11/04/2011 13:00

Activities

- Practice (from 11/04/2011 11:15 to 11/04/2011 12:55)
- Introduction (from 11/04/2011 09:05 to 11/04/2011 10:05)
- Types (from 11/04/2011 10:10 to 11/04/2011 11:10)

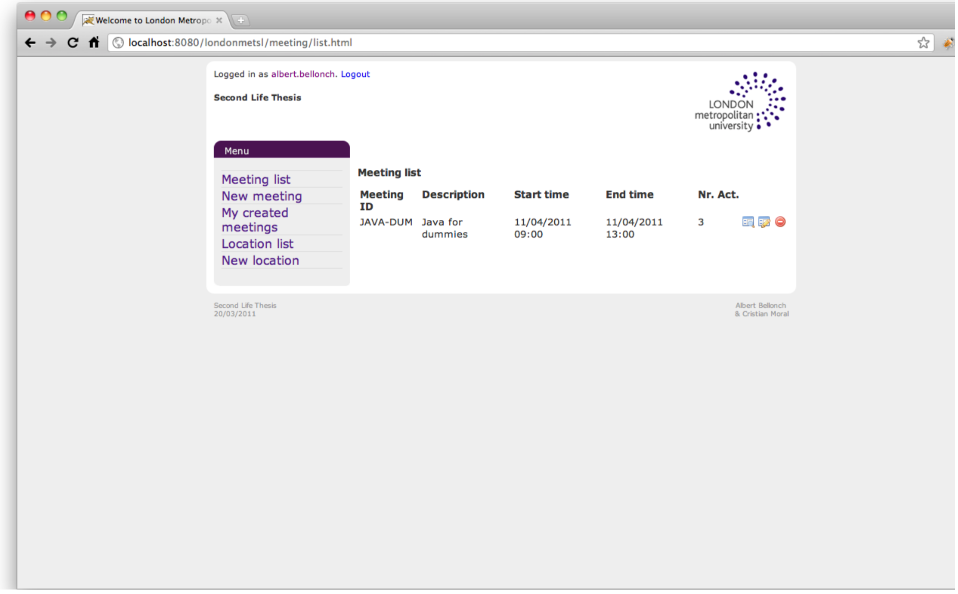
Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 43. Meeting creation in Web User Application: Step 2

4.3.7.5 Meeting management

1. Go to the **meeting list** (from the main page, click the first link in the left).



Welcome to London Metropo

localhost:8080/londonmetsl/meeting/list.html

Logged in as albert.bellonch. Logout

Second Life Thesis

Menu

- Meeting list
- New meeting
- My created meetings
- Location list
- New location

Meeting list

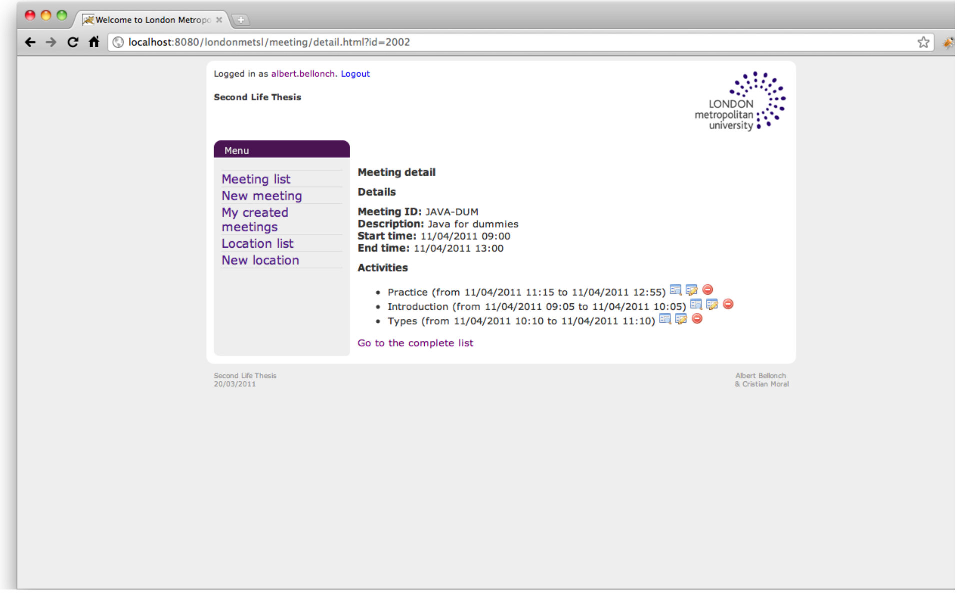
Meeting ID	Description	Start time	End time	Nr. Act.	
JAVA-DUM	Java for dummies	11/04/2011 09:00	11/04/2011 13:00	3	▤ ✎ ⊖

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 44. Meeting management in Web User Application: Step 1

2. Click the ▤ button in order to see its detail (first capture), the ✎ button in order to edit the meeting (second capture), or the ⊖ button in order to delete it.



Welcome to London Metropo

localhost:8080/londonmetsl/meeting/detail.html?id=2002

Logged in as albert.bellonch. Logout

Second Life Thesis

Menu

- Meeting list
- New meeting
- My created meetings
- Location list
- New location

Meeting detail

Details

Meeting ID: JAVA-DUM
Description: Java for dummies
Start time: 11/04/2011 09:00
End time: 11/04/2011 13:00

Activities

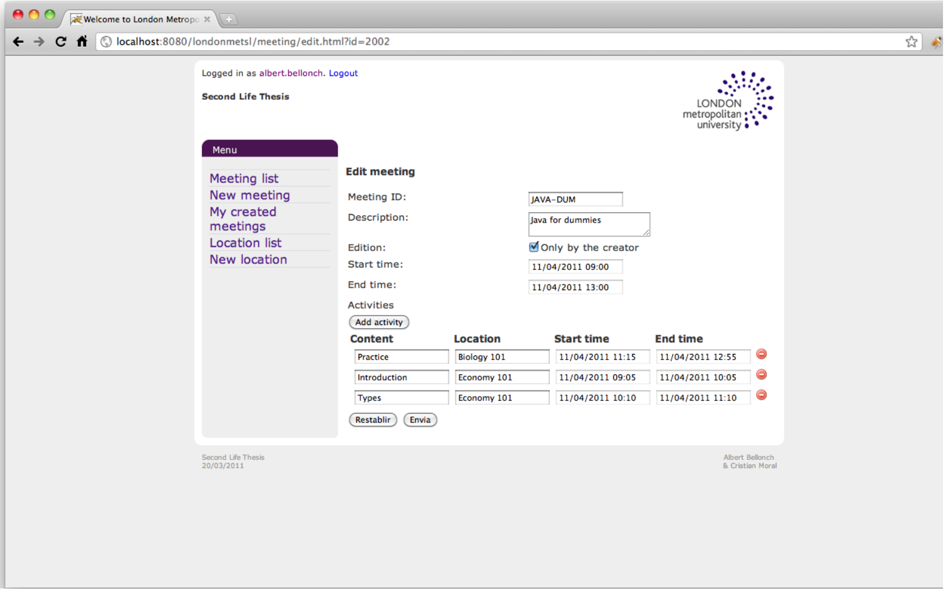
- Practice (from 11/04/2011 11:15 to 11/04/2011 12:55) ▤ ✎ ⊖
- Introduction (from 11/04/2011 09:05 to 11/04/2011 10:05) ▤ ✎ ⊖
- Types (from 11/04/2011 10:10 to 11/04/2011 11:10) ▤ ✎ ⊖

Go to the complete list

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 45. Meeting management in Web User Application: Step 2



Logged in as albert.bellonch. Logout

Second Life Thesis

Edit meeting

Meeting ID: JAVA-DUM

Description: Java for dummies

Edition: ☑ Only by the creator

Start time: 11/04/2011 09:00

End time: 11/04/2011 13:00

Activities

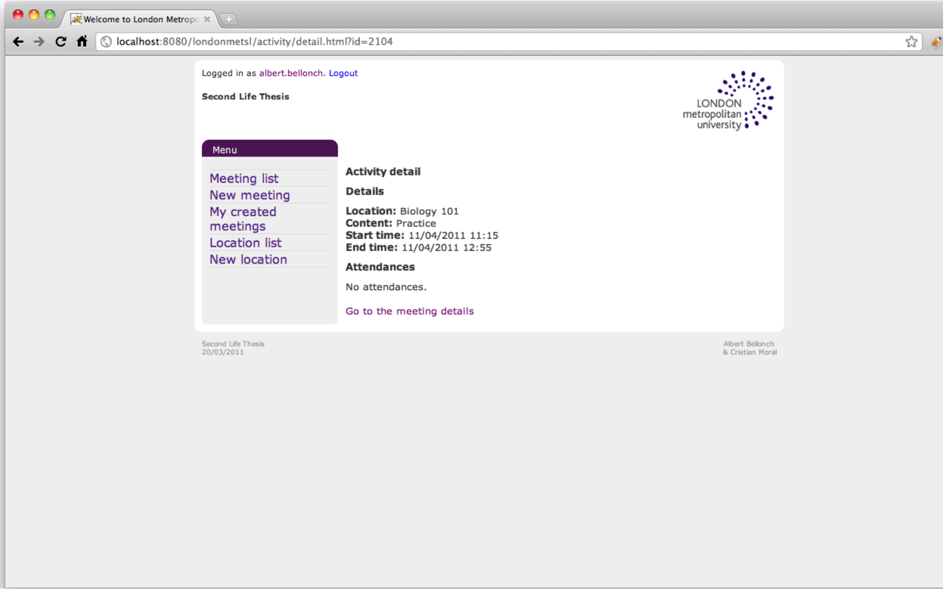
Content	Location	Start time	End time
Practice	Biology 101	11/04/2011 11:15	11/04/2011 12:55
Introduction	Economy 101	11/04/2011 09:05	11/04/2011 10:05
Types	Economy 101	11/04/2011 10:10	11/04/2011 11:10

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 47. Meeting management in Web User Application: Step 3

3. In the meeting detail, you can access to the activity detail (first image) or edit an activity (second image), as well as deleting it.


Welcome to London Metropo
localhost:8080/londonmetsl/activity/detail.html?id=2104
Menu
Meeting list
New meeting
My created meetings
Location list
New location

Logged in as albert.bellonch. Logout

Second Life Thesis

Activity detail

Details

Location: Biology 101
Content: Practice
Start time: 11/04/2011 11:15
End time: 11/04/2011 12:55

Attendances

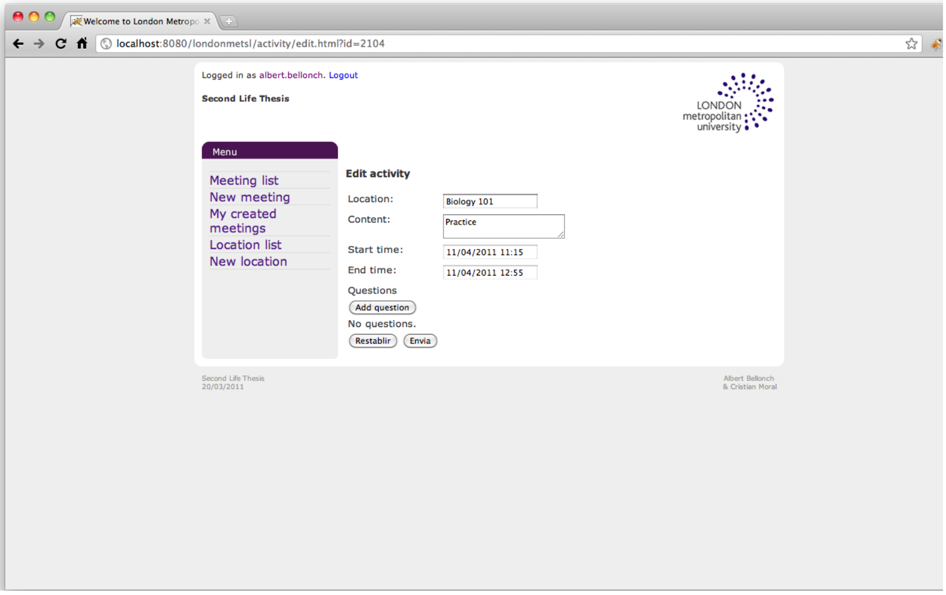
No attendances.

Go to the meeting details

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 48. Meeting management in Web User Application: Step 4



Logged in as albert.bellonch. Logout

Second Life Thesis

Edit activity

Location: Biology 101

Content: Practice

Start time: 11/04/2011 11:15

End time: 11/04/2011 12:55

Questions

No questions.

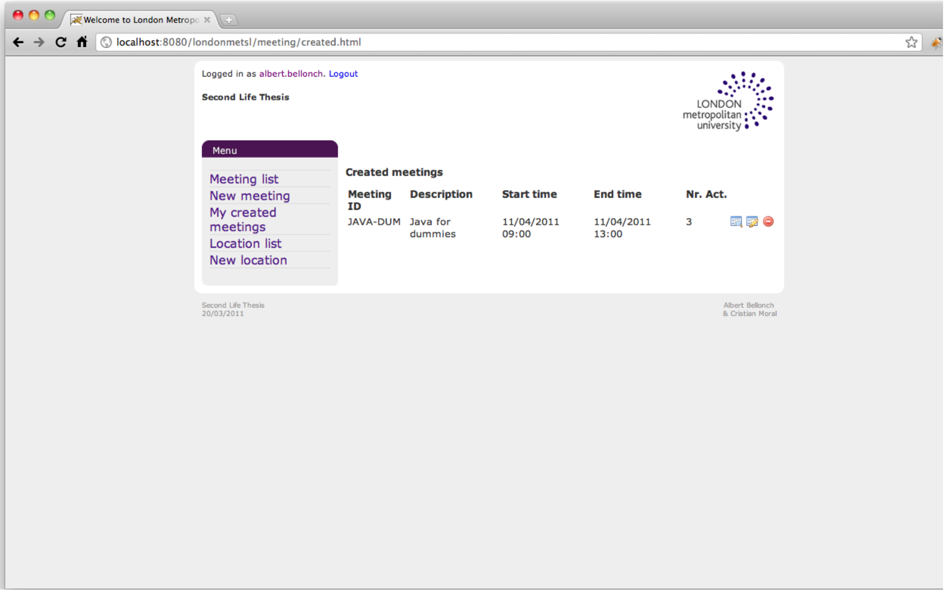
Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 49. Meeting management in Web User Application: Step 5

4.3.7.6 Own meetings

1. Go to **my created meetings** (from the main page, click the third link in the left).


Welcome to London Metropo
localhost:8080/londonmetsl/meeting/created.html
Menu
Meeting list
New meeting
My created meetings
Location list
New location

Logged in as albert.bellonch. Logout

Second Life Thesis

Created meetings

Meeting ID	Description	Start time	End time	Nr. Act.
JAVA-DUM	Java for dummies	11/04/2011 09:00	11/04/2011 13:00	3

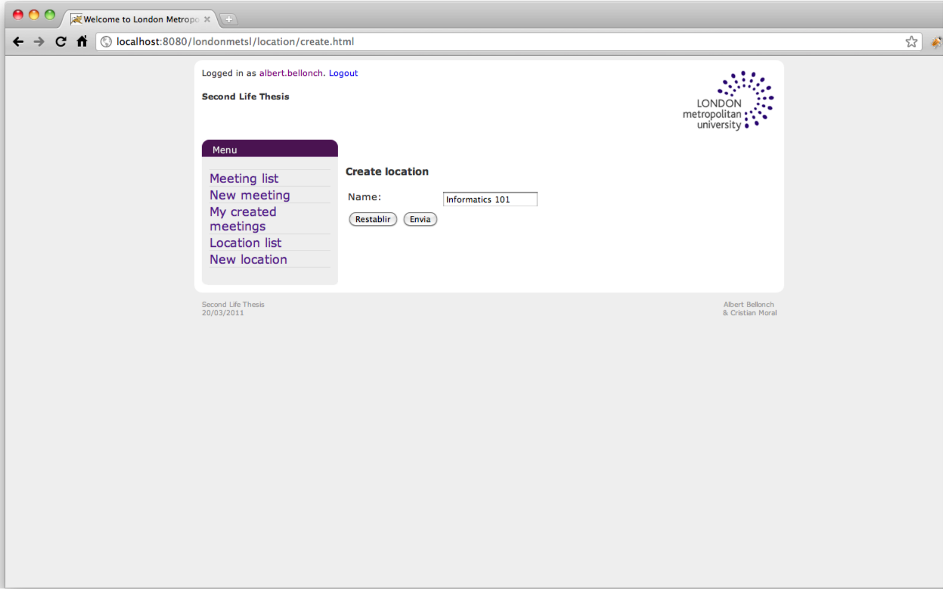
Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 50. Own meetings in Web User Application

4.3.7.7 New location

1. Go to **new location** (from the main page, click the fifth link in the left).



localhost:8080/londonmetsl/location/create.html

Logged in as albert.bellonch. Logout

Second Life Thesis

Create location

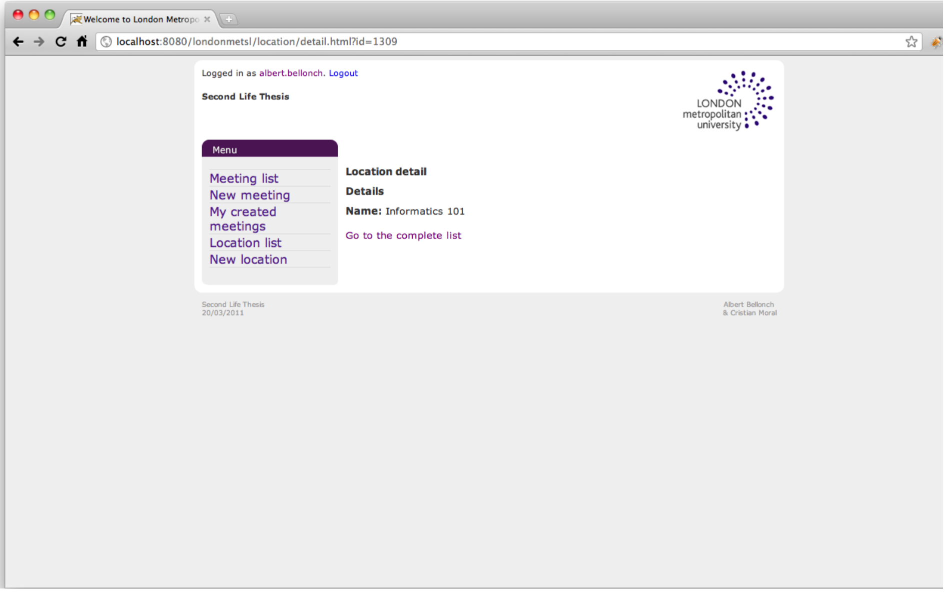
Name: Informatics 101

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 46. Create new locations in Web User Application: Step 1

2. You will be redirected to its detail.



localhost:8080/londonmetsl/location/detail.html?id=1309

Logged in as albert.bellonch. Logout

Second Life Thesis

Location detail

Details

Name: Informatics 101

Go to the complete list

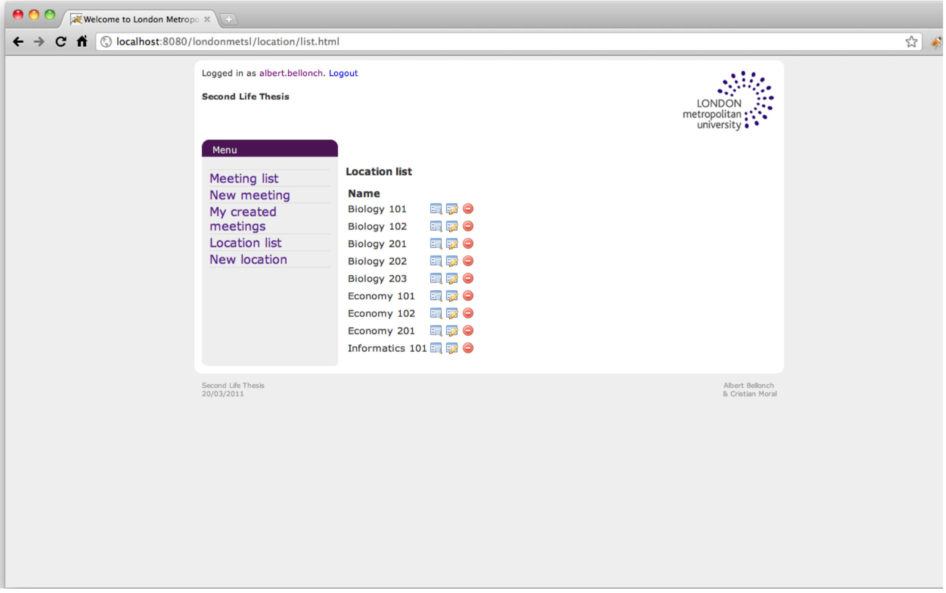
Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 47. Create new locations in Web User Application: Step 2

4.3.7.8 Location management

1. Go to the **location list** (from the main page, click the fourth link in the left).



Welcome to London Metropo
localhost:8080/londonmetsl/location/list.html

Logged in as albert.bellonch. Logout

Second Life Thesis

Location list

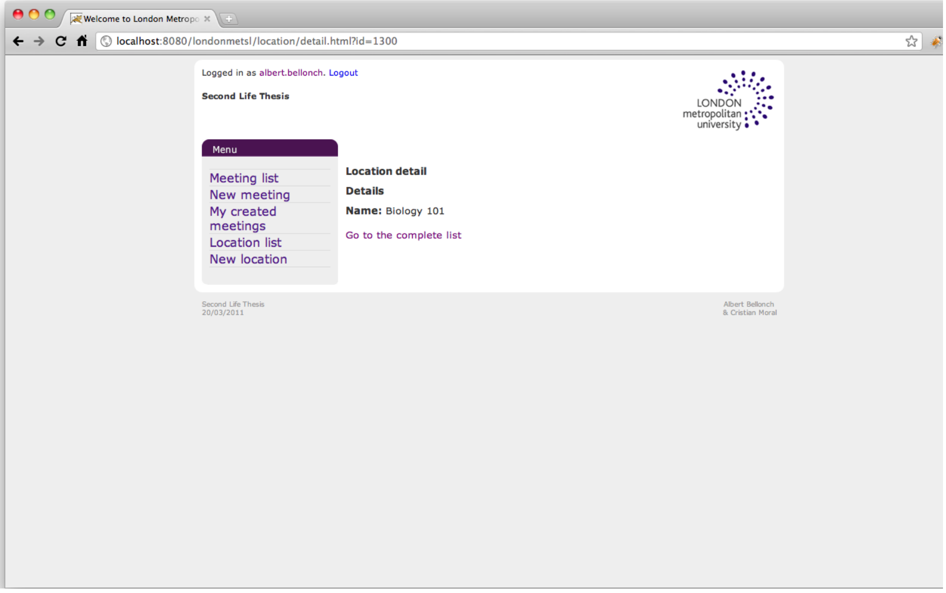
Name	
Biology 101	▤ ✎ ⛔
Biology 102	▤ ✎ ⛔
Biology 201	▤ ✎ ⛔
Biology 202	▤ ✎ ⛔
Biology 203	▤ ✎ ⛔
Economy 101	▤ ✎ ⛔
Economy 102	▤ ✎ ⛔
Economy 201	▤ ✎ ⛔
Informatics 101	▤ ✎ ⛔

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 48. Location management in Web User Application: Step 1

2. Click the ▤ button in order to see its detail (first capture), the ✎ button in order to edit the location (second capture), or the ⛔ button in order to delete it.



Welcome to London Metropo
localhost:8080/londonmetsl/location/detail.html?id=1300

Logged in as albert.bellonch. Logout

Second Life Thesis

Location detail

Details

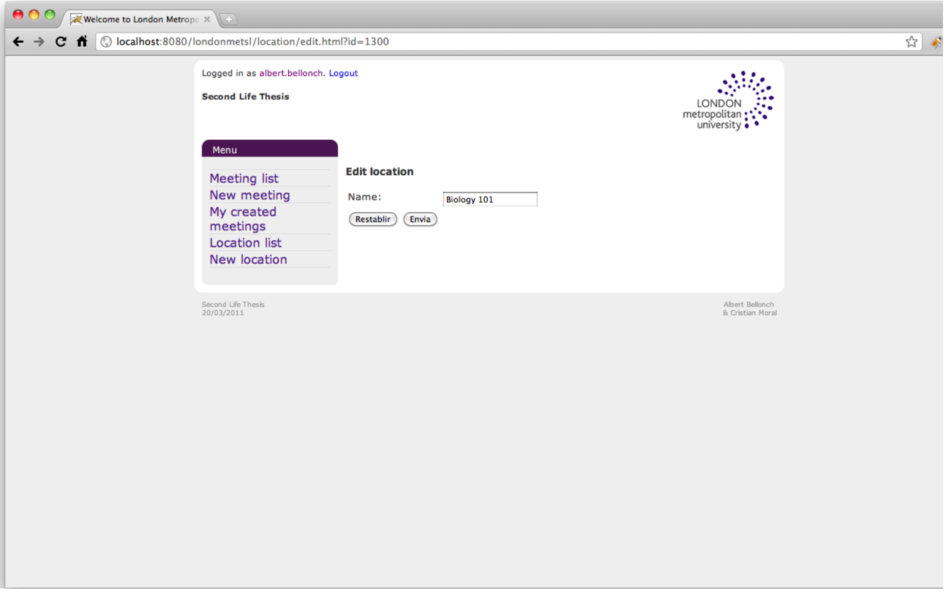
Name: Biology 101

Go to the complete list

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 49. Location management in Web User Application: Step 2



Logged in as albert.bellonch. Logout

Second Life Thesis

Edit location

Name: Biology 101

Second Life Thesis
20/03/2011

Albert Bellonch
& Cristian Moral

Illustration 50. Location management in Web User Application: Step 3

4.4 Server side

As explained in several places of this documentation, Second Life has not the ability to store user-defined data for itself, so if we want to do it we have to provide a supplementary ad-hoc system. In our case, we designed a web application, accessible from both a browser and Second Life itself (via an adequate interface). The purpose of the server is to provide a place to store and retrieve data collected from the Second Life environment.

Several technologies have been used to design the whole architecture: Java as the gluing language; Tomcat as the servlet container; Javascript for client-side interactions; HTML and CSS for the web design and appearance; Velocity for the template rendering; Spring for gluing the application together; Hibernate for storing data without having to write almost any SQL; and so on.

In order to understand this whole environment, totally explained in this Server Side section, some web and Java engineering background is needed, as well as a good knowledge about software engineering patterns. The scope of this whole section, though, is not to be a master class of every technology and piece of knowledge necessary to implement the environment, but to explain the conventions used and how it has been organized. In short, someone with software engineering background is necessary in order to completely understand and extend this system.

4.4.1 Managing the server

Once the whole set of functionalities is designed, implemented and tested (or if you want to directly use the current version in the Subversion), we have to move the web application into a production stage. That is, to place the web application into a server available from anywhere.

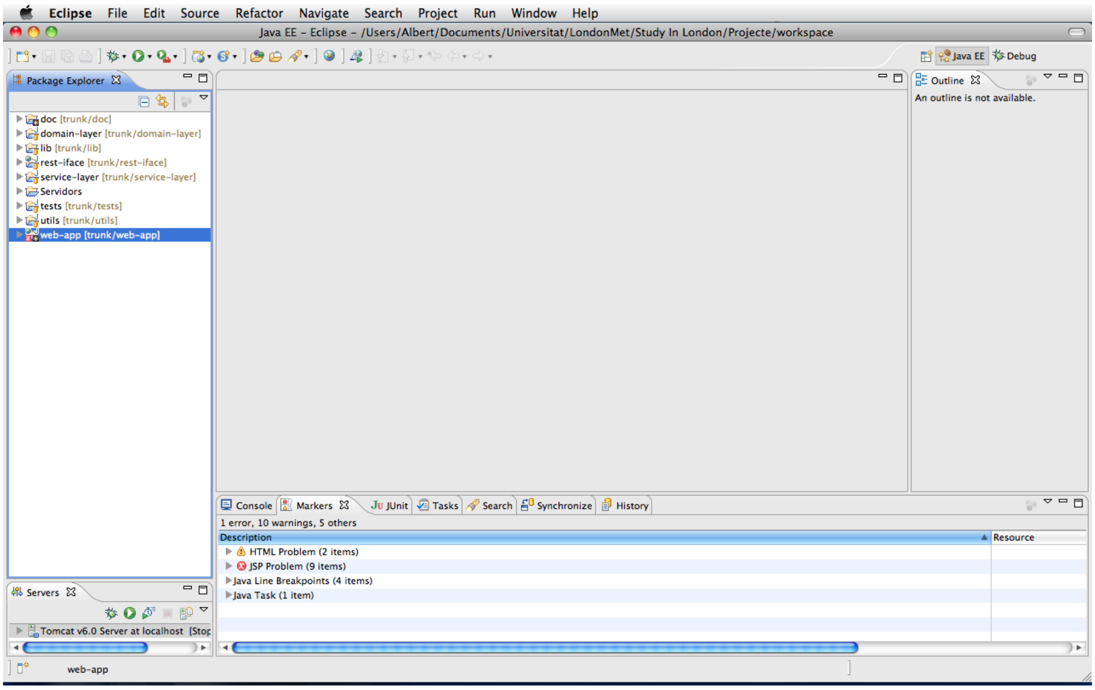
4.4.1.1 Bundling

The first step to take is to bundle our web applications (both the browser-capable one and the REST interface for Second Life) into WARs (22). A WAR file is a Web Application Archive, a particular JAR (Java Archive) used to distribute a collection of JSPs, servlets, Java code, XML files and others, which together constitute the whole web application.

In our case we will have two WARs plus an additional JAR file. The two WARs will undoubtedly be one for the web application and the other one for the REST interface to Second Life. The additional JAR would be the tests bundle, in case we want to do the tests also in the server or we want to populate the database with some data.

In order to do a WAR bundle from a web application from the Eclipse IDE, we simply follow the next steps:

1. Place the focus above the web application you want to export.



Java EE – Eclipse – /Users/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace

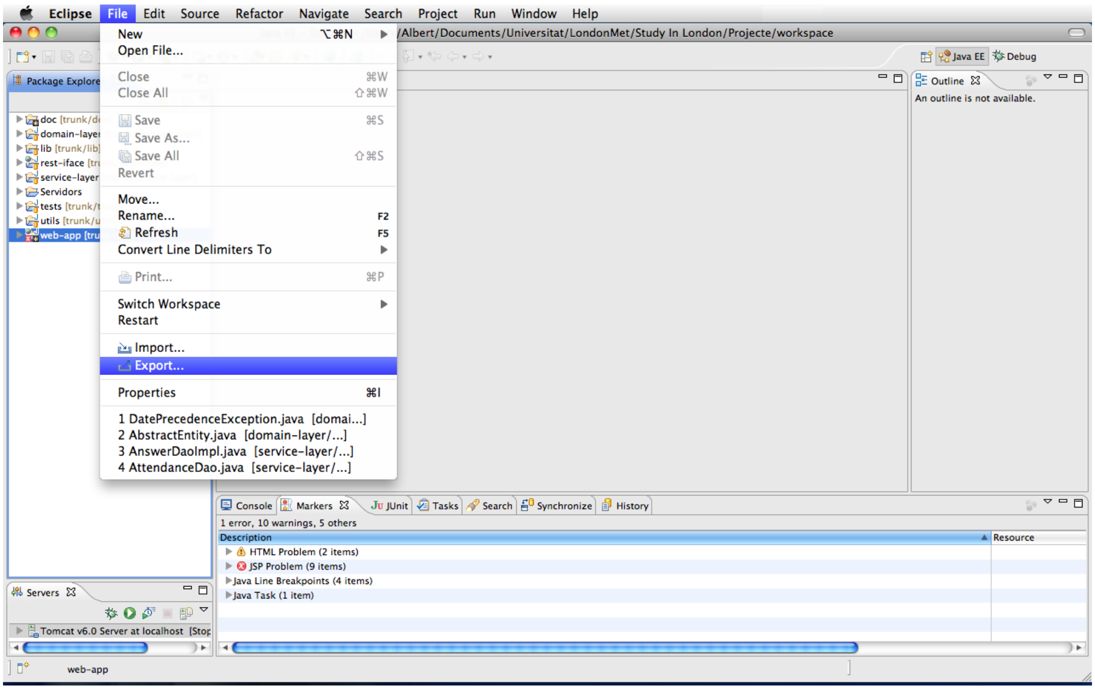
An outline is not available.

1 error, 10 warnings, 5 others

Description	Resource
HTML Problem (2 items)	
JSP Problem (9 items)	
Java Line Breakpoints (4 items)	
Java Task (1 item)	

Illustration 51. Managing the server: Step 1

2. Go to **File > Export**, and click it.



/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace

An outline is not available.

1 error, 10 warnings, 5 others

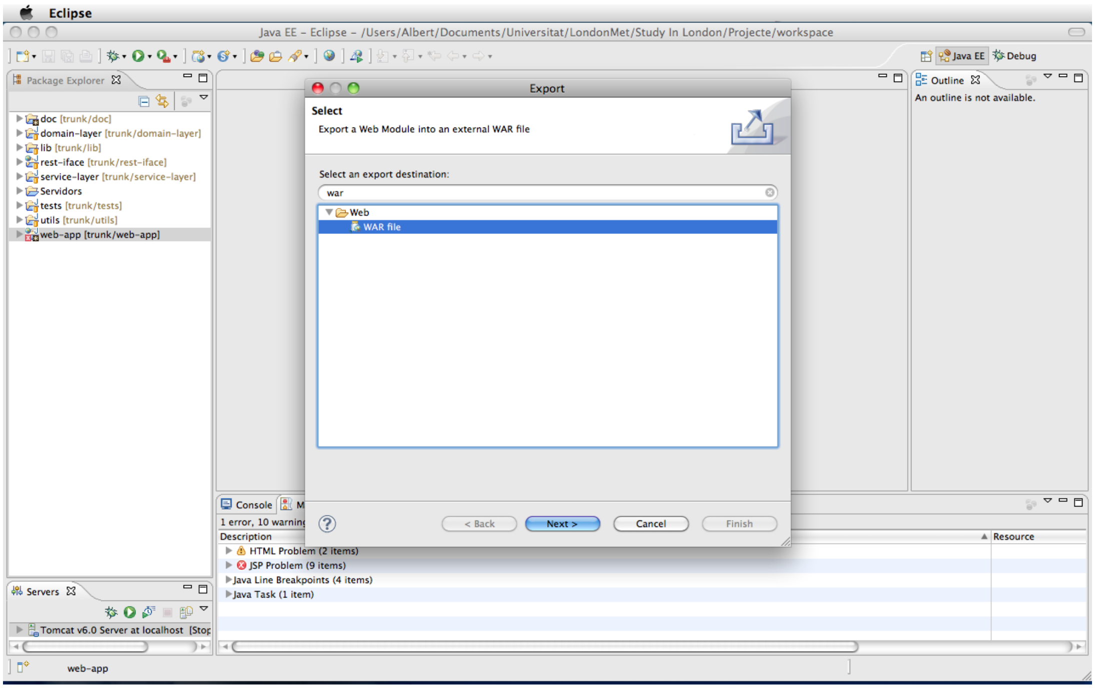
- HTML Problem (2 items)
- JSP Problem (9 items)
- Java Line Breakpoints (4 items)
- Java Task (1 item)

Tomcat v6.0 Server at localhost [Stop

web-app

Illustration 52. Managing the server: Step 2

3. Select **WAR file** from the list and then **Next**.



Java EE – Eclipse – /Users/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace

doc [trunk/doc]
domain-layer [trunk/domain-layer]
lib [trunk/lib]
rest-iface [trunk/rest-iface]
service-layer [trunk/service-layer]
Servidors
tests [trunk/tests]
utils [trunk/utils]
web-app [trunk/web-app]

Select

Export a Web Module into an external WAR file

Select an export destination:

war

An outline is not available.

1 error, 10 warning

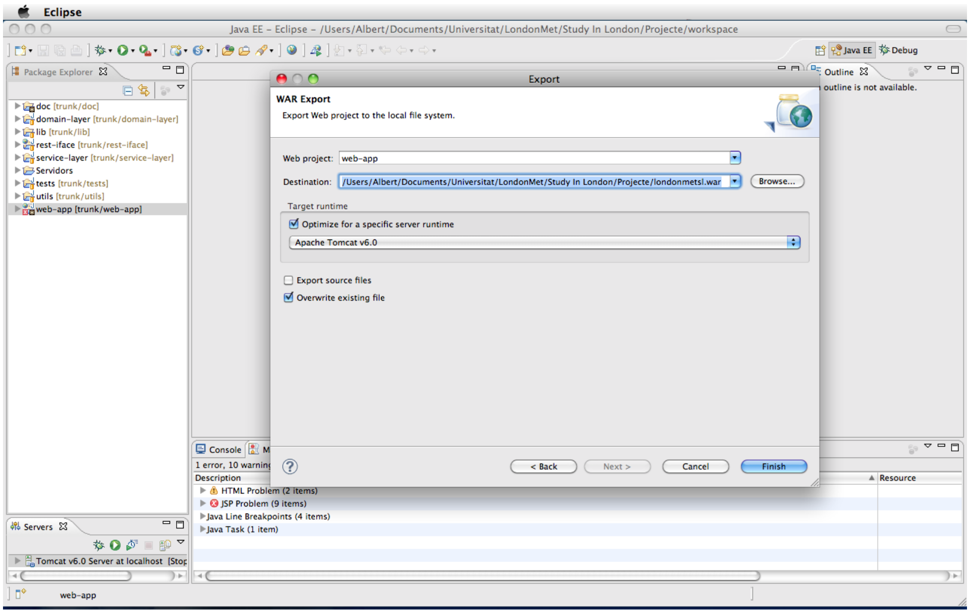
- HTML Problem (2 items)
- JSP Problem (9 items)
- Java Line Breakpoints (4 items)
- Java Task (1 item)

Tomcat v6.0 Server at localhost [Stop

web-app

Illustration 53. Managing the server: Step 3

4. Choose a destination where you want to export your web application to, and make sure to select the first and third checkboxes. Then select **Finish**.



WAR Export

Export Web project to the local file system.

Web project: web-app

Destination: /Users/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/londonmetsl.war

Target runtime

☑ Optimize for a specific server runtime

Apache Tomcat v6.0

☐ Export source files

☑ Overwrite existing file

Illustration 54. Managing the server: Step 4

5. The web application WAR will now be in the selected destination.

Note: *this can vary in operating systems different from Mac OS X.*

Now, to do the same for a JAR file (23):

1. Place the focus above the application you want to export.

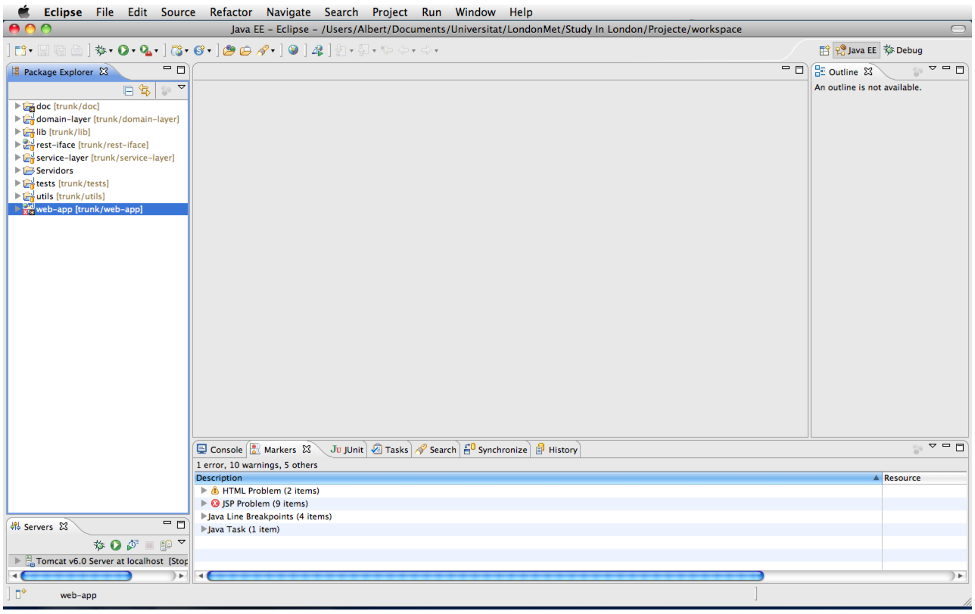


Illustration 55. Managing the server: Step 5

2. Go to **File > Export**, and click it.

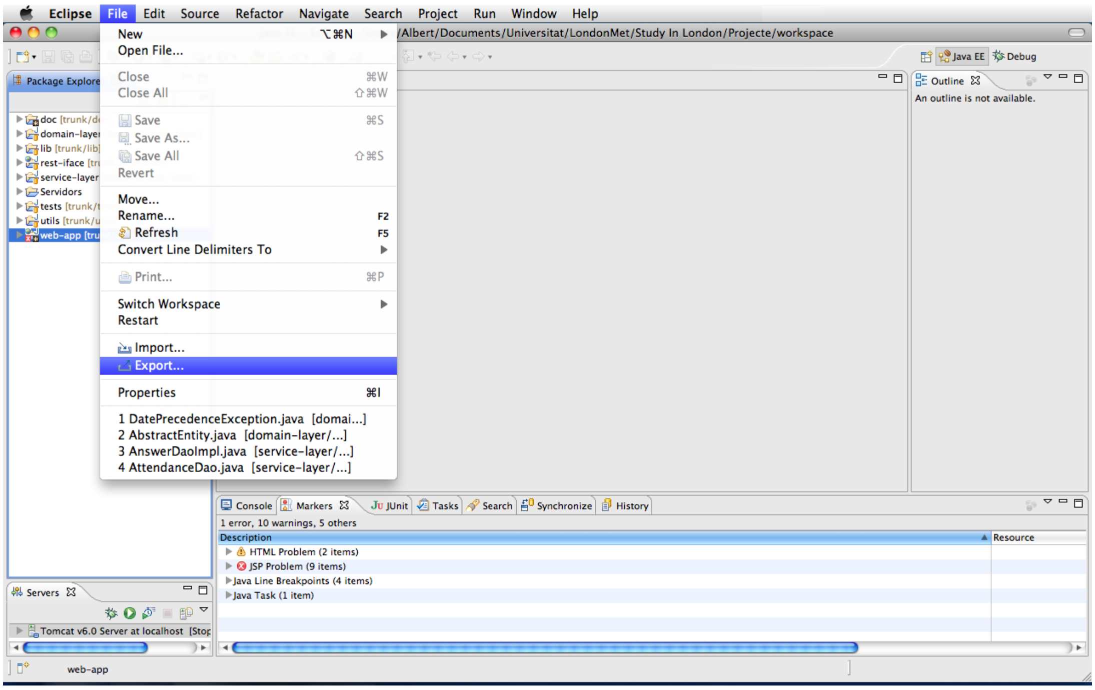
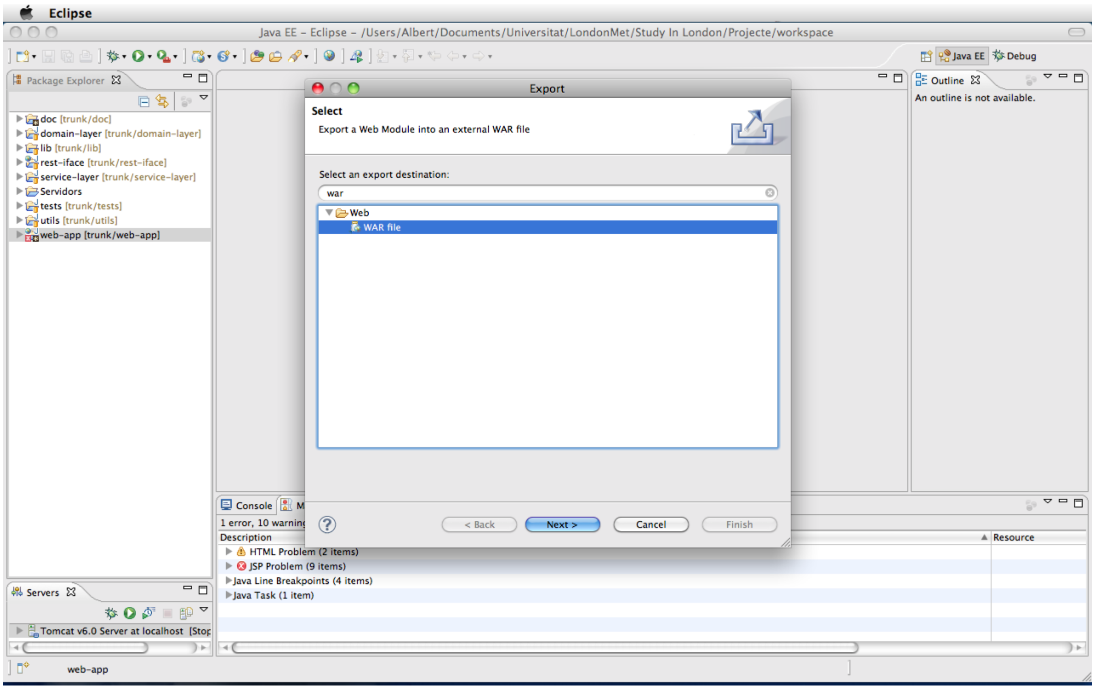

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help
/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace
New ⌥⌘N
Open File...
Close ⌘W
Close All ⇧⌘W
Save ⌘S
Save As...
Save All ⇧⌘S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... ⌘P
Switch Workspace
Restart
Import...
Export...
Properties ⌘I
1 DatePrecedenceException.java [domai...]
2 AbstractEntity.java [domain-layer/...]
3 AnswerDaoImpl.java [service-layer/...]
4 AttendanceDao.java [service-layer/...]
Package Explorer
Outline
An outline is not available.
Java EE
Debug
Console Markers JUnit Tasks Search Synchronize History
1 error, 10 warnings, 5 others
Description
Resource
HTML Problem (2 items)
JSP Problem (9 items)
Java Line Breakpoints (4 items)
Java Task (1 item)
Servers
Tomcat v6.0 Server at localhost [Stop
web-app

Illustration 56. Managing the server: Step 6

3. Select **Runnable JAR file** from the list and then **Next**.


Export

Select

Export a Web Module into an external WAR file

Select an export destination:

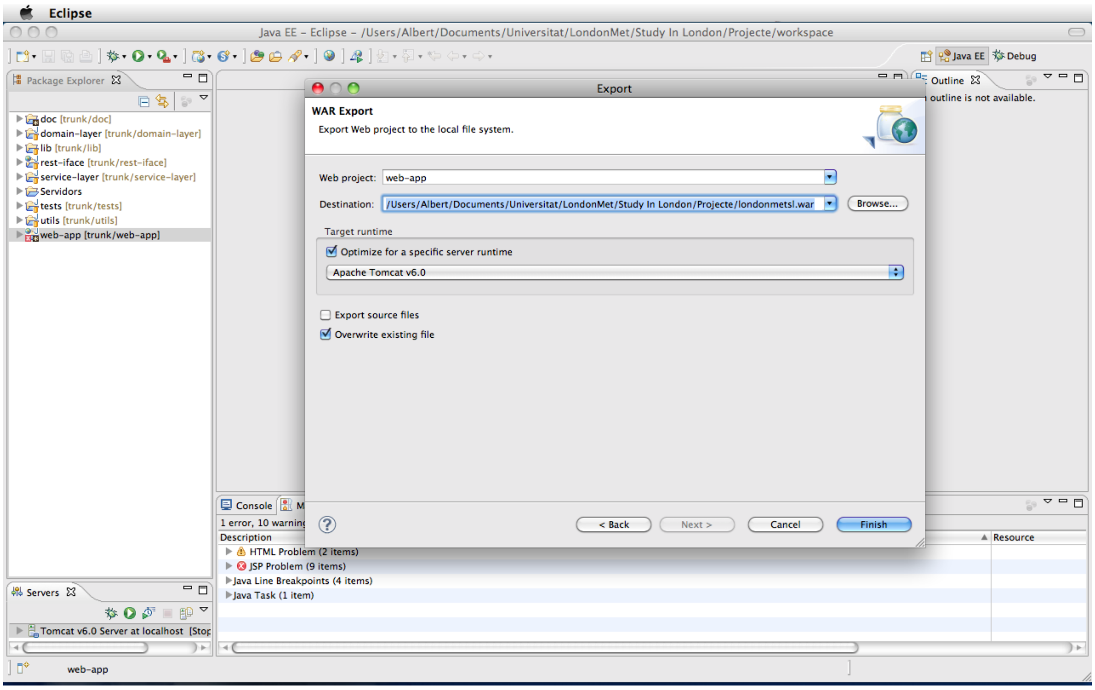
war

- Web
  - WAR file

Eclipse
Java EE - Eclipse - /Users/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace
Package Explorer
doc [trunk/doc]
domain-layer [trunk/domain-layer]
lib [trunk/lib]
rest-iface [trunk/rest-iface]
service-layer [trunk/service-layer]
Servidors
tests [trunk/tests]
utils [trunk/utils]
web-app [trunk/web-app]
< Back
Next >
Cancel
Finish
Outline
An outline is not available.
Java EE
Debug
Console
1 error, 10 warnings
Description
Resource
HTML Problem (2 items)
JSP Problem (9 items)
Java Line Breakpoints (4 items)
Java Task (1 item)
Servers
Tomcat v6.0 Server at localhost [Stop
web-app

Illustration 57. Managing the server: Step 7

4. Choose a destination where you want to export your application to, and select **Package required libraries into generated JAR** in the optionbox. Then select **Finish**.


Eclipse
Java EE – Eclipse – /Users/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/workspace
Package Explorer
doc [trunk/doc]
domain-layer [trunk/domain-layer]
lib [trunk/lib]
rest-iface [trunk/rest-iface]
service-layer [trunk/service-layer]
Servidors
tests [trunk/tests]
utils [trunk/utils]
web-app [trunk/web-app]
Servers
Tomcat v6.0 Server at localhost [Stop
Outline
outline is not available.
Java EE
Debug
Console
1 error, 10 warning
Description
HTML Problem (2 items)
JSP Problem (9 items)
Java Line Breakpoints (4 items)
Java Task (1 item)
Resource
web-app

Export

WAR Export

Export Web project to the local file system.

Web project: web-app

Destination: /Users/Albert/Documents/Universitat/LondonMet/Study In London/Projecte/londonmetsl.war Browse...

Target runtime

☑ Optimize for a specific server runtime

Apache Tomcat v6.0

☐ Export source files

☑ Overwrite existing file

< Back Next > Cancel Finish

Illustration 58. Managing the server: Step 8

5. The application JAR will now be in the selected destination.

With these simple steps applied to both web applications, and optionally to the test application, we have the three required files for the server to run.

4.4.1.2 Choosing a server

Now we need a remote server, able to run the Tomcat servlet container and also to have a MySQL database. In the development stage, in order to perform tests against a remote server, we used a server from **Linode** (24), a Web Application hosting and cloud computing provider which is a standard in the community. Particularly, we chose to have our server in London, as most interactions to our application will be from there.

This step is up to whoever is in charge to balance economical and programmatic needs.

4.4.1.3 Connecting to the server

As the server is normally a Unix-based computer, we usually connect to it via the command shell. Though, the server is remote, so we need a tool in order to manage it as if it was local. The standard tools in order to achieve this goal are **SSH** (Secure Shell) and **SCP** (Secure Copy) (25). The former makes us able to manage the server through the command line, and the latter allows file transfers between our computer and a remote one. Extended information is present in the bibliography.

In order to connect to the server, simply do:

```
user> ssh [user]@[host];
```

Where **[user]** is our user in the server, **[domain]** is the server's hostname. Once this done, we will be connected into that server via a command line.

```
londonmetsl-server>
```

4.4.1.4 Configuring the server

The following step is to configure the server. There are only two important steps to follow in order to have a correctly configured server: setting Tomcat up and configuring a MySQL database. Normally, Tomcat is already set up and running without further configuration needed (additional notes on Tomcat are present in the bibliography), so we will focus on configurating MySQL.

MySQL is one of several relational database management systems, and one of the most used ones. Assuming MySQL is correctly configured, we now need to set up a database with the adequate parameters. In the **domain-layer** project unit, if we take a look at **src/main/resources/jdbc.properties**, we will see that we are using a MySQL connector, and also that the database is local (the host is **localhost**) and its name is **londonmetsl**. Username and password are also **londonmetsl**. You can freely change this database name, username and password, but change it also in **jdbc/properties**.

In order to set up a database suitable for these parameters, we have to:

1. Log into the MySQL command shell (you will be prompted with MySQL root's password)

```
user> mysql -u root;
```

2. Create the database.

```
mysql> create database londonmetsl;
```

3. Grant permissions to that database to a newly created user, specifying also its password (please note that we delimit our user only to a local environment by using "@localhost"; he cannot access remotely).

```
mysql> grant all on londonmetsl.* to londonmetsl@localhost
identified by "londonmetsl";
```

4. Finally exit and you will return to the system command line.

```
mysql> exit

user>
```

4.4.1.5 Transferring the bundles

Once the server is chosen and properly configured, we finally need to transfer our bundles so that they can be deployed in the server, allowing us to connect to that server. As the servlet container is a Tomcat (it is specified before), we have a special directory where bundles are automatically deployed after we put them in there: **[tomcat-path]/webapps**. If we put a bundle named **mock.war** in there, then we will be able to access it via browser with **http://[domain]/mock**.

Let's transfer those bundles now. Analogically with **SSH**, we will use **SCP**, Secure Copy, which will allow us to transfer a file from our computer to the server. Assuming we are in the directory having both WARs for **londonmetsl** (to be accessible via browser) and **lmsl-rest** (to be accessible via Second Life), and assuming we have were the Tomcat folder is in our server, we only have to do:

```
user> scp *.war [user]@[host]:[tomcat-path]/webapps
```

Additionally, if we want to transfer the test JAR to our home (it does not make sense to have it inside the Tomcat folder), we can do (please note that "~" stands for our home folder:

```
user> scp *.jar [user]@[host]:~
```

4.4.1.6 Checking everything

Once the files are transferred, please wait for about a minute (Tomcat has to unbundle the WAR files and, the first time, create the whole database schema) and go to **http://[domain]/londonmetsl**. You should see the interface of the web application, and be able to register and create, update or delete meetings, activities and locations. Also, the server should now be receiving information from Second Life (whose script has to be configured to transfer it all to **http://[domain]/lmsl-rest**).

4.4.2 Three-tier architecture

The other split that allows us to comprehend the whole design is the split by application tiers. This architecture, generally known as **multi-tier** (26), is a client-server architecture in which presentation, application processing and data management are logically separate processes.

If we take a look to the figure about architecture overview, we can distinguish these three tiers into dark blue panels separated with light blue vertical bars. In the **server part**, each tier has several **modules** (some are in more than one tier at once) and communicates with one or more modules. It must be noted that every **server module** is a project specifically design for a purpose.

4.4.2.1 Presentation Tier

The first tier is the **presentation**. Here the server deals with the data sent from the user, extracts it into an internal-application suitable format, and delegates to the application tier. Finally, transforms the data return from the application tier into user-understandable data. We have four modules in this tier.

4.4.2.1.1 Web Application

In charge of receiving HTTP requests, extracting them into a suitable format for the core application, getting returned data from the same core application and transforming it into HTTP responses. In this tier, this module only covers the HTTP requests and responses part.

Code

For each code unit, its name, package and description will be shown.

<table>
<tr><td><b>Controllers</b></td><td>uk.ac.londonmet.secondlife.web.controllers</td></tr>
<tr><td colspan="2">Several controllers, each one for a domain model object, in charge of show lists of the object, showing an object detail or allowing its removal or edition. The lifecycle of each action in a concrete controller is simple: parameter validation if appliable, a call to the core application in order to delegate the operations to the data, and a final stage of showing data through HTML pages.</td></tr>
<tr><td><b>Filters</b></td><td>uk.ac.londonmet.secondlife.web.filters</td></tr>
<tr><td colspan="2">One dedicated filter, acting as a <i>policeman</i> allowing requests to be done or not in function of the user making the request and the security policy found in <i>resources/authentication.properties</i>.</td></tr>
<tr><td><b>Mail</b></td><td>uk.ac.londonmet.secondlife.web.mail</td></tr>
<tr><td colspan="2">Mail management code. Three different code files:
• <b>MailController</b>, in charge of sending activation mails but expandible in case of other future mail use cases
• <b>MailBuilder</b>, in charge of building an HTML mail content based on certain parameters, by using templates
• <b>MailSender</b>, in charge of simply sending an HTML mail content to a destination</td></tr>
<tr><td><b>Security</b></td><td>uk.ac.londonmet.secondlife.web.security</td></tr>
<tr><td colspan="2">Security code. It contains a session handler to manage the user session when logging in and out, and validating it while accessing different web resources.</td></tr>
<tr><td><b>Views</b></td><td>uk.ac.londonmet.secondlife.web.views</td></tr>
<tr><td colspan="2">Presentation code, it contains both a view utilities helper, which allows us to treat different formats and errors, and a JSP template view, an intermediate between the controllers and the rendering stage.</td></tr>
</table>

Resources

For each resource unit, its name, directory and description will be shown.

<table>
<tr><td><b>Mail templates</b></td><td>mail and mail/*</td></tr>
<tr><td colspan="2">Mail templates' folder, contains one template for each different use case. For the moment we only have one template, but the folder is expandable to the number of templates we would like to have.</td></tr>
<tr><td><b>Authentication properties</b></td><td>authentication.properties</td></tr>
<tr><td colspan="2">A list of the clearance needed for each path. Its structure is:
<i>{path}={role}</i>
Where <i>path</i> is the path within the web application to which we want to give access, and <i>role</i> is the minimum role needed to access that path. The list of roles is:
• <i>anyone:</i> Anyone can access the path, either registered or not
• <i>web-user:</i> Only registered web users (and admins) can access
• <i>web-admin:</i> Only registered web admins can access
If several rules apply for one path, the first matched one will be the rule applied. If no rules apply for one path, the last rule will be the one applied.</td></tr>
<tr><td><b>Web application context</b></td><td>webapp-applicationContext.xml</td></tr>
<tr><td colspan="2">Spring Framework's (27) configuration for this web application. It contains the description of where to find key controllers and several other source code.
Spring Framework will be treated in the next <i>technologies and libraries</i> section.</td></tr>
</table>

Web

A **web** folder is also present in this project, which contains all the static files needed for our web application. This folder will be directly available (except its **WEB-INF** and **META-INF** subfolders) via HTTP; for example, through a browser.

- **css:** Cascade StyleSheets; files in the CSS language which describe presentation semantics, that is, look and formatting of every HTML tag present in the delivered HTML files.
- **img:** Folder containing every image to be used from the web application.
- **js:** JavaScript code; files in that language which describe client-located interactions, allowing improvements in the UI and dynamic web pages.
- **META-INF:** some metainformation, not interesting for our case.
- **WEB-INF:** mainly important for JSP (Java Server Pages), which allow us to have dynamic webpages, that is, pages which can be built in function of certain data (provided from the user itself, a database, etc.). Configuration files and the **favicon** (the icon to be shown along with the URL in the browser) are present too.

Technologies and libraries

For each technology or library used, its name, version and description will be shown.

Apache Tomcat (28)	**6.0**
Open source servlet container, developed by Apache. Implements Java Servlet and JavaServerPages or JSP, is one of the most used web servers for Java.	
Spring Framework	**2.5.6**
Open source framework for Java, used to accelerate productivity and time to market. It really eases the programming of Web applications.	
Apache Velocity	**1.6.4**
Open source project for Java, developed by Apache. It basically is a Java-based template enginge that provides a template engine capable of using and rendering data objects.	
Java Mail	**1.4.3**
Java API used to send and receive email via different mail protocols, such as SMTP, POP3 and IMAP.	
GSon	**1.5**
Java API, developed by Google, used to convert Java Objects into its JSON representation, and viceversa.	

4.4.2.1.2 REST Interface

In charge of receiving mutually agreed data structures, extracting them into a suitable format for the core application, getting return data from the same core application and transforming it into mutually agreed data structures again. In this tier, this module only covers the mutually agreed data structures.

Code

For each code unit, its name, package and description will be shown.

Filters	`uk.ac.londonmet.secondlife.rest.filters`
Only contains a unique controller, in charge of verifying that whoever sends that request has the correct username and password.	
Resources	`uk.ac.londonmet.secondlife.rest.resources`
Contains one controller for each REST use case, except **GenericSecondLifeResource**, a generic controller from which every controller inherits common methods. Every controller contains one action that parses the JSON data received (through either GET or POST), validates the data, interacts with the service layer in order to do the whole business logic, and finally creates a response returned to the sender. The list of use cases is as follows:	

- Access control for an avatar, checking if it is registered or not
- Introduction of an answered question for an avatar, a location and a question of that location's active test
- Avatar registration by using its ID, name, birthday and locale
- Presence control, by receiving the current list of presences and checking which are still there, which are new and which are gone.
- Retrieval of a quiz information for a specific location and avatar. The response for each question is return.
- Reset of a quiz for an avatar, which deletes every information stored to the moment about a quiz for a specific avatar.

Technologies and libraries

For each technology or library used, its name, version and description (if not explained before) will be shown.

Apache Tomcat	**6.0**
Jersey	**1.4**
Java API, developed by Sun (now Oracle), for RESTful Web Services	
GSon	**1.5**

4.4.2.1.3 Tests

In charge of testing both previous two modules. It acts as each one of the clients and allows us to check that every use case is correctly implemented and returns the expected output data for a specific input.

Code

For each code unit, its name, package and description will be shown.

DB	`uk.ac.londonmet.secondlife.test.db`
Package wich allows the tests to communicate to the database we use through the service layer. Thanks to **AbstractDBTest**, we can begin transactions and commit or rollback them, and in general use Spring to ease that connection. Moreover, with **InitDBTest** we can populate the database with data present in a resource, the *InitDB.xml* file. We use a specific library to do that: **DBUnit**.	

REST	`uk.ac.londonmet.secondlife.test.rest`
In this package we have different tests for every REST use case. This way we can check that every use cas involving interaction between server and Second Life has no errors and goes smoothly fine. We have a **GenericSecondLifeTest** which allows us to connect with our server, so that the whole tests can use this encapsulated function easily. For the rest of the tests, we check that the returned data from the server is the one expected.	
Test executor	`uk.ac.londonmet.secondlife.test`
Class that allows us to choose which test we want to use at each time. We can extract it into an executable JAR.	

Resources

For each resource unit, its name, directory and description will be shown.

DBUnit data	dbunit/InitDB.xml
Data used to test the application via web in a more real context. It is in XML format; each tag corresponds to an object instance, and each attribute of a tag corresponds to an object attribute.	

Technologies and libraries

For each technology or library used, its name, version and description (if not explained before) will be shown.

Spring Framework (27)	**2.5.6**
Hibernate (29)	**3.2.6**
Hibernate annotations	**3.3.1**
Hibernate entity manager	**3.3.2**
JUnit	**3.8.2**
Unit testing framework for the Java programming language, and the most used one.	
DBUnit	**2.2**
JUnit extension to perform unit testing with database-driven applications	

4.4.2.1.4 Utils and libraries

Utilities such code helpers or constants, and libraries used for the presentation tier to work well.

Code

For each code unit, its name, package and description will be shown.

Utils	`uk.ac.londonmet.secondlife.utils`
Package which encapsulates different utils methods used from all the other project units. The list is as follows: • **DateUtils:** date conversions between formats • **JSONHelper:** conversions between data objects and JSON text • **ListUtils:** utils related to the List class • **MD5Helper:** helper used to hash a string, in our case always the password • **RefConstants:** some constants • **WebConstants:** some web-related constants	

Technologies and libraries

For each technology or library used, its name, version and description (if not explained before) will be shown.

Gson	**1.5**

This project unit contains one folder for each library pack used in our application. The list with name and version is as follows:

AspectJ	**1.6.9**
DBUnit	**2.2**
Gson	**1.5**
Hibernate	**3.2.6**
Hibernate annotations	**3.3.1**
Hibernate entitymanager	**3.3.2**
Hibernate validator	**3.00**
Jakarta taglibs	**1.1.2**
Java mail	**1.4.3**
Jersey	**1.4**
JUnit	**3.8.2**
MySQL Connector	**5.1.13**
Quartz	**1.8.3**
Spring Framework	**2.5.6**
Velocity (30)	**1.6.4**

4.4.2.2 Application Thiers

The second tier is **application**, the most important tier in our application (and web applications in general), as implements the whole logic. This tier receives refined data from the user, operates it through the business logic and stores it via the third tier: the data management. We have four modules in this tier:

4.4.2.2.1 Web app and REST interface:

Sends to the application suitable data to the service layer. In this tier, it is only covered this part (as explained before, data extraction from the user is done in the first tier).

4.4.2.2.2 Service layer

Fully implements the business logic. Coordinates the application, processes commands and makes logical decisions and evaluations, apart from moving data between layers.

<u>Code</u>

For each code unit, its name, package and description will be shown.

DAOs	`uk.ac.londonmet.secondlife.dao(.impl)`
The *dao* and *dao.impl* packages include the data access object (DAO) interfaces and implementations, in that order. Interfaces define exclusively the header of the available methods for that domain model's object, and implementations show how that methods are actually built. This separation is due to encapsulation and possible future different implementations for other backend systems (for example, a domain model not built with hibernate). For each domain model's object, so, an interface and implementation are present. Each one has different methods which directly apply to the domain model, and are available for an upper level of abstraction: the services. Typical methods in the DAOs are finding an instance by its name, getting all the instances, deleting one instance, and so on. **GenericDao** and **GenericDaoImpl** are a generic interface and a implementation with generic methods so that we do not have to implement them for each domain model's object.	
Services	`uk.ac.londonmet.secondlife.service`
Services make effective business logic in an upper level. We are not accessing data directly anymore but through the DAOs, which encapsule this functionality. This package truly implements the whole business logic of the system. There is one service for each domain model's object, and typical methods are strictly dependent of what use cases are offered related to that object. It is the entry point	

for this *services* project, so both web and REST interfaces access business logic through the services. A **GenericService** exists which, as the DAO's generic classes, encapsulate typical unitary methods.

Resources

For each resource unit, its name, directory and description will be shown.

Services application context	services-applicationContext.xml
Spring Framework's configuration for this services. It basically exposes every single service so that both web and REST interfaces can used them. Spring Framework will be treated in the next *technologies and libraries* section.	

Technologies and libraries

For each technology or library used, its name, version and description (if not explained before) will be shown.

Spring Framework	**2.5.6**
Open source framework for Java, used to accelerate productivity and time to market. It really eases the programming of Web applications.	
Hibernate	**3.2.6**
Object-relational mapping library for the Java Language, providing a framework for mapping an object-oriented domain model to a traditional relational database.	
Hibernate annotations	**3.3.1**
Hibernate extension that allows us to add syntatic metadata to certain classes, avoiding to do it programmatically.	
Hibernate entity manager	**3.3.2**
Hibernate extension that allows us to access a database in a particular unit of work. The entity manager can create, remove, find and update entities.	

4.4.2.2.3 Utils and libraries:

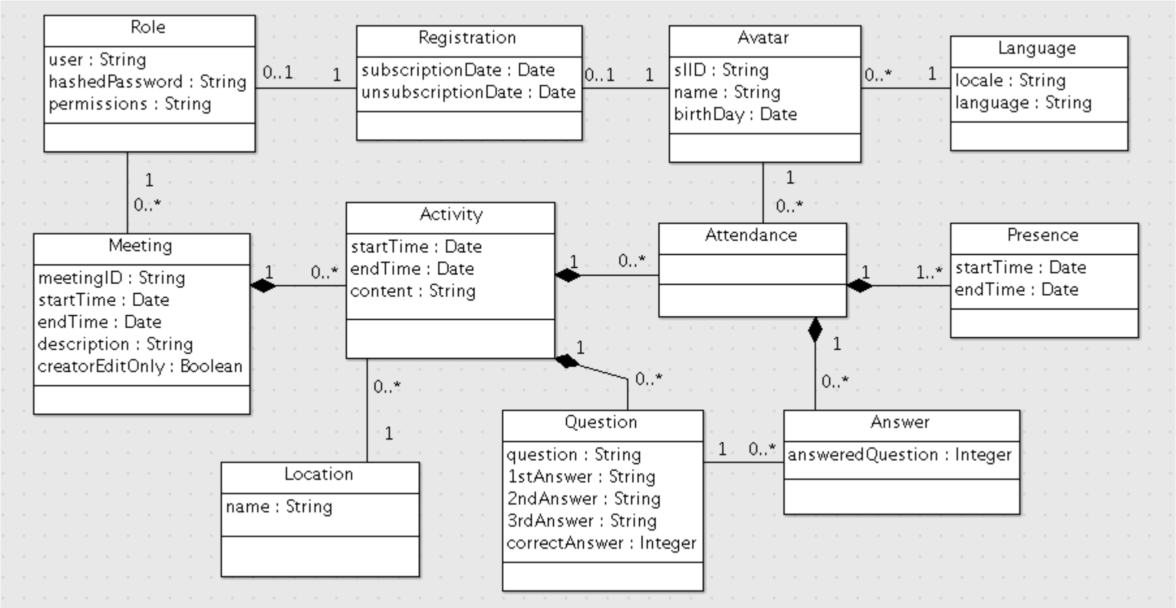
Utilities such code helpers or constants, and libraries used for the application tier to work well. We have only two modules in this tier.

4.4.2.3 Data management Tier

The third tier is **data management**. This tier stores and retrieves the persistent information of the application in a database or a file system. In our case, we use a database and this tier is quite straightforward; as we will see in more detail in posterior sections, we use certain libraries that do the work for us.

4.4.2.3.1 Domain model

Represents the model stored in the database through objects. This way, we can think of objects instead of table records, gaining in perspective and abstraction. Here we can see the relation between the data model classes, which is extracted from the file data model/LondonMetSL-datamodel.zargo in the doc project unit. The file can be opened with the ArgoUML program.



A Registration is a class involving an Avatar and a Role. This way, a user (from Role) is related to an avatar from Second Life (from the Avatar Class). On the one hand, a Role has the username, the password and the permissions (web-user or web-admin). On the other hand, the Avatar stores information from Second Life (its ID, the name, the birthday and the language) so we can relate registered users to the presences in a room sent from Second Life's script.

A user (from Role) can have many meetings (the ones that he/she creates). One Meeting has an ID (understandable from humans, like "JAVA-BEGINNERS" or "CHEMISTRY-ADVANCED"), start and end times (obviously, the end is after the start), a description and a boolean value describing if it can be edited only by the creator or by other web admins too. A meeting has many activities too (Activity class): each one has a start and an end (again, the end is after the start, but also the activity dates are within the meeting date range), a content, and also a location.

For each activity, we have many questions (Question class), which include the question itself, the three options and the correct answer. An activity also has many attendances (Attendance class, when an avatar is in the location of the attendance), which has many presences (Presence class, the time between the avatar enters and leaves) and many answers (Answer class, which responds to a specific Question of the attendance's activity).

Code

For each code unit, its name, package and description will be shown.

Model	`uk.ac.londonmet.secondlife.domain.model`
This package includes the whole data model as Java classes. Every class represents a database table (though, addicional tables may be in the database in case of certain association mappings). Attributes and associations of each class are encapsulated and can only be accessed through *get* or *set* methods, and certain Hibernate Annotations are used to indicate certain characteristics of the class. A generic AbstractEntity class is used to aglutinate every generic behaviour of the data model classes.	
Validation	`uk.ac.londonmet.secondlife.domain.validation`
Package encapsulating every class related to validation management: one exception for each validation exception type. A generic ValidationException is used to encapsulate generic behaviour.	

Resources

For each resource unit, its name, directory and description will be shown.

Data model application context	datamodel-applicationContext.xml
Spring Framework's configuration for this services. It basically describes how to connect with the database, and some other needed classes. Spring Framework will be treated in the next *technologies and libraries* section.	
Hibernate configuration	hibernate.cfg.xml
Hibernate configuration, which includes the mappings of all the classes in the domain model and some other parameters.	
JDBC properties	jdbc.properties
JDBC properties, concretely every parameter related to the database credentials including username, password, driver class and database URL.	

Technologies and libraries

For each technology or library used, its name, version and description (if not explained before) will be shown.

Spring Framework	**2.5.6**
Hibernate	**3.2.6**
Hibernate annotations	**3.3.1**
Hibernate entity manager	**3.3.2**
MySQL connector	**5.1.13**
Connector to the database; in our case a MySQL one.	

4.4.2.3.2 Utils and libraries

Utilities such code helpers or constants, and libraries used for the data management tier to work well.

4.5 Extending the system

In this section we will fully explain how to extend every single part of the system. For every possible extension, the path to be followed will be discussed. Please see examples for each case in order to fully comprehend how it is done.

4.5.1 Web use cases

Adding a new web use case (meaning an interaction between the web user and the server itself) requires a view and a controller, and some other modifications:

- **View:** the design of a certain page must be present as a Java Server Page in the **web/WEB-INF/jsp** page of the **web-app** project unit, in a mix of HTML static tags and JSP dynamic tags. The route of this JSP within the **web/WEB-INF/jsp** folder must be memorized, as it will be used from the controller to render it.
- **Controller:** the controller (all are present in the **uk.ac.londonmet.secondlife.web.controllers** Java package) must expose the page to the user and manage every interaction from him, validations included. We have a method for each interaction: edition, list, detail, and so on. As explained in the architecture section, a typical controller method lifecycle is to receive the HTTP request, validate its parameters, do some business logic through the service layer and finally render the web page. Please note that the folder in which the JSP of the page is saved is used here.
- **Security:** we have to indicate if this web page can be accessed from anyone, a web user or only a web admin. We can do it thanks to the **src/main/resources/authentication.properties** file within the **web-app** project unit. Simply add a line for this web page router and the correct permission. Please note that, unless explicitly specified, every web page can only be accessed from a web user or admin, not for anyone.

4.5.2 REST resources

Adding a new REST use case (meaning an interaction between Second Life and the server itself) requires a controller to do so. This controller, placed in the **uk.ac.londonmet.secondlife.rest.resources** Java package, is specific for a certain path within the whole server, and typically consumes JSON (POST methods) or some parameters (GET methods) and finally delivers JSON (GET methods) or XML (POST methods).

Inside each controller, in a similar way to web controllers, we receive an HTTP request, validate its parameters, interact with the service layer in order to do some business logic, and finally deliver the data. The difference is that we are no longer interacting with a browser but with a Second Life script.

4.5.3 Service layer

As explained in the specific architecture section, the **service-layer** project unit is composed with DAOs and services. In the DAOs, we design and implement methods that directly operate with the data (find all, get one, delete, and so on), and in the services we combine these simple methods in order to implement a whole business logic for each use case.

In order to implement the service layer, so, we have to implement the business logic as a service method, and add eventual DAO operations. Please note, though, that common DAO operations are implemented so that we do not have to rewrite them for addicional data model classes.

4.5.4 Data model

The **data model** project unit is expandable in two ways: adding more classes in order to expand the whole conceptual model, or adding validation exceptions. In order to add more classes, we simply have to put them in the **uk.ac.londonmet.secondlife.domain.model** section, specifying its attributes and relations with other classes (with the help of Hibernate annotations). In the following server startup, the Hibernate engine will detect these changes and add them into the database.

For the validation exceptions, we have to add them to the **uk.ac.londonmet.secondlife.domain.validation** package by extending the

GenericException (we only have to specify addicional attributes and the message redefinition), and that's it. From now on, we will be able to use them when necessary.

4.5.5 Libraries

Adding a library is quite easy and only requires two steps: first adding the library into the **lib** folder, and then linking it from the project using it. Let's see it in more detail:

- Adding a new library into the **lib** folder
  1. Create a new folder into the **lib** folder
  2. Put every single new library into that folder
  3. Just refresh the **lib** project and see the new library there

- Linking the new library from the project using it
  1. Right click on the project that will use the new library
  2. Hit **Properties**
  3. Go to **Java Build Path**
  4. In the **Libraries** tab, hit **Add JARs...** and found the library or libraries in the newly created folder
  5. In the **Order and export** tab, check every single added library
  6. Hit **OK**

4.5.6 Images or web styles

Addicional images and web styles can be put easily into the website. Simply add them into the **web/img** (for images) or **web/css** (for CSS styles) folder in the **web-app** project unit. Please note that you will have to link CSS styles from the generic template (**web/WEB-INF/jsp/template.jsp** file), in the head section of the HTML. This way, that CSS will be available from every single web page of the web application.

4.6 Management and Control Tools

Even if we only were two persons working in the project, which ease very much the communication and synchronization, we have decided to use some tools regarding the management and control of the project in order to adopt good practices and to take profit of the advantages of them. The main aim of these tools is to increase the efficiency of the production and to be of use to the experts by facilitating their work. The main advantages come from the fact that they are accessible from anywhere at any moment maintaining the coherence of the data, which are always updated. In our case we have used all the tools given by Hosting of Projects of Goggle named Google Code[1], firstly because it's free, and because it offers all the tools we needed to carry out our project.

4.6.1 Wiki

Wiki is a piece of software that allows users to freely create and edit Web page content using any Web browser. Wiki supports hyperlinks and has simple text syntax for creating new pages and crosslinks between internal pages on the fly.

In our case we have used this tool to take notes about the evolution project, both for the client side and the server side. We also used it to save important links useful for our projects and finally to save the prototypes of the system and the database.

4.6.2 Issues

The tool named Issues into Google Code is an enhanced Wiki and issue-tracking system for software development projects. It uses a minimalist approach to web-based software project management. Its objective is to help developers to write great software while staying out of the way. One of its better features is that it's almost fully customizable depending on the needs.

The system that Google Code Issues uses is based on tickets. A ticket is a specific task with several characteristics given by default by the tool:

- **Id:** Number of the task. Increase automatically when creating new tickets.
- **Summary**: A summary of the task.
- **Description**: A full description of the task.
- **Status**: Define the status of the task. There are two types of statuses.

[1] http://code.google.com

    - **Open Statuses**:
        - **New**: The task has not had an initial review yet
        - **Accepted**: Someone in the team has accepted to carry out the task.
        - **Started**: Work on the task has begun
    - **Closed statuses**:
        - **Fixed**: The task, which implies coding, has been completed successfully but must be verified.
        - **Verified**: The task work has been verified.
        - **Invalid**: The task is not a valid issue report.
        - **Duplicate**: The task duplicates another task of an existing issue.
        - **WontFix**: The team decides to not take action on this task.
        - **Done**: The task, which doesn't imply coding, has been completed successfully but must be verified.
- **Owner**: The member of the team responsible for the task.
- **Cc**: The members of the team co-responsible for the task.
- **Labels**: This field allows to give extra information about the task:
    - **Type**: The kind of task (E.g.: Type-Defect, Type-Enhancement, Type-Task, Type-Review, Type-Other, Type-Codification, Type-Research)
    - **Priority**: There can be several priority levels to determine if a task is urgent (E.g.: Priority-Low, Priority-Medium, Priority-High, Priority-Critical)
    - **OpSys**: which OS users are affected by the task (E.g.: OpSys-All, OpSys-Windows, OpSys-Linux, OpSys-OSX).
    - **Milestone**: The task should be completed to reach the version specified in this option. (E.g.: Milestone-release1.0).
    - **Component**: The area of the project where the task is located (E.g.: Component-UI, Component-Logic, Component-Persistence, Component-Scripts, Component-Docs).
    - **Security:** Indicates that the task implies security risks for the users.
    - **Performance:** The task is a performance issue.
    - **Usability:** The task is a usability issue.
    - **Maintainability:** The task is a maintainability
- **Block on:** Indicates the dependency of the task on other tasks, that's to say if we change one of them, the dependent tasks must also be modified.

The great advantage of this tool is that we can combine as many labels as we want, and hence we can give too much metainformation about the task that can help the members of the team to reach the objectives, above all easing the management of them.

Moreover this, users are allowed to add, delete or modify any of the features that can be used in the labels, in manner that it’s possible to customize the labels as desired.

Finally, the tool also allows attaching files to the ticket that are important to solve the task.

Besides, a tracking can be done together with these units, and the responsible can check if there is anything to be done to reach a specific milestone or monitor the work of any member.

A good example of the use of this powerful tool is a tester who assigns tickets to the appropriate team member as he or she finds mistakes in different parts of the project.

4.6.3 Subversion

Subversion (SVN) is a version control system initiated in 2000 by CollabNet Inc. It is used to maintain current and historical versions of files such as source code and documentation.

The best option to have all the files of a project available and updated from any location is by setting up a system like this one. It consists of a central repository with all the data which has the ability to recover every change ever made. It indicates what the status of the project was at a certain date. It also makes all the users aware of the commits that other members have done in real time. Each of them has to be tagged in natural language in order to know the tasks that were done in that specific revision. An additional advantage is that users do not necessarily have to be working at the same location.

It is also useful for experimental practices through what are called branches. At a certain point of the project, if a risky implementation is to be carried out, it can be done in a separate line of development. While a part of the team is working on something possibly unstable, the others continue as before. When the changes are completed, they can be integrated into the main branch.

Other important tools are included, like the capacity of comparing or merging two or more files of any version. These conveniences can only be used with text files where the source code is located, since binary ones cannot be performed with these kinds of operations.

Furthermore, we have used a user-friendly software with interface, called Tortoise SVN[1], which simplifies the use of the tool and make it very intuitive.

4.6.4 Eclipse

In order to understand and expand this application, which will be explained later, a detailed look of both the code structure and its implementation is needed. So, besides from the obligation of a software engineering background, a tool is needed in order to fully comprehend the whole system. A tool allowing us to navigate throughout the structure, showing every code file, the project units and everything: an Integrated Development Environent. In our case, the Eclipse IDE, one of the most famous and used IDEs for Java, has been chosen.

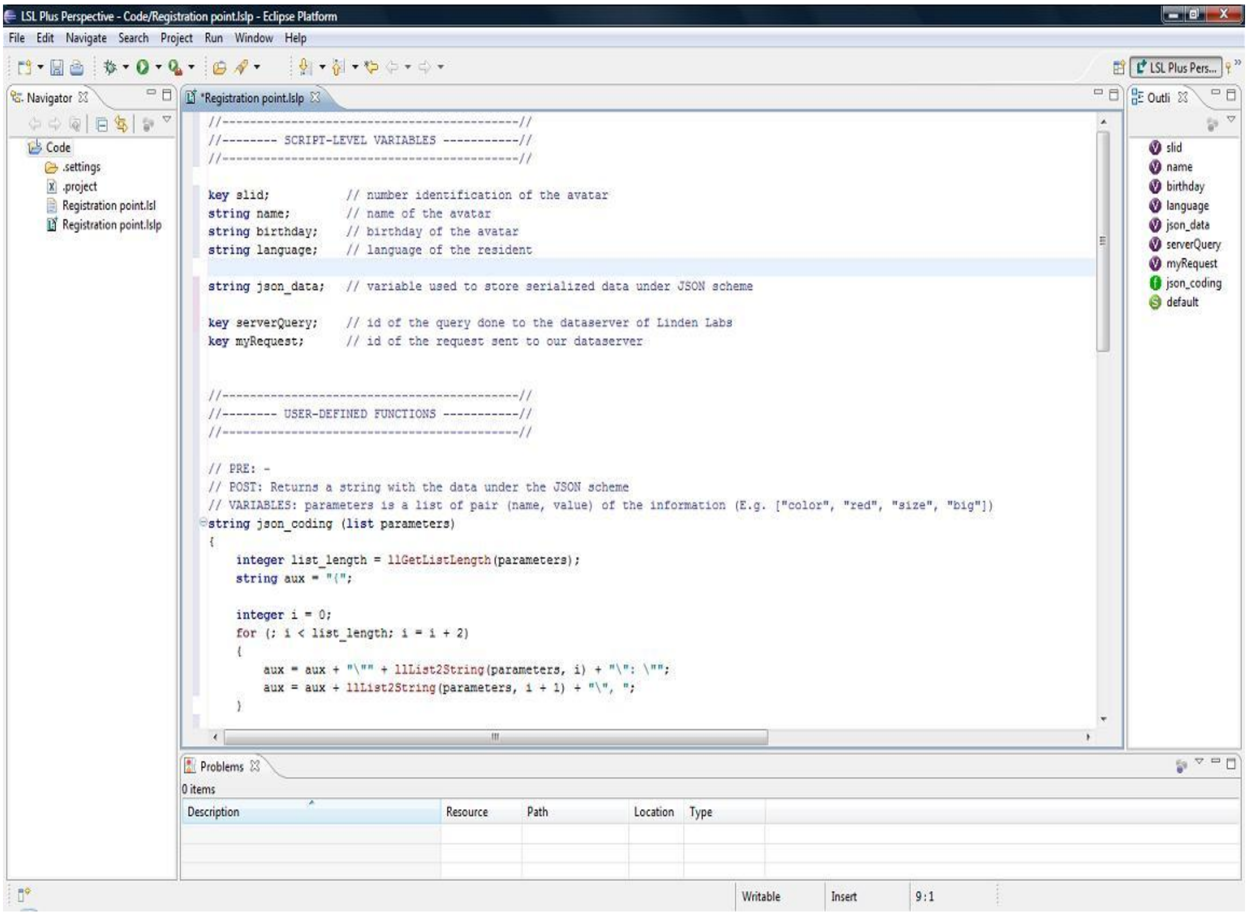
Eclipse is a multi-language software development environment comprising an integrated development environment (IDE). In addition, it's has an extensible plug-in system, which is really important for our project as the client side has been developed in the language Linden Scripting Language, which is specific of Second Life, and consequently it doesn't have any own software development environment. However, it exists of plug-in of LSL for Eclipse called LSL Plus[2] which allows to program in it with all the lexical, semantic and syntactic rules of the language. This is very useful, as we don't need to connect every time to Second Life via the Viewer to write any line of code, as we can write and test them with Eclipse.

Moreover, Eclipse has another plug-in, named Subclipse[3], which works as subversion system in order to synchronize the code with the repository created and managed with Tortoise SVN.

[1] http://tortoisesvn.net

[2] http://lslplus.sourceforge.net

[3] http://subclipse.tigris.org/



```
//-----------------------------------------//
//-------- SCRIPT-LEVEL VARIABLES -----------//
//-----------------------------------------//

key slid;           // number identification of the avatar
string name;        // name of the avatar
string birthday;    // birthday of the avatar
string language;    // language of the resident

string json_data;   // variable used to store serialized data under JSON scheme

key serverQuery;    // id of the query done to the dataserver of Linden Labs
key myRequest;      // id of the request sent to our dataserver


//-----------------------------------------//
//-------- USER-DEFINED FUNCTIONS -----------//
//-----------------------------------------//

// PRE: -
// POST: Returns a string with the data under the JSON scheme
// VARIABLES: parameters is a list of pair (name, value) of the information (E.g. ["color", "red", "size", "big"])
string json_coding (list parameters)
{
    integer list_length = llGetListLength(parameters);
    string aux = "{";

    integer i = 0;
    for (; i < list_length; i = i + 2)
    {
        aux = aux + "\"" + llList2String(parameters, i) + "\": \"";
        aux = aux + llList2String(parameters, i + 1) + "\", ";
    }
```

Illustration 59. Example of use of Eclipse with LSL Plus plug-in

Chapter 5

Conclusion and future lines

As a conclusion, we can say that we have managed to set the base of an e-learning system using as platform Second Life. Our system offers provides the necessary functionalities to carry out simple e-learning activities in Second Life, storing all the data related to the activities and the users that can needed in a future. Besides this, we have also created all the tools needed to manage the system in a very intuitive way.

Moreover this, as we have said, our system has been designed in order to be easily expandable, as we are convinced that there are lots of other features that can complete the system: add learning materials to the web user application linking them to its related activity, allow the users to consult multimedia materials stored in the server through Second Life, carry out practical exercises in Second Life supported and led by the server…

Bibliography / References

1. **Linden, Nelson.** Second Life Blogs. [Online] 01 26, 2011. [Cited: 02 10, 2011.] http://blogs.secondlife.com/community/features/blog/2011/01/26/the-second-life-economy-in-q4-2010.

2. **Urbanist, Web.** Web Urbanist - Sensational, Educational & Inspirational. [Online] 06 18, 2007. http://www.weburbanist.com/2007/06/17/top-3-look-alike-avatars-and-people-from-second-life-to-real-life/.

3. **Neustaedter, Carman and Fedorovskaya, Elena A.** *Avatar Appearances and Representation of Self: Learning from Second Life®.* Arlington : Association for the Advancement of Artificial Intelligence Fall Symposium, 2009. p. 39. 978-1-57735-435-2.

4. **Linden Research, Inc.** Second Life Wiki. *How far does my voice carry?* [Online] 03 20, 2010. http://wiki.secondlife.com/wiki/How_far_does_my_voice_carry%3F.

5. **Pérez, Luís.** *Communication in Second Life.* [PDF] Tampa, South Florida, United States : University of South Florida, 2008. Communication in Second Life.

6. **Curtis, Anthony.** *How To Build in Second Life.* [PDF] Pembrke, North Carolina, United States : University of North Carolina, 01 27, 2011.

7. **Linden Research, Inc.** Second Life Wiki. *Permission.* [Online] 11 02, 2010. http://wiki.secondlife.com/wiki/Permission.

8. **Druart, Gil.** Understanding Rights/Permissions in SL. *A second Life Guide.* [Online] 08 01, 2008. http://knol.google.com/k/understanding-rights-permissions-in-sl#.

9. **Wikipedia.** Economy of Second Life. [Online] 01 30, 2011. http://en.wikipedia.org/wiki/Economy_of_Second_Life.

10. **Linden Research, Inc.** LindenX™ Exchange. [Online] 2011. https://secondlife.com/my/lindex/index.php?.

11. **Randolfe.** Capitalism 2.0. *The Linden dollar Game.* [Online] 25 520, 2007. http://randolfe.typepad.com/randolfe/2007/02/intersecting_tr.html.

12. **Linden Research, Inc.** Terms of Service. *Articles 4.5 and 11.5.* [Online] 12 15, 2010. http://secondlife.com/corporate/tos.php.

13. **Jana, Reena.** Bloomberg BussinessWeek. *Breathing Second Life into Business.* [Online] 10 2006. http://images.businessweek.com/ss/06/10/secondlife/index_01.htm.

14. **SID Lang, Andrew and Bradley, Jean-Claude.** Chemistry Central Journal. *Chemistry in Second Life.* [Online] 10 23, 2009. http://www.journal.chemistrycentral.com/content/3/1/14.

15. **Mark, Roy.** InternetNews.com. *Politics, Virtually.* [Online] 01 12, 2007. http://www.internetnews.com/commentary/article.php/3653691.

16. **Walsh, K.** EmergingEdTech. *Using Second Life for Educational applications.* [Online] 06 21, 2009. http://www.emergingedtech.com/2009/06/using-second-life-for-educational-applications/.

17. *Bases pedagógicas del e-learning.* **Cabrero, Julio.** s.l. : Universitat Oberta de Catalunya, 04 2006, Revista de Universidad y Sociedad del Conocimiento, Vol. 3, pp. 1-4. www.uoc.edu/rusc. 1698-580X.

18. **Latina, e-Learning América.** Estados Unidos: los alumnos secundarios prefieren estudiar online. [Online] 05 02, 2005. http://www.elearningamericalatina.com/edicion/mayo2_2005/it_2.php.

19. **Saleem, Usman.** Developer.com: A Pattern/Framework for Client/Server Programming in Java. [Online] 06 10, 2002. http://www.developer.com/java/ent/article.php/1356891/A-PatternFramework-for-ClientServer-Programming-in-Java.htm.

20. **JSON.org.** Introducing JSON. [Online] http://www.json.org/.

21. **W3C.** HTTP: Status Code Definitions. *RFC 2616.* [Online] 09 01, 2004. http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

22. **Microsystems, Sun.** The J2EE Tutorial: Packaging Web Components. [Online] http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/WebComponents3.html.

23. **Oracle.** The Java Tutorials: Packaging Programs in JAR Files. [Online] http://download.oracle.com/javase/tutorial/deployment/jar/.

24. **Linode.com.** Deploy and Manage Linux Virtual Servers in the Linode Cloud. [Online] http://www.linode.com/.

25. **Blog, Linux Tutorial.** SSH and SCP: Howto, tips & tricks. [Online] 01 2007. http://www.linuxtutorialblog.com/post/ssh-and-scp-howto-tips-tricks.

26. **Hirschfeld, Robert.** Three Tier Distribution Architecture. [Online] 07 30, 1996. http://c2.com/cgi/wiki?ThreeTierDistributionArchitecture.

27. **Rod Johnson, Juergen Hoeller, Alef Arendsen, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Mark Fisher.** The Spring Framework - Reference Documentation. [Online] http://static.springsource.org/spring/docs/2.5.x/reference/index.html.

28. **Foundation, The Apache Software.** Apache Tomcat. [Online] http://tomcat.apache.org/.

29. **Gavin King, Christian Bauer, Max Rydahl Andersen, Emmanuel Bernard, Steve Ebersole.** HIBERNATE - Relational Persistence for Idiomatic Java. [En línea] 15 de 09 de 2010. http://docs.jboss.org/hibernate/core/3.5/reference/en-US/html/.

30. **Foundation, The Apache Software.** The Apache Velocity Project. [En línea] 06 de 03 de 2007. http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html.

31. **Linden Research, Inc.** Second Life Wiki. *LSL Portal.* [Online] 02 08, 2011. http://wiki.secondlife.com/wiki/LSL_Portal.

Manual Reference

Specifications of the library functions in LSL used (31):

1. llRequestAgentData
   http://wiki.secondlife.com/wiki/LlRequestAgentData
   8-11-2010

2. llHTTPRequest
   http://wiki.secondlife.com/wiki/LlHTTPRequest
   27-05-2010

3. llMessageLinked
   http://wiki.secondlife.com/wiki/LlMessageLinked
   25-12-2010

4. llSetPrimMediaParams
   http://wiki.secondlife.com/wiki/LlSetPrimMediaParams
   23-10-2010

5. llSensorRepeat
   http://wiki.secondlife.com/wiki/LlSensorRepeat
   19-10-2010

6. llListen
   http://wiki.secondlife.com/wiki/LlListen
   11-01-2011

7. llSay
   http://wiki.secondlife.com/wiki/Llsay
   25-12-2008

8. llSensor
   http://wiki.secondlife.com/wiki/LlSensor
   25-12-2010

9. llModPow
   http://wiki.secondlife.com/wiki/LlModPow
   08-12-2009

10. llBase64ToInteger
    http://wiki.secondlife.com/wiki/LlBase64ToInteger
    04-06-2009

11. llStringToBase64
    http://wiki.secondlife.com/wiki/LlStringToBase64
    08-07-2008

12. llIntegerToBase64
    http://wiki.secondlife.com/wiki/LlIntegerToBase64
    20-05-2008

13. llBase64ToString
    http://wiki.secondlife.com/wiki/LlBase64ToString
    04-08-2009

14. Sensor
    http://wiki.secondlife.com/wiki/Sensor
    07-01-2011

15. No Sensor
    http://wiki.secondlife.com/wiki/No_sensor
    16-04-2009