NTNU – Trondheim
Norwegian University of
Science and Technology

Location-based games and the use of GIS information

Design of a DSL for (re)locating a pervasive game

Andrea Mannara

Master of Science in Computer Science
Submission date: June 2012
Supervisor: Hallvard Trætteberg, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Norwegian University of Science and Technology

Master Thesis

Location-based games and the use of GIS information
Design of a DSL for (re)locating a pervasive game

Author:
Andrea Mannarà

Supervisor:
Hallvard Trætteberg

June 2012

Abstract

In Pervasive games locations play an important role, so the task of finding proper locations for activities in the games is crucial. It is also relevant to be able to relocate a game to new area/region, to make it available for new players. This project investigated how GIS information can be utilized for (re)locating a pervasive game, and designed a DSL for controlling the location process....

Acknowledgements

I would like to say thank you first of all to my supervisor Hallvard Trætteberg for the patience and the big help that give to me to face this project.

I would like to say thanks also to NTNU and Politecnico di Torino to give me the possibility to have this great experience.

I also have to say thanks to all the people that i met in this wonderful journey that help me to have the ideas for the project.

Finally i want to say thanks to my mother and my father for supporting me in all the aspect of this work and to my sister Martina to have always believed in me.

...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past 5 or 6 years, technology has made many great strides and, day by day, accelerating its steam. Consider, for instance, the evolution of smartphones that within ten years have passed since gadgets for men of business to spread globally.
Hand in hand with the evolution of technology, traveling all the sciences associated with it, from software to marketing. it is normal, then, that in the software, there has been a significant change during all these years.

With a spread, more and more widespread, of mobile devices able to perform any type of application, with performance comparable to that of a standard personal computer, software engineering is increasingly oriented to mobile applications or hybrid. So now we have a lot of software, websites and anything else that have a mobile version.
Obviously, the science of digital games has oriented in this field, constituting the largest market [1] compared to other types of software.

In this great universe of mobile games, dominated by independent games and casual games (a term much discussed and still in definition, as evidenced by the interesting discussion *Casual games discussion* [2]), however, leaves a wide space to study new forms of entertainment, based, mainly, on the possible new interfaces that mobile devices provide.
And it is using these interfaces, in particular those used for the tracking and geolocation, it's possible to create and design games, which are known as *pervasive games*, ie games where the game environment, the scenario, is the the real world. The game, as will be explained below, interacts with the real real world, and makes the player, the *avatar* of himself.

Being a fairly new area of developing, aside from standard tools of programming, yet there is no something specific to the development of this type of games. Unlike other

types of games, which, however, usually are provided with special IDEs, with engine already pre-packaged and anything else, the pervasive game provide a knowledge of different application fields, so it is difficult to develop such products.

Precisely for this reason it is more and more prominent the need to have special tools that allow, in a simpler way, to develop this type of game, or part of them.
As mentioned previously most such games is also educational. They normally include the need for the player to move to various locations and in them pass a test which can be of various nature.

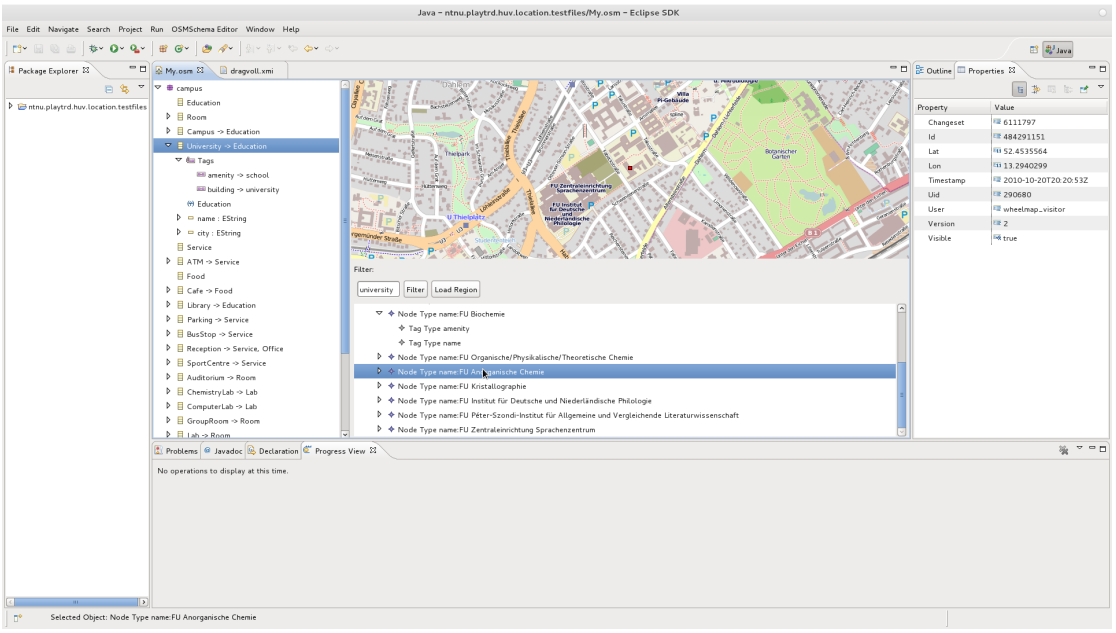


FIGURE 1.1: how can a IDE for location of pervasive game could be

To develop such an application you need a module in which the game defines the locations and the various task, an application that runs on portable support and uses various technologies to interact with the player and the environment .
This can be simplified by building a dedicated IDE and an application skeleton that can then be enriched and personalized.

More specifically, the IDE to the permit, the developer, at least the following operations:

- To define places and region of the game is involved
- To define the characteristics of the player, the basics rules and other features of the game.
- To define the various task and make them easily linkable with the places previously defined
- Be able to reuse previous places and task for previous games

This project, as indicated in the introduction, focuses on developing that part of the IDE to define, redefine and re-use, the places where the game will take place. Key features of the IDE will be the possibility to easily define this information, to be able to choose the places easily and without knowledge of the technologies that will be used later to verify the position during the game itself. Finally, to be able to define the places in a "logical" way, in order to reuse the same structure of the game in different places in the world. The **Fig: 1.1** show how can this such of IDE could result.

1.1 Outline

This document aims to present all the studies and applications made to achieve the purpose stated above.
In particular, the first part focuses on the background studies necessary to carry out the project.

Chapter 1 focuses on the definition of a pervasive game and the reasons why it's needed to build an IDE designed for them.

Chapter 2 explains briefly the tools used: the working environment Eclipse, the EMF framework, and XText, the framework for the development of SDL.

Chapter 3 focuses in more detail on the concept of Location and Relocation, and the difference between the coordinate system and the "logical location".

Chapter 4 deals with exposing the various sources of data used in the project.

The second part of this document, however, exposes what are the steps for creating the environment and the strategies used.

Chapter 5 then describes the process of production and what the initial statements were made before starting to develop.

Chapter 6 focuses on the description of the development process and user interface components of the IDE

Chapter 7 explains in a comprehensive manner the development of the DSL for the production of the location file.

Finally, the last part draws conclusions from the work, the...

Part I

Background Studies

Chapter 2

Pervasive Game

2.1 What is a Pervasive Game

The 1947 is considered the year of creation of the first video game with the "Cathode Ray Tube Amusement Device" [3], followed by *"OXO"* in 1952, the first video game with graphical interface. To get the idea that we have today of video games we have to wait until the '70s with the advent of the first arcades (Galaxy Game, Pong, Gun Fight).
Since those legendary years to now, games has made great strides, and their sudden evolution phenomenon is incomparable to anyone else. Today digital games offering increasingly realistic graphics, gameplays ever more diverse, more interesting stories and interfaces increasingly unusual. Even the production of games has changed a lot. It has gone from amateurs who developed on mainframes during leisure time, and often secretly, in real multinational companies with teams of up to 100 or 200 people for the development of a single games.

Precisely because of to the immense variety of possibilities that the current world of games (and information technology in general that you can talk about something like Pervasive Game, a category of games very young and under definition and development.

According with the definition of Markus Montola:
Pervasive games are a curious form of culture. They exist in the intersection of phenomena such as city culture, mobile technology, network communication, reality fiction, and performing arts, combining bits and pieces from various contexts to produce new play experiences.[4]

In other word a pervasive game is a game that use technology as a support but locate all the game experience in the real world. In the same book Montola describe different type of pervasive games. This project focus his attention of a type of game called location-based game or geogames, like in the article of Schlieder, Kiefer and Matyas: *"Geogames are intended to bring game playing back to the physical world."* [5]. The goals of this type of games are to move into the real world and interactive with that. Other type of pervasive games can use something on the real world to increase the virtual experience (like using the body of the player as interface of the game). [1]

A digital game set in the real world may have difficulty not just irrelevant. It requires, in fact, technologies and tools to verify the position of the players, the places where the game is played, of potential objects, as well as the creation of the position for non-player characters (NPC). The issue of locate, within of a pervasive game becomes fundamental and not to underestimate. In a normal game, being the virtual world, you can jump from one place to another, connect unrelated places, using graphical techniques to represent places, and know exactly the position of the avatar of the player almost automatically. In a pervasive game this can not be done. It must take into account the spaces and places, the problem of connection of these places and the possibility to reach of the same, and moreover to effectively verify the position of the players, they are not in degrees of falsifying it.
We can say that the problem of location and position tracking is the main problem, but as well the main features of a pervasive game.

2.2 Why a Pervasive game

The reasons why it feels the need to create a pervasive game can be quite varied. A pervasive game, in fact, lends itself to a variety of applications that can go far beyond the simple entertainment. For example, a pervasive game set in a city can be very useful for all those who are about to visit.

A good example of applying the concept of pervasive game associated with the visit of the city is the work developed at the university NTNU in Trondheim. The game is developed on the Android platform and consists of a series of tasks of different nature in order to learn in an interactive way the city of Trondheim and at the same time fun to deal with other players. The project was presented at the IGIC (International IEEE Games Innovation Conference) in 2011 and in the paper [7] it possible to find also an extensive study on various aspects of the game and on the proposed questionnaires to the subjects after gaming experience. The results based on the System Usability

[1] Partly recovered by the Project report [6]

Scale (SUS) can say that the game has an enjoyable than the average for other software evaluated in this manner. This demonstrates the fact that the use of pervasive games in these fields is effective and to be taken into consideration.

With the game, as demonstrated , you can visit and learn about the city in an easy and fun way, combining, therefore, the fun part the learning and the exploration. The same thing can be done within a museum, or, as orientation program for new students of a school or a university.

Training and physical activity may be another interesting field in which a pervasive game can fit properly. Imagine you create a game that simulates emergency situations for police or firemen, or games of exploration and treasure hunt for the training runners and athletes.

Finally, you can use the game as a pervasive medium for teaching in schools and universities. More and more, in fact, the science education approaches the concept of the game, since, various studies have demonstrated the efficacy of using playful instruments for the purposes of learning. A good example of this concept is proposed in the work done at the University of Cordoba in Spain, where it was tried to use a pervasive game set in the city to engage students to answer to different tests [8]. The project attempts to combine the technology Moodle (Modular Object-Oriented Dynamic Learning Environment) [9], with location technologies and RFID Identificator. The document is also presented as the use of the game significantly increases student performance in examinations of the courses under study.

As shown in the examples above the pervasive games are ideal for teaching purposes, therefore, are often associated with the concept of "serious games", a term well-defined in the book "Developing Serious Games" by B. Bergeron [10]:

A serious game is an interactive computer application, with or without a significant hardware component, that

- *has a challenging goal;*
- *is fun to play and/or engaging;*
- *incorporates some concept of scoring;*
- *imparts to the user a skill, knowledge, or attitude that can be applied in the real world.*

In particular, what distinguishes the serious game from a normal game is just the latest feature. The serious game lets you learn or improve skills of the player.

As said the pervasive game, by their nature, are perfectly suited to be of educational games, but not only. Indeed, there are many examples of pervasive games that are outside the educational component and fit perfectly into the category of entertainment products.
A particular example of this type of games is "Parallel Kingdom", one of the first MMORPG for smartphones. As you can read in the article by B. Patterson[11]:

Parallel Kingdom places the virtual world on top of the real world using the GPS inside your phone. Attack, dance, hug or team up with anyone around you. Set up trade routes, craft items or even create your own kingdom.
Wherever you are, you can open your phone and play, whether you're on the bus, walking down the street, or at home. Anywhere in the world, any time of the day. Parallel Kingdom never stops, you play it when you want and where you are.

As demonstrated, therefore, the applications of these types of games are numerous without counting the simple aspect of entertainment.
Finally, if we consider the increasingly widespread diffusion of mobile devices and the need for users to interconnect at any place, is almost natural to think that a large branch of the evolution of video games is directed to devices no longer static, but scattered over the world, and it would be unwise to underestimate this condition and do not use in the development of new softwares.

2.3 Location in a Pervasive Game

As mentioned in the preceding paragraphs, for a a pervasive game is essential the concept of localization. A a pervasive game is, by definition, set in the real world, for which, requires a system for defining location in which it is set. There are a pervasive game "location free" that can be played everywhere, but these are often affected by the lack of interaction with the environment, a feature that, most of all, makes interesting a pervasive game. Except for these cases, therefore, the problem of defining the various places where the game takes place is crucial, especially for those types of games (often educational) that aim to explore and discover the place where the game set.
Define the places is not simple. This location can be based on explicit definition, for every location, geographic coordinates of that in the code. This simplistic approach brings a number of consequences should not be underestimated. First of all, write the

data directly in the code is a programming technique that is not recommended in any course book, in any course at any university and any working environment.
Secondly, use directly the coordinates of a location to define it brings a significant amount of work in order to find those coordinates. The programmer will have to worry about first choose the exact locations where he wants to set his work, and after using a system to retrieve the coordinates for each place and then write them in his code (or in an external file). This, beyond to include human error, leads waste of time and energy.

Another problem is the freshness of data. During the programming process and also during the actual usage of the game, the environment, initially studied by the developer, can change. Imagine, for example, to design an interactive game that allows players to live the experience more interesting and rich of visiting a museum. Basing everything on the coordinates of the various works (imagine that the museum is large enough to use the geographic coordinate system to distinguish various locations), does not take into account that, in the museum, the works can be moved, loaned to other museums, added. In this case, the developer must rewrite (and recompile) software to the new condition. This problem, for highly variable environments, it is far from trivial. Try to think, finally, the possibility that the museum will change the whole location, the programmer must use the same time that he first used to retrieve all references to the various posts and rewrite them.
Finally, a problem less important than the other but to be considered, imagine that the software should be used in various places (taking the example above, several museums around the world). In this case, the developer will inevitably have to rewrite the software for each location where it should work.

From this analysis it can be understand that should strongly avoid this strategy. But, even if you put references to the coordinates in different files, and referring to them in a dynamic way, some of the problems described above, this would still leave unresolved.

In order, therefore, to avoid such problems, it's possible to think of referring to locations, not in terms of physical-mathematical (coordinate system), but in terms of "logical". Nowadays, especially if we talk about larger environments than a building (a neighborhood, a city, a region), there are a multitude of geolocation services and presentation of data, up to date and reliable. Often, these systems combine to the presentation of the coordinates of a point, a series of logical information, such as street name, building type, business name, description and more. And, above all, these systems have search system that are based on this information, rather than on the simple system of coordinates. Just think, for example, how the users use certain systems such as Google Maps copyright: there are very few users in the search field write a pair of coordinates. Most

of the time will just write an address, a name of a famous place, or a search on the types of place.

Refer to the sites in a logical way allow avoiding the problems listed above and have a number of advantages. Consider now, for example, want to build a game that consists of an interactive tour of a city. Just give the game the names of the most important that we are the stages of the tour and let the system retrieve the coordinates, and displays the map of the software on your PDA or smartphone. But not only this: having the logical reference, the game may use other services and provide recovery information automatically using other online search services (e.g. web sites of encyclopedias), avoiding the burden to the programmer from having to build previously a database to refer to.

Chapter 3

Introduction about the concept of Location and "logical location"

The term location can mean many things. As defined in the Oxford dictionary [12]:

location: *n,*

1. *particular place or position*
2. *an actual place or natural setting in which a film or broadcast is made, as distinct from a simulation in a studio*
3. *the action of locating someone or something*
4. *a position or address in computer memory.*

Somehow the concept of location, in our case, is something between the first and the second definition. Basically it is a specific place at a precise point of our planet, but at the same time a pervasive game can in some way resemble a movie or a play where the site where it takes place, the location, is the set where the game takes place.
Considering then the second aspect, as it well be known, often, movies are filmed on artificial and fictitious sets which simulate a special place in studios. In some way, therefore, there is a scene that should be filmed in place, but for convenience and often for economic reasons is simulated in another place, or even in a virtual way.
This concept, although the film industry is emphasized, can somehow introduce an essential concept to the purpose of this study, i.e. the possibility of locating the pervasive game in different places, similar in nature or by function. This introduce the concept of "logic" location, but first of all it is good to deepen the concept of "physical" location to better understand the following definitions.

3.1 Coordinate System

[1] In order to locate places and tracking player, have a sufficient knowledge how the coordinate system work is essential. According to wikipedia:

A geographic coordinate system is a coordinate system that enables every location on the Earth to be specified by a set of numbers. The coordinates are often chosen such that one of the numbers represent vertical position, and two or three of the numbers represent horizontal position. A common choice of coordinates is latitude, longitude and elevation. [13]

In most of the maps software and website elevation is not a strictly required value. In order to find a specific point in the map longitude and latitude are sufficient. This two values of a specific point corresponds respectively to the angular distance between the equatorial plane and the line normal to the surface in this specific point (latitude), and the angular distance from a reference meridian (as usual Greenwich meridian) (longitude). In software this to value are represented in decimal degrees. Usually, for what concern GPS technology and maps services web site this number have a precision of 6 decimals, it mean that the minimum distance from two point is c.ca 1 cm. So theoretically it's possible to use this system also in small places. The problem, in that case, come from the technology and the intentional limitation of the GPS technology that is property of the U.S Department of Defense. The limitation impose an precision of at least 7.8 meters. [14] For now coordinate system is the most used way to represent places in the world. With the enter of the internet web in our daily life it will be more useful to start to think different ways to represent that. Coordinate give only one information of a place (the position in the Globe), but today when we looking for a place the position is not the only information that we need. Using other type of positioning of places, for example logical description and tags collection, could be the startpoint for the geo-location system of the future.

3.2 Definition of "logical location"

As mentioned in **par 2.3** there are several reasons why it is not convenient to use directly the coordinates to refer to a place. It's needed, therefore, to use an alternative way to define the concept of place.
The increasingly widespread dissemination of search systems and systems of classification of places of the same, allows to create a system which a place can be defined by

[1] Edited from the Project report [6]

qualitative features, rather than using coordinates. It can be used, for instance, the name of a place, although this is often not unique, coupled with other information like the city belong, the region, the state. With enough information you can obtain a qualitative description of a place that is unique throughout the world. Is defined, then, **logical location**:

A set of qualitative information that renders the definition of a unique place in a given context.

The concept of context is important. In fact, defining first the context, all the information necessary to make a unique place can be significantly reduced. Imagine that you define, for instance, as a context, a university campus. Define a site with the simple term "secretary" could be enough to make this place unique, which, however, in a wider context may be misconceived.
By defining the place with enough information and associating them to the context you can create a system that uses ,in order to retrieve the exact location, geolocation services or online maps. In this way, when the exact position will be necessary (for example at the time to verify the position of the player with respect to the target) can be easily access to the geographic coordinates.
The amount of information needed depends on a number of factors, including the main one is the service used to retrieve the coordinates.

3.2.1 From logical location to Coordinate: Sources

At a time when the video game will be started on a portable device is necessary that the places, defined in a logical way during the design and programming of the game, are somehow converted to geographic coordinates. This process can be delegated to the portable software, but this can cause an additional burden that, in the case of software on mobile devices, should not be underestimated. Bearing in mind that after the design and programming of the game the places are not changed, is possible do this conversion directly after the selection of the logical places. It will, therefore, the work environment that will create a document in which every place has defined the appropriate coordinates. In **chapter 6** will explain how this document is produced.
The various sources often contain associated with each element of the database, a whole series of information, often in the form of tags, which allow, for each geographical coordinate, to have a logical, often in a natural language, description of the type of places present in it. If the description logic used in the software is designed to be consistent with these information, is possible to build a system that combine this information to

the data sources so that it returns exactly one result. This is very difficult to obtain due to the nature of the information.

There are many types of data sources from which it is possible to draw this type of information. First of all the online mapping sites often provide specific APIs to help programmers to use the data within them. These should be considered the main sources where take information since they usually attach to the simply coordinate information to help the searching system and to improve that.
Even sites like, yellow pages, or the like can be useful because, often, alongside a range of useful information easy to address, and often these are very accurate, also because most of the time the owner of places put information.

Combining all these resources, you can create a versatile tool that can gradually build a geolocation system purely logical and not associated with geographic coordinates.

Chapter 4

Eclipse and EMF

In order to design a complete IDE, flexible, multi-platform, extensible and integrated with other frameworks, the choice of possible of platform to use is fairly limited.
The combination of Eclipse software and EMF framework turns out to be the best solution because it has all the features listed above. Using this combination is possible to produce a highly professional software, which can be easily distributed and integrated with future implementations. Furthermore, the community nature of the Eclipse project allows the developer to be able to use a wide range of solutions and tools that are very powerful and effective.

For these reasons it was decided to use such a combination with the addition of XText, a relatively new tool but very effective to produce DSL and tools to use it.

4.1 Eclipse

It is not very easy to define what exactly is Eclipse. Eclipse is many things: it is primarily a software, but also a community, is an ongoing project, but it is also a foundation. Quoting directly from the Eclipse web site [15]:

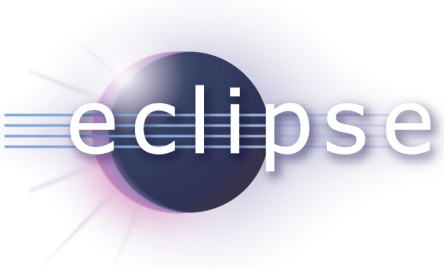


FIGURE 4.1: Eclipse foundation Logo

Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and run times for building, deploying and managing software across the life cycle. The Eclipse Foundation is

a not-for-profit, member supported corporation that hosts the Eclipse projects and helps cultivate both an open source community and an ecosystem of complementary products and services.

With regard to this project that it's going to use is the "Eclipse Platform", which is a complete development environment, modular, editable.
The reasons why it's decided to use Eclipse for development of this project are different and can be listed below:

- As it is said, Eclipse is a full and expandable, written in Java, and primarily designed to develop in Java. Develop the project in Java port a whole series of advantages which the use of libraries and constructs already available, but also the peculiarity of obtaining a software multi-platform, i.e. the possibility to execute in all the main operating systems.
- On Eclipse you can use other frameworks that enable and facilitate programming with different techniques, such as model-driven approach or DSL development, which is what will use in this project.
- Since the aim of this project is to develop an IDE, use Eclipse allows direct programming plugins for Eclipse itself and create a specific development environment fully customized on the basis of Eclipse. In other words, you can create a version of Eclipse that implements the desired IDE.
- The Community nature of the project allows Eclipse to expose their work to the community that can enrich it, use it and review it. This can lead to very different future scenarios that increase the potential of the software. Similarly you can use the material in the same community to enrich their own software.

One of the features of the Eclipse platform, making it the ideal instrument for the purposes of this project, is its plugins structure. The paper *"Eclipse Overview"* [16] says:

A plug-in is the smallest unit of Eclipse Platform function that can be developed and delivered separately. Usually a small tool is written as a single plug-in, whereas a complex tool has its functionality split across several plug-ins. [...]
Plug-ins are coded in Java. A typical plug-in consists of Java code in a JAR library, some read-only files, and other resources such as images, web templates, message catalogs, native code libraries, etc. Some plug-ins do not contain code at all.

Since the objective of this project is to develop an IDE for managing locations of a pervasive game, the user interface is critical. The Eclipse user interface (UI) framework makes arrangements which puts all the APIs necessary to build interfaces for plugins. In particular, the framework is composed by using two general-purpose libraries SWT and JFace.

The Standard Widget Toolkit (SWT) is a graphics library written in Java that provides a basic set of graphical widgets with which you can build interfaces. It is used in most Java projects and is the main tool for creating graphical user interfaces in that language. Its peculiarity is to be very performed compared to other libraries (eg swing), most complete and richest of APIs comparing with other libraries (eg AWT) and being a wrapper library of graphical operating system (Win32 and WPF on Windows, GTK on Unix-like OS, Cocoa on OS X) works seamlessly with it, while remaining OS-independent.

JFace is a high level library of written on the SWT, and provides a variety of tools based on the concept of the Viewer. There is basically a subclass of Viewer for nearly all the widgets in SWT, allowing an approach t high-level data displayed in those widgets, proposing methods for updating and rebuilding themselves.

4.2 Model Driven and EMF

In order to understand what EMF (Eclipse Modeling Framework) is, it's necessary first define two specific concept: **what is a model** e **model driven programming**.

4.2.1 Model definition and UML

According with the Oxford dictionary's definition [17] a model is:

a simplified description, especially a mathematical one, of a system or process, to assist calculations and predictions

In computer terms, we can say that a model is a graphical representation, a diagram, a table that represents a process, data, a concept, and so on. In computer science there are a multitude of different ways to represent a model. Since the early nineties, mainly as a consequence to the sudden spread of object-oriented programming and the development of large and complex software, various corporations and consortia have tried to create a standard for a unique language that would serve to model the various characteristics of a software. But it would take until 1996 for the first version of UML (Unified Modeling

Language).
In August 2011 this language reaches version 2.4.1 which says [18]:

The objective of UML is to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes. [...]
One of the primary goals of UML is to advance the state of the industry by enabling object visual modeling tool interoperability. However, to enable meaningful exchange of model information between tools, agreement on semantics and notation is required. UML meets the following requirements:

- *A formal definition of a common MOF-based metamodel that specifies the abstract syntax of the UML. The abstract syntax defines the set of UML modeling concepts, their attributes and their relationships, as well as the rules for combining these concepts to construct partial or complete UML models.*
- *A detailed explanation of the semantics of each UML modeling concept. The semantics define, in a technology-independent manner, how the UML concepts are to be realized by computers.*
- *A specification of the human-readable notation elements for representing the individual UML modeling concepts as well as rules for combining them into a variety of different diagram types corresponding to different aspects of modeled systems.*
- *A detailed definition of ways in which UML tools can be made compliant with this specification. This is supported (in a separate specification) with an XML-based specification of corresponding model interchange formats (XMI) that must be realized by compliant tools.*

Today the UML language include different type of models that represent different subjects. The main diagram are:

- Use Case Diagram
- Class Diagram
- State Diagram

The UML is designed to represent any type of system, not just information technology. Today, UML diagrams are used to represent any type of process or system such as economic, productive, industrial.

In the context of computing the use of a modeling approach has led to an improved technique for programming and managing the implementation process. He also permitted to establish a real programming technique called **Model-driven**.

4.2.2 Model Driven Programming

As described in *MDA Guide* by J. Miller and J. Mukerji [19] the Model Driven Programming is: *The Model-Driven Architecture starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform.*
MDA provides an approach for, and enables tools to be provided for:

- *specifying a system independently of the platform that supports it,*
- *specifying platforms,*
- *choosing a particular platform for the system, and*
- *transforming the system specification into one for a particular platform.*

The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns.

Basically, the MDA technique consists of representing all the software in the form of UML models, and to make sure that these models directly reflect the software. In general tried to create languages that are directly represented as models or even that the language used to describe the model itself is the same that is compiled and / or executed. The technique is premised on the absolute necessity to model the production process of the software and, therefore, has as an objective speed up this process, avoiding the manual conversion from model to code but by placing it as an automated, semiautomatic or direct process.

One way to obtain a model that is partially or totally executable, is to use the union between XML, XMI, and Java, which is precisely the technique that EMF uses.

4.2.3 XML, XMI, OCL

[1] *Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process*

[1] Partly recovered by the Project report [6]

them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents [20]

XML is the most used markup languages in information technology. With XML is possible to represent data with tags and attribute. It's used for exchange data or represent that. Currently most of the markup languages on the web is a subset of XML (HTML is an XML based markup languages). In the **Code 4.1** there is an example of XML document.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <tag attribute="attribute value">
    <subtag>Sub tag value<</subtag>
  </tag>
```

Listing 4.1: Example of XML document

When data are represented with XML files often is present also an XSD (called also XMLSchema) file. XML Schema Description is a xml file that describe which elements are allowed for the type of data into the XML file. It's also described the structure of the various elements inside the XML data. For better understand what the difference between an XML file and a XSD file it's possible to draw an analogy with OOP (Object-Oriented Programming) paradigm. OOP is based on two elements: classes and instances. A class is the declaration of an object structure and its behavior. An instances of a class is an occurrence of the object describe by the class. With this perspective an XSD is the collection of the declaration of classes and an XML is the collections of instances. **fig. 4.2** show a better description of the analogy.

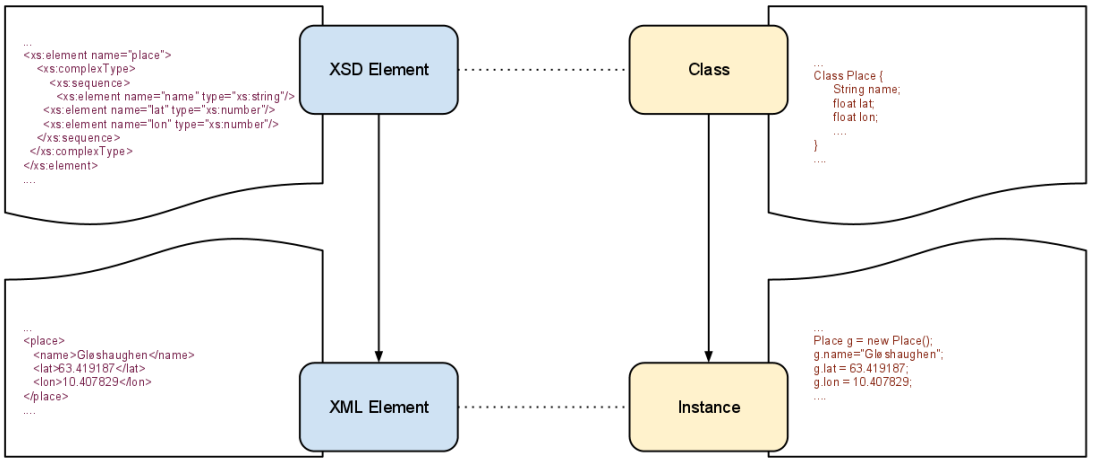


FIGURE 4.2: Analogy with XSD-XML and OOP class-instance

This analogy will be useful during the description of XSD-XML approach into the framework.

Mapping Data on Model Driven Software

4.2.4 What is EMF

The Eclipse Modeling Framework enables developers to rapidly construct robust applications based on surprisingly simple models.[21]

EMF is a framework for the Eclipse editor that have the objective to create an IDE for the model driven programming technique. The goal of the framework, which is written in Java, is to give the possibility to design models and it automatically generate the code that implements the model. EMF uses a subset of UML to define patterns that are mostly class model. EMF has the ultimate objective of making it virtually the same code of java classes, the UML model and the definition XMLSchema for data serialization. Nothing change if, using EMF, first define the Java classes and then it's got from it the model or, conversely, if first designs a model and then want the definition of Java classes for the same.
This allows a model driven programming that tends to optimize the working process and to accelerate the production times. EMF combines, as stated, Java, UML and XML in a single equivalent product.

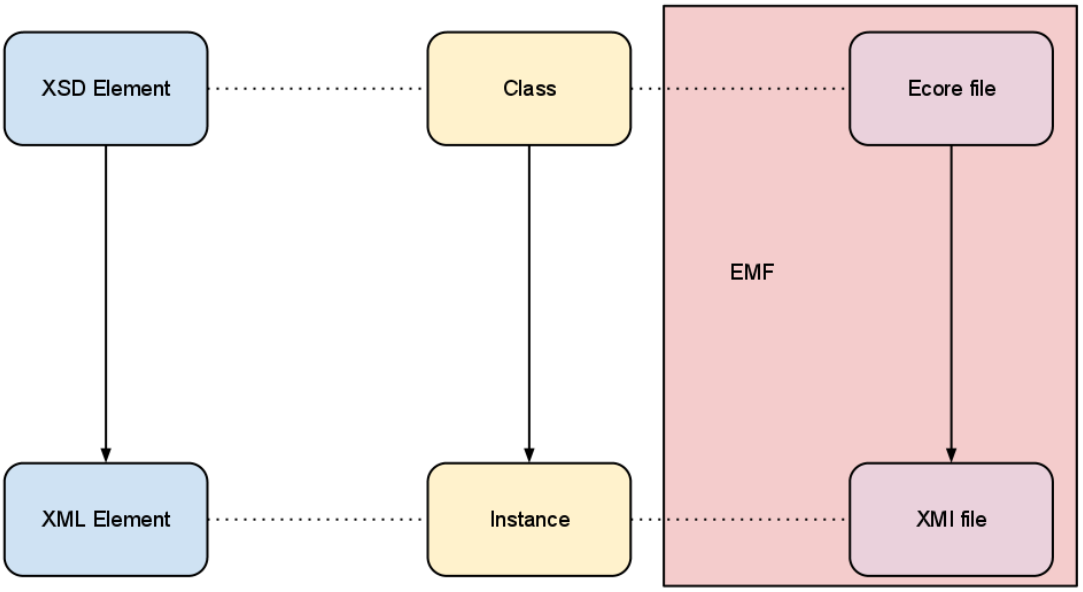


FIGURE 4.3: Relation between XML, Java and Ecore Model

EMF provides a type of model called Ecore, which is itself a model and is considered the metamodel to build other models in the framework (in addition to be the metamodel of itself). Ecore model provides a number of elements to represent any model. The **Fig. 4.4** from the book *Eclipse Modeling Framework second edition* [21] illustrates the elements of this metamodel. As the **Fig. 4.3** [1] shows there is a direct relation between xml, Java code and Ecore model.

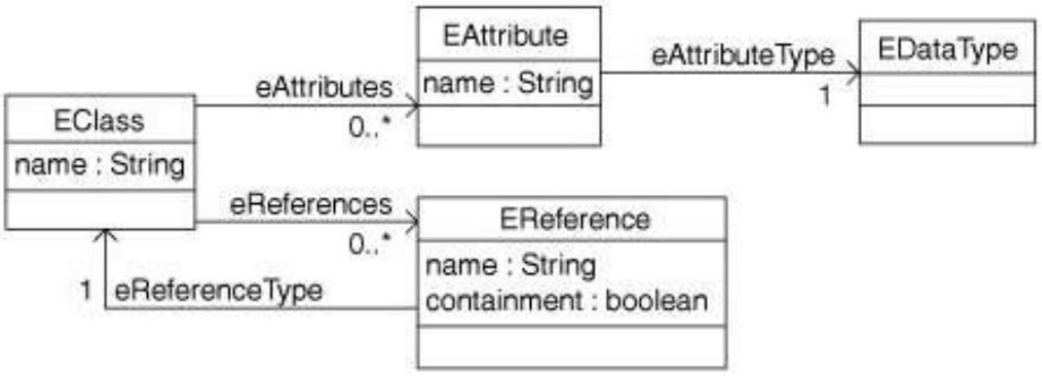


FIGURE 4.4: A simplified subset of the Ecore Metamodel

Therefore, using these elements, it is possible to draw any kind of model. In particular, class models, where each element is an element EClass instance, an instance of each attribute and each relationship an instance of EAttribute eReference. In addition, in the moment of generation of the code for the model, each element will be implemented as a subclass of the class Ecore and so on.

Another peculiarity of EMF is the ability to generate not only the execution code of the model, but if, for example, the model would represent the structure of a database, it gives the possibility to generate an editor for the instances of the model and therefore, to create a new file type that serializes the instances of the model. The generated editor (which it's possible to edit) will be similar to the editor used to manipulate Ecore models and can be installed as a plugin or distributed as a minimal version of eclipse. The **Fig. 4.5** shows how is the ecore editor and also the ecore diagram, an useful tool that allow the developer to see his model like a graph, as like as UML standard.

4.3 XText

[2] In order to understand what xText is and why is useful for this project, is better to explain what DSL is and why is important in this context. DSL is the acronym

[1] Recovered by the Project report [6]
[2] Partly recovered by the Project report [6]

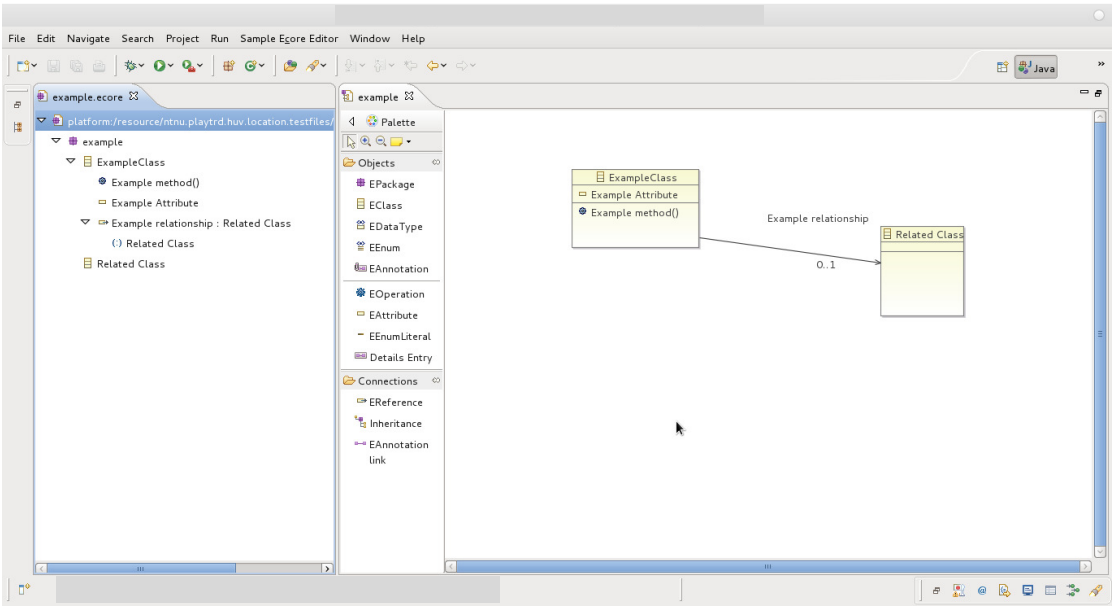


FIGURE 4.5: The ecore editor and Ecore Diagram Editor in Eclipse

of Domain-specific Language and, as the name says, concern all the specific language created for a specific purpose.

Xtext is a framework for Eclipse that permit to define a DSL and create an environment for that. As the Xtext user guide explain perfectly [22]:

Xtext provides you with a set of domain-specific languages and modern APIs to describe the different aspects of your programming language. Based on that information it gives you a full implementation of that language running on the JVM. The compiler components of your language are independent of Eclipse or OSGi and can be used in any Java environment.

Chapter 5

Geographical and Location Source: OpenStreetMap

[1]

> OpenStreetMap (OSM) creates and provides free geographic data such as street maps to anyone who wants them. The project was started because most maps you think of as free actually have legal or technical restrictions on their use, holding back people from using them in creative, productive, or unexpected ways.

OpenStreetMap is a website where everybody can add geographical information. There is a very big community works on this project and the data are almost updated. The advantages of using this instead of other maps services is that being an open-source project the data are directly accessible for everybody. So openstreetmap could be used as other maps services website or it's possible to help the project and adding data. As shown in **Fig: 5.1** Open Street Map is presented in a manner similar to many other websites that offer mapping services, in addiction of the typical wiki layout. OSM in fact, was born to be a Map wiki community. On the right you can find the text box to search and links to project information and guides to take part actively or to use the service. Above we have the core services that OSM have:

view is the main environment of OSM and the first that appears. In it you can browse the map, using different rendering engines and display the search query.

[1] Extended from the Project report [6]

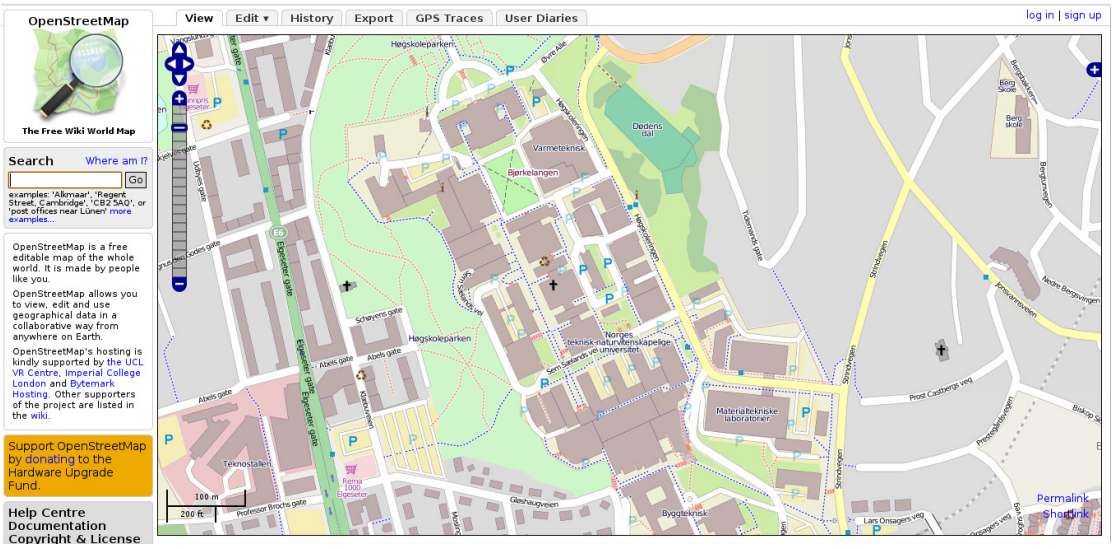


FIGURE 5.1: Main page of Open Street Map

edit is the editing environment of OSM. Once logged in, you can enter your data and update the map with their information or correct the information of others. There are several tools to edit the OSM maps.

history displays the history of changes in the area you are viewing.

export is very similar view environment and allows you to select an area to export data in different formats.

gps-traces collects the tracks that the various users have loaded so as to be possible to used to draw the maps.

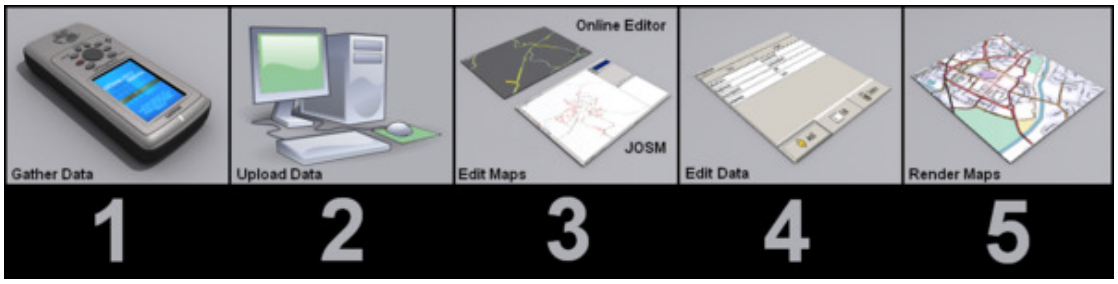
user diaries collects various travel diaries of the community.

In addition to this OSM provides a complete and comprehensive wiki on how to best use all the tools available, and, finally, provides the API to use OSM in different environments.

5.2 Open Street Map Database

The most important part of this work is the database. In the beginning it was based on MySQL but after 2009 it switched on PostgreSQL. In the Db there are two tables for each kind of data. One is the Master Table and the other one is the Current table. The current table is a subset of the master table and contain the current data. The master table contain also the history of each data. The Data primitives are the generic types of data in OSM and for each type there are the two tables.

5.1 Insert data process on Open Street Map



Like the **Fig: 5.2** shows in order to put data on OSM it's needed to follow some the steps for adding geographical information to the system. To do that the "uploader" need a special software on this gps device to retrieve gps information (there are several) and after write with hands, or using software tool to create the XML-like files with the description of the region. If the document is well-formatted the OSM engine read the data and render the region (there are also different render engines). After this the data will be available for everybody.
Have a community support is useful because the amount of data is more than the other close maps services and you can find information very update, also because the open-source nature allow people to using this for other project and in these cases they add the other information that they need. This is the case, for example, of Gløshaughen Campus. There are lots of information about this campus because for other project, that focused of creating an interactive map of the campus, they adding all the information that they need.

FIGURE 5.2: Steps to add a map on OSM

NODES are the basic elements of OSM and define almost everything in OSM. The field tags refer to an array of tags (in another table) that define the nature of the node.

TAGS are simply couples of key-value associated to each node.

WAYS consist in a group of nodes and it's used to describe paths, ways etc... They also have an array of tags but also an array of nodes. If the started node and the ended node are the same it's a CLOSE WAY or an AREA. Area are often used for design building shapes.

RELATIONS connect nodes or ways with specific roles.

These three primitives build all the OSM dataset. This means that in order to distinguish the nature of the various nodes, the only possible way is to use tags. For that the domain of tags results quite extensive and the nature of the tags is different depending on how they are used. Although formally tags are all the same, simple key-value pairs, the different users, real or software, assign a different weight and a different meaning and use them in various ways.
Based on this principle, then, by analyzing the domain of tags is possible to trace

to several subdomains, entirely arbitrary and without clear rules, but maintained by agreements. Mainly tags are used to represent the nature of a node, ie which class node might belong. Obviously, a node can belong to several different classes, as for example a node might have a tag identifying it as a bar but at the same time by another tag, the connotation of room for events.
While the firsts are used to build families of nodes with the same characteristics (for example, the graphical representation on a map, or for the search), other tags are used to distinguish the various instances; in this way we could have tags describing the name of a street or a building, a room, the number of entrances, which buses stop at a specific stop.
Finally, other tags are used by specific applications for various uses. We can imagine, for instance, a company with several locations around the world, wishes to use OSM to create an interactive map of the company for internal purposes and/or external. In this case you can find specific tags used by the company in question only to identify their own buildings. Another example would be someone a bit 'too self-centered who decides to sign with a custom tag all nodes inserted. Since the domain of a folksonomy tags that is possible.

5.3 Open Street Map API

[2] Since OSM is an Open Source project which brings together the different entities and even big partners, tools and the possibilities of using OSM are many. So too are the APIs. Currently there are three major sets of APIs:

XAPI is the first set of APIs created by a user and now provided by mapquest [23]. They are APIs for the read-only data that provide the possibility of search and querying. The other sets of APIs are mostly based on this syntax. They consist of HTTP GET requests where the keys are inserted as parameters of query. Return a package which contains a HTTP osm file in XML.

Overpass API are an extension of XAPI that differ from the latter to have a real query language, **Overpass QL**. Request are always made in the HTTP GET format and are also compatible with the XML format used in XAPI. If using the Overpass QL, query must be sent into the field *data* of the GET request.

JOSM is a Java software for editing OSM. Since it is an OpenSource project, it is possible to use their own libraries of search and querying the database into other

[2] Non present in the Project report [6]

software. In this case, the APIs are called by the methods of a class and re-implement XAPI.

Except for JOSM that returns an instance of a class, in the other two cases the result of a request is an XML file formatted in XML Schema (OSM osm extension). The result will be just a subset of the XML database of Open Street Map, wich is also fully downloadable.

As discussed, searches and queries must be made via HTTP request to a server. There are different depending on which is the API, and you can, if desired, create your own server by installing the necessary API version and downloading the entire database. In general both APIs allow to request a set of nodes via the sending of an area expressed by four cardinal points. Also you can make requests via tags. And finally, it is possible to combine these two requests to get all the nodes in a specific region of a specific type. In any case the result will be an XML file with a series of nodes.

```
//request bounding box with XAPI
  http://api.openstreetmap.org/api/0.6/map?bbox
  =10.39843,63.41414,10.41165,63.4201
//request bounding box with OverpassQL
  data= node(10.39843,63.41414,10.41165,63.4201);
      out body;
  http://overpass.osm.rambler.ru/cgi/convert?data=data=node
  %2810.39843%2C63.41414%2C10.41165%2C63.4201%29%3Bout%20body%3B
```

Listing 5.1: Example of OpenStreetMap request

The **Code: 5.1** shows a request of a bounding box that surround the Gløshaughen campus of NTNU, in the two different type of APIs.

```
<osm version="0.6" generator="Overpass API">
  <meta osm_base="2012-02-22T14\:07\:03Z"/>
  <node id="31264142" lat="63.4263262" lon="10.3958034">
    <tag k="alt_name" v="Trondhjem"/>
    <tag k="name" v="Trondheim"/>
    <tag k="name:de" v="Trontheim"/>
    <tag k="old_name" v="Nidaros"/>
    <tag k="old_name:de" v="Drontheim"/>
    <tag k="place" v="city"/>
    <tag k="source" v="earth-info.nga.mil"/>
    <tag k="wikipedia:en" v="yes"/>
  </node>
  <node id="671401968" lat="50.9730261" lon="11.0279965">
    <tag k="name" v="Trondheim"/>
    <tag k="shop" v="clothes"/>
    <tag k="wheelchair" v="no"/>
```

```
    </node>
</osm>
```

Listing 5.2: Example of OpenStreetMap API answer

The **Code: 5.2** shows a typical answer of a request, in this case a node.

With these tools you can use OSM for all possible information on various places in a specific region, and that is exactly what is needed for the purpose of this work.

5.4 Nominatim

The APIs of OSM allow to obtain a collection of nodes specifying an area (bounds) or, operating on tags. In the case of tags, however, we must specify at least the key, making the very specific query. This system, in fact, lacks of a semantic search, i.e. a search that returns the nodes that have some tags or some parameter that matches or is similar to query search.
To avoid this and to make OSM similar to other online mapping services, OSM provides a project (also open source and easily installed on any server) that is focused on providing a service called **Nominatim**.

Nominatim is a service that has, again like the API, a HTTP interface, which via GET queries returns a set of information that are the result of a semantic query on the database. Nominatim performs a series of queries based on certain tags (for the list you can see the page [24]), tags, which mostly contain information about the names or addresses, and then returns the data in xml or json but in a different format than the data returned by the APIs or saved in the database. The **code 5.3** exposes a simple query to the service and its response in XML.

Nominativ, finally, provide also a service called *reverse geocoding* that aims to result a specific address of provided coordinates.

```
http://nominatim.openstreetmap.org/search?q=Trondheim&format=xml
```

```
<searchresults
  timestamp="Tue, 05 Jun 12 14:15:58 +0100"
  attribution="Data Copyright OpenStreetMap Contributors, Some Rights
    Reserved. CC-BY-SA 2.0."
  querystring="Trondheim" polygon="false"
  exclude_place_ids
    ="128891033,184917,69612272,69623905,51503057,7140200"
```

```
  more_url="http://nominatim.openstreetmap.org/search?format=xml&
    exclude_place_ids
    =128891033,184917,69612272,69623905,51503057,7140200&accept-
    language=en-US,en;q=0.8&q=Trondheim"
>
  <place place_id="128891033"
      osm_type="relation"
      osm_id="406549"
      place_rank="14"
      boundingbox
    ="63.3037910461426,63.5188484191895,10.0044040679932,10.7260761260986"

      lat="63.4091124566799"
      lon="10.3442806402932"
      display_name="Trondheim, Sor-Trondelag, Norway"
      class="boundary"
      type="administrative"
      icon="http://nominatim.openstreetmap.org/images/mapicons/
    poi_boundary_administrative.p.20.png"
  />
  <place place_id="184917"
      osm_type="node"
      osm_id="31264142"
      place_rank="16"
      boundingbox
    ="63.416322937012,63.436326751709,10.385802497864,10.405803451538"
      lat="63.4263262"
      lon="10.3958034"
      display_name="Trondheim, Sor-Trondelag, Norway"
      class="place"
      type="city"
      icon="http://nominatim.openstreetmap.org/images/mapicons/
    poi_place_city.p.20.png"
  />
  <place place_id="69612272"
      osm_type="way"
      osm_id="70724671"
      place_rank="26"
      boundingbox
    ="56.0665512084961,56.0667915344238,-3.42122840881348,-3.4209291934967"

      lat="56.066629370311"
      lon="-3.42103396739874"
      display_name="Trondheim, Halbeath, Fife, Scotland, KY11, United
    Kingdom"
      class="highway"
      type="residential"
  />
```

```
<place place_id="51503057"
    osm_type="way"
    osm_id="46275499"
    place_rank="26"
    boundingbox
 ="51.8508987426758,51.8515129089355,4.50965213775635,4.51104021072388"

    lat="51.8511993825579"
    lon="4.51021507633146"
    display_name="Trondheim, Vaan Park, Barendrecht, Stadsregio
 Rotterdam, South Holland, 2993 LL, The Netherlands"
    class="highway"
    type="unclassified"
/>

...

</searchresults>
```

Listing 5.3: Example of Nominativ query and Answer

5.5 Other Possible Sources

Since Open Street Map is a community, day by day, increase the information and this nature put this source in the position to be perfect for this kind of service.
Nevertheless, you can add other data sources that allow you to enrich and refine the process of conversion from logical information to geographic coordinates.
Data sources that may be suitable for the purpose could be:

- yellow pages sites (such as gulesider.no, the site of the Norwegian yellow pages)
- sites of tourist guides (as full of information about places and often with indexing and very rich classification)
- online encyclopedias (very rich in information and often they associate to the description of the places the coordinates)

Combine different sources can be done by building ad hoc modules to extract data and then design a system to merge this data into a single source. This most likely would require an internal database, with all the problems that result from it (data inconsistency, freshness of data, database size etc...)

Part II

Developer of the IDE for *logical* location of pervasive games

Chapter 6

Production Process

In order to develop better all the objectives of this project, we started from the idea of what should result and then go up in a bottom-up technique to try to understand what is needed and, in the discovery phase of the elements that constitute the end product has been changed, added all that was necessary and eliminated what was superfluous.

First, it must be clear what will be the final result. The objective of this project is to develop an environment and a DSL for the development of pervasive games. In particular, in the context of this specific study, the objective is to develop an environment and a DSL for the declaration of the places affected by the pervasive game. As previously mentioned a Pervasive Game inevitably need to specify the various places where it will run. Finally, a additional request, was to make these statements is completely independent of the coordinate system or the specific location of a place
To summarize, therefore, the end products will have to be:

- Development Environment
- DSL
- Independence from specific geographic locations

Finally, it is good to think about what the process of creating the game, or at least the party directly affected the game in order to imagine the elements and products that will produce the IDE and the DSL. The next chapter aims to understand precisely what process leads to the development of this document and what are the necessary elements to it.

6.1 Process Study

In the realization of a pervasive game the definition of locations is a fundamental part. This definition can be done directly by entering the coordinates in the code when they are needed. This approach, however, includes a series of disadvantages, such as for instance the review of the code in case of changes in the coordinates, errors, modifications of the game, etc..
The next solution could be to create aliases for the coordinates in a single file and refer to the sites using these aliases in the code.

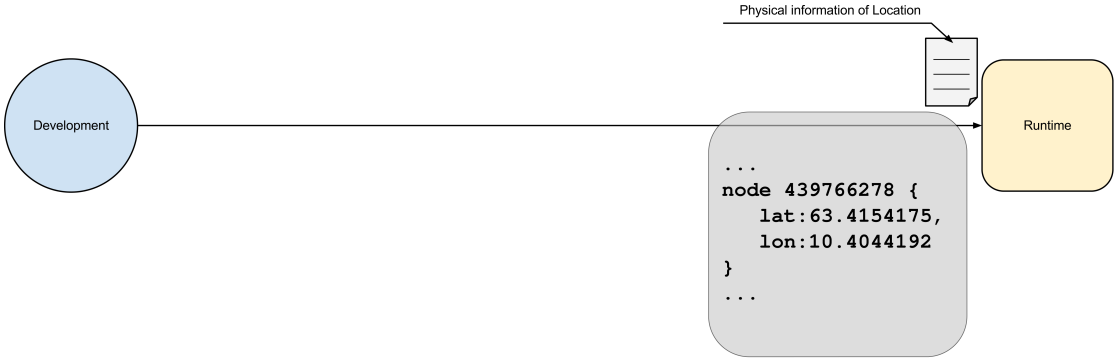


FIGURE 6.1: Simple process flux

This approach is the beginning of a modular approach. Places are not directly defined in the code but in another file. This provides a better and easier code review. As shown in **Fig. 6.1** this approach can be extended by making the conversion between aliases and real coordinates at runtime. In this way the game can be modified without necessarily recourse to recompilation (if the game is written in a compiled language). In addition, this next step we open the doors to a relocatable vision of the game, because, simply changing the aliases file you can edit the places where the game is run. From the programmer's perspective, this approach, although with some positive aspects, however, poses several difficulties, mainly related to the retrieval of physical information (coordinates) and in the drafting of aliases for each physical location.

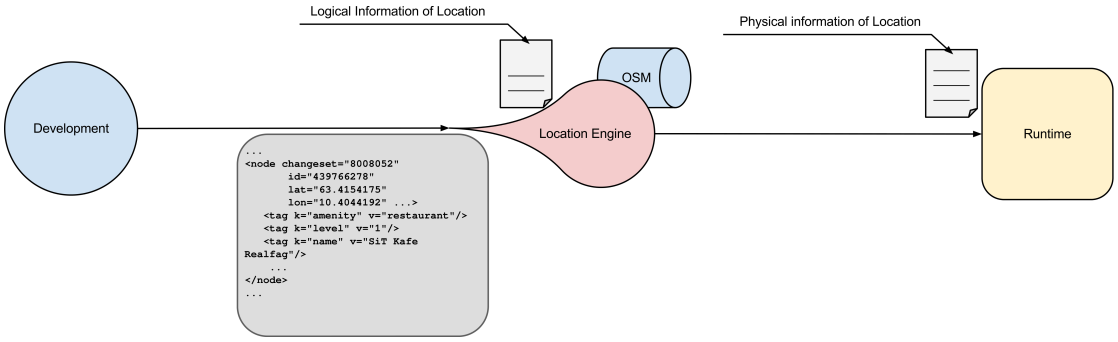


FIGURE 6.2: Process flux with automatically retriever of physical information

The **Fig. 6.2** shows a further improvement of the process of creating the game. In this case the programmer during the development phase refers to the places using a special document. In this document the places are defined in a logical manner, avoiding as much as possible to refer to existing locations, but referring to them in a logical manner. This approach further consists, therefore, the use of an automatic system and one or more external services that are able to convert the logical information that the programmer provides in the document in real references. In this way, during the development phase, the programmer will refer to the various locations during the writing of the code by reference to an external engine which when attached to the logic of the game will be able to use the logical descriptions of the places provided by the programmer in places existing at runtime and return these places to various location systems. We imagine, in fact, that the logic of the game itself is separated from the location system of the player and, therefore, to such a system is only necessary to send the coordinates.

This approach raises, therefore, the developer in the position of having to choose the locations of its game based on logical and qualitative criteria, and not on to geographical coordinates. This development philosophy lays the foundation for a vision of the game relocatable. Defining, in fact, as the concept of region, a particular physical space, represented by a rectangle of coordinates, or also in a logical manner, it is possible to find places "logically" similar in different regions. To better understand this concept, it should give an example. If we mean a city region, we can safely say that the places described in a logical manner as a "Town Hall", "Hospital", "Marketplace", "Church", we can easily imagine that, once decided the city region in which you want localize the game, is not too difficult to think that the game can work in a different city, because most cities have the same type of places described. The same would not be easy in the game if we had used the coordinates of the various places, and even using also alias, because the developer would have to be to look for various coordinates.
Finally, this approach, which dissociates from the physical locations, also allows to dissociate the technology used to identify the location of the player. Until now there has been talk of coordinates since such a system is the easiest, intuitive level and technology, to identify a location. This does mean that we may use other technologies such as image recognition or qrcode, or via cell phone. If the conversion system is associated with an outside service that returns this information, the developer does not have to worry about the conversion.
At this point in the process, what you should cover the developer is to be able to describe locations in a logical manner so that the data source is able to convert them into physical information. The developer have to know the source data, how to manage them, and what information is useful in order to generate the same places that he has in mind.

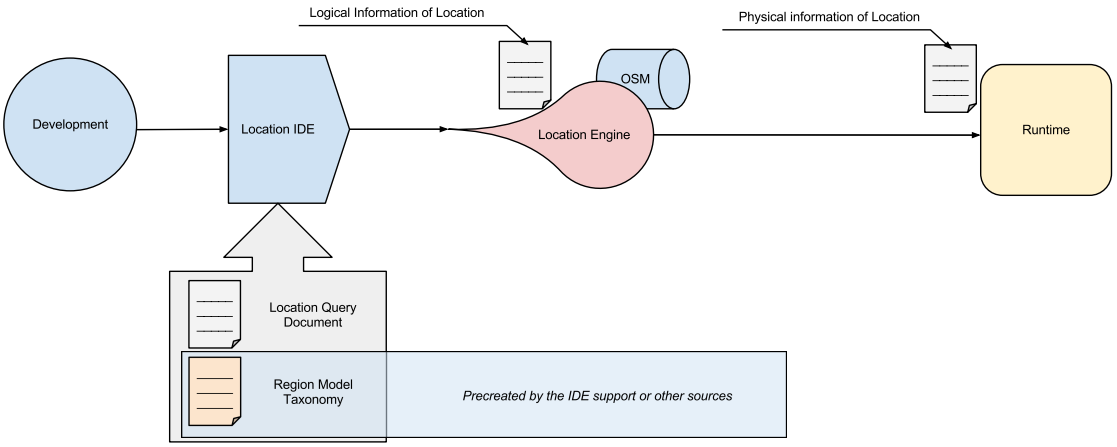


FIGURE 6.3: Process flux with automatically retriever of physical information

Assuming that the studio work of the datasource is made by the same team that provides the conversion system, the **Fig. 6.3** shows the final process that wants to reach. The programmer in this case uses a special IDE to write the document that describes the places in a logical manner. This document, written in a pseudo-language query, will be associated with a specific taxonomy of places. The taxonomy is provided according to the type of region and is composed of classes of places. Each class has inside information to give to the conversion service to extract data from external services and build an instance that represents the physical location to associate with the runtime of the game. In this way, the developer defines with the QL places in a logical manner specific to one type of region, avoiding references to existing places. The system takes this document and the taxonomy of the region and generates a file with instances of the classes of places associated to the specify region. These instances convert to real places will be given to the game at runtime.

The study shown in this chapter, is therefore essential to the development of a specific IDE and the definition of a series of documents, in particular three documents:

Region Model that describe the taxonomy of a specific model of region. In that model it's the taxonomy of classes and the tags used for describe these. For example a model of region could be a model that describes a university campus with a taxonomy that include rooms, auditoriums, study rooms, cafe, library, canteens.

Region Model Instances Document this document will describes the different instances of taxonomy classes in a specific region. It depends on a region model document and it will be automatic or semiautomatic generated by specific engines that querying the different sources. In this document the description of places is only logical and it does't have any reference to physical information. To continue the example of a campus this document will contain the logical structure of

a specific campus with the name of the rooms and logical attributes descriptors (number of seats, furniture etc..).

Location Document this will be the final document linked with the game levels description document. It will contain descriptions of places that the development team want to use in their game. This document it will write with a domain specific languages and it references to other of two document types. With this references it will be possible to create an environment that suggest type of available places and specific places in according with attributes that the team want for places. This document have to have a pure logical nature (without physical references) but it should give the possibility at the team to specify specific places using region documents.

Separate information in different file is an application of the modular approach, widely used in various information technology areas.

This modular approach allow to create different location that using the same model or change some aspect of the model without changes the Location document. In addiction is essential for reuse the Location document because, only the Region Document contains information linked with a specific region. With this modular techniques is more easy to create games that can be adapts to several Regions.

Another aspect is that does not put all the possible Taxonomies in one unique file but, has different files for each type of taxonomy. This is better for two reasons:
first because the game developer can choice only the taxonomy fits better for his game, and that make the game lighter;
second because this allow to create future taxonomy for other type of region, so could be upgraded in future.

Modular approach is useful also for the process of creating Region Document. With this modular approach is possible to generate Region Document in a (semi)automatic way, and this is essential, especially because this kind of documents need to be upgraded when the different sources have more fresh data. A region in the real world is much more different compared with a virtual world. In a virtual world things changes only if the creator of the world want to modify something, and he will change for the game purposes.
In the real world places constantly changes and so it's necessary to be able to upgrades the Region of the game without changes anything else. In this view is useful that the instances of games continue to use this document and don't convert in a document

with real references. In order to develop better an integrated IDE that gives the opportunity to work with these and other types of documents, for the reasons explained in **Chapter 4** has chosen to use Eclipse and EMF, the framework for model driven development. Using these two tools is suitable in order to achieve the stated objectives have required the definition and development of various elements, including:

- A model defines the structure of a Region Model Instances Document
- A model defines the retrieve resources process
- An UI permits to work on that kind of resources and documents

To better understand, however, precisely how to develop these elements, first of all, it will be well to consider the definition of the Region Model and how to retrieve and use resources, a subject already discussed in the **Chapter ??**.

6.2 Region Model

The Region Model (RE) is a document that describes the structure of a specific region. It contains a description of possible types of place that the region can hold and the interactions among the various. RE is essentially an instance of the metamodel **fig. 6.4**, where it is possible to see that each class can interact with another class through two relationships:

- **subclass**: indicates the inheritance relationship. A subclass of another class inherits all the attributes and relations with the tag that has the superclass.
- **contain**: indicates the relationship for which a class of places may contain within it other places in the class. For "contains" it is meant that physically or logically, the place can be found inside the other location that contains it.

This model then allows us to build a taxonomy of a region by defining classes of places, subclasses, and indicating, which may contain other places.

6.2.1 Why using a Taxonomy

Before describing why it was decided to represent a certain class of regions with a taxonomy of classes of sites is good to expose in more depth the concept of taxonomy and the role in this situation.

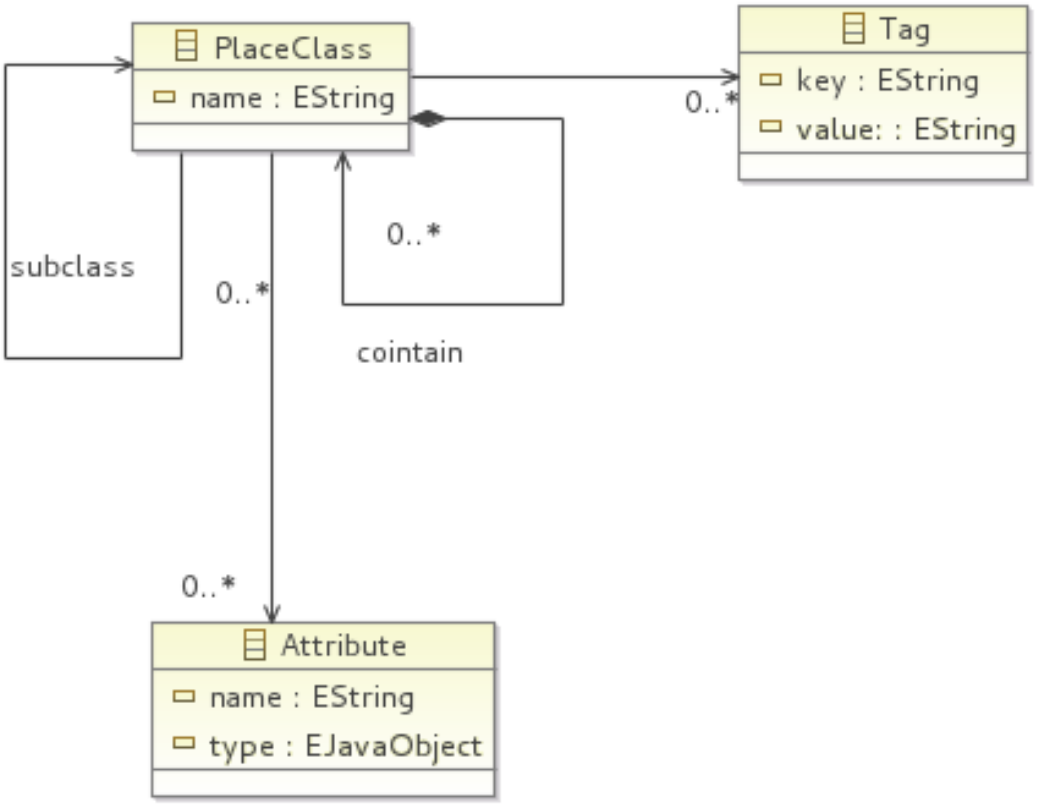


FIGURE 6.4: meta-Model for Region Model

Definition of taxonomy and Why using it A representation of all individual components and their relationships within a finite group, using a predetermined method and rule. It used to be static, predetermined, based on specific rules, without any redundancies, often with model driven approach.

Within this project, it is useful to define a taxonomy when we model a region. In this way we could create a hierarchy among the various classes of places and facilitate the search and the localization of places. This is also useful to the final developer because having a hierarchy based on which, it is easier to choose the class of places that best suits the purpose.
What is going to create is, therefore, a taxonomy of classes, that is very close to the concept of inheritance in object oriented programming.
In a taxonomy, first, there is a general classification (all places are "places") and gradually builds a tree of groups and subgroups, where the elements have common characteristics. If we consider as characteristics common attributes and methods, this is nothing more than create a class and subclasses that inherit characteristics from the main one.

In our taxonomy, there are two levels of interaction with each element can have. The first is to subclassification, which is entirely consistent with the concept of inheritance

found in object-oriented programming. This first deviates from inheriting relationship between elements in some languages, in that, poses no limit to the number of elements that interact. In other words, a class can be subclass of more than one element.
The other type of interaction is called "containment" and represents the physical possibility that an element of a class can contain inside it an element of another class. Being that our elements are places it is possible that a place inside it may have some other elements. Imagine a class of places "city", "mall" or "university", all these places have other elements in them (shops, bars, bathrooms, classrooms, etc ...). This type of relationship, which in fact is another type of taxonomy is less close to the concepts of object-oriented programming, but can easily be represented in it with a variety of different structures (arrays of pointers, maps, etc ...).
The ability to see the taxonomic classification used for the places, from the point of view of the object-oriented programming, much ease the construction of data structures which represent the elements.

Describe a model of the region as a taxonomy ff classes allows us to classify the possible locations for this region in a logical matter and help the construction of a logical structure of the region.
RM by defining a taxonomy allows adapt this model to a model-driven structure and therefore facilitates the definition of tools for creating and managing it. Furthermore, from the point of view of use of such a model for the definition of place documents, the taxonomy makes the search and selection of the various elements more effective. Taxonomies tend to be inflexible and are likely to force you to redo all the work again in case of large changes. But, since the modular approach has been what it's chosen to follow, this is not a major obstacle as such taxonomies will, mostly, specific to a type of region and, therefore, not large, and without a large number of elements. Indeed, the modular approach tends to suggest precisely to avoid documents that describe large regions.
In the case where the game present the need to have a wide range of possibilities, it may be appropriate to use more models of the region in the same place-document.

6.2.2 How to represent a Region Model

In order to build a DSL for the development of pervasive game must be possible to define the various RM in order to give effective tools to the end developer. Furthermore, the same developer, will have the chance to create his own taxonomy for a region when the the existing ones are insufficient.
To create the appropriate tools for define RM documents, it should first decide how the RM will be defined. Having chosen to represent the region as a taxonomy of classes of

places, it is natural to use the model-driven approach which shows that RMs will be a model. To represent this model there are a variety of different techniques. In this case we will focus on two possible implementations and finally draw the conclusions about which is better to use. The two techniques are in particular, **the definition of a DSL specific for the Region Models** or **the use of the Ecore architecture as a representation of the metamodel**.

In order to show the pros and cons of the two techniques it will use a generic university campus as an example of region, having, classrooms, offices, auditoriums, laboratories and all the necessary services.

6.2.3 Region Model with DSL

One way to create RM is to define a DSL act to this. The **code 6.1** shows the beginning of an example of how a document could be with using a special grammar studied for that (to see the entire document ...). This syntax allows you to write quickly and if the taxonomy editor that uses syntax is modeled on that in a few seconds, you can define complex taxonomies. The flaw in this approach is that it is difficult to adapt these documents in different contexts and it's not very easy to extend or using in alternative meaning.

```
region campus

class Education{}

class Campus subclass of Education {
  attribute name : String,
  contains{
    Library,
    Room,
    ATM,
    Parking,
    BusStop,
    Reception,
    SportCentre,
    Cafe,
    BookShop,
    Canteen,
  }
}

class University subclass of Education{
  attribute name : String,
  attrubute city : String,
```

```
    tags{
      {'amenity','school'},
      {'building','university'}
    }
}

class Service{}

class Room subclass of Education {
  attribute room_number : Integer,
}
class MeetingRoom subclass of Room{
  attribute name : String,
  attribute n_student : Integer,
  tags{
    {'room_type','group'}
  }
}

...
```

Listing 6.1: Example of Region Model DSL

6.2.4 Region Model using Ecore

In order to you Ecore architecture to represent our model is useful to associate the various concept of the Ecore Model to our meta model. In particular we will not using all the Ecore framework but only a part. The **Table 6.1** show which elements of the Ecore architecture will be associated to our metamodel.

Region MetaModel	**Ecore Elements**
PlaceClass	EcoreClass
subclass rel.	Inheritance
Attribute	EAttribute
cointain	Cointainment relationship
Tag	EAnnotation

TABLE 6.1: Associaton from Region MetaModel to Ecore Elements

That make easy to adapt our meta-model to various type of application and help to create other tools using Ecore framework. And also it allow to use EMF to design the various taxonomy like **Fig. 6.5** show.

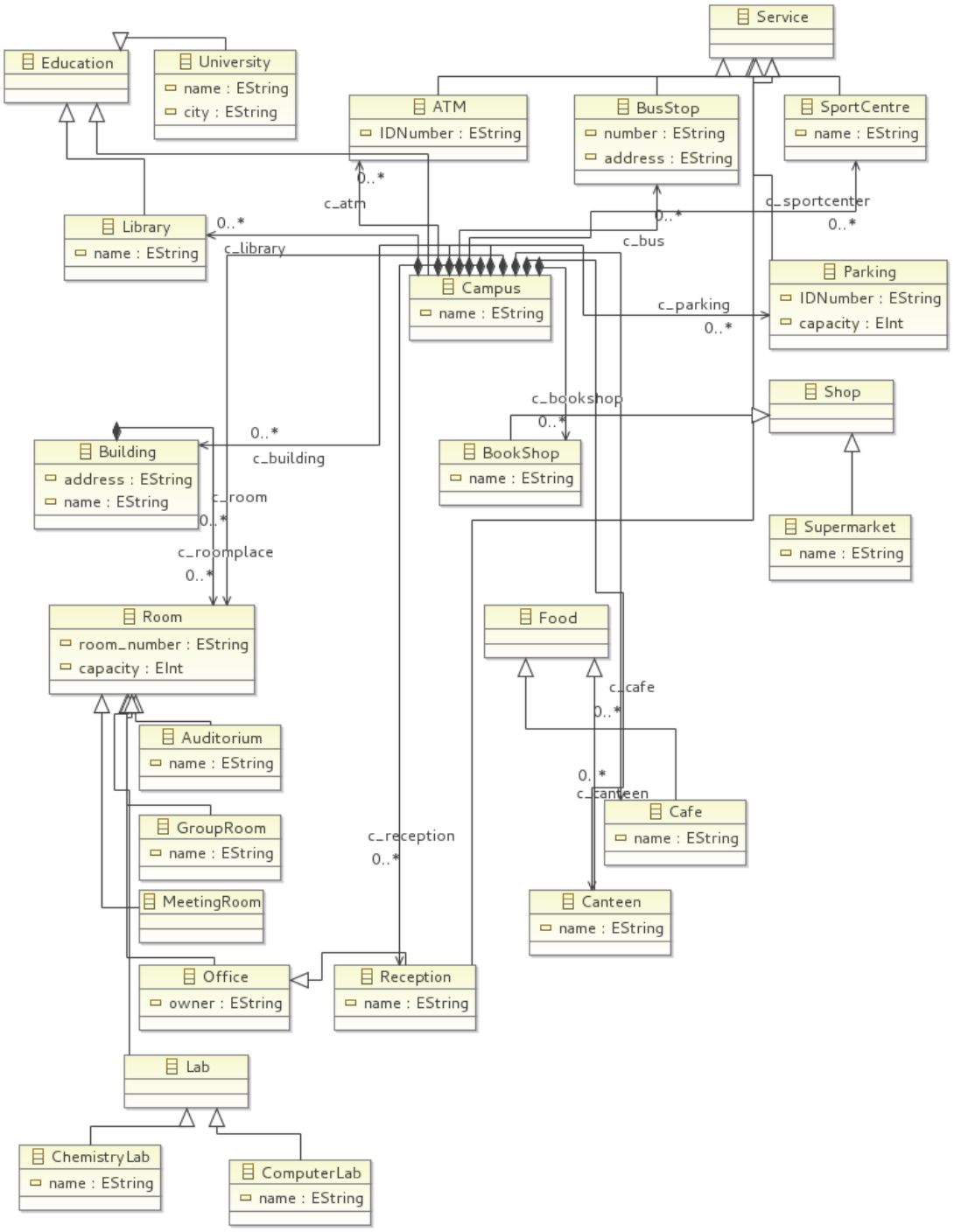


FIGURE 6.5: Region Model of a Campus designed with Ecore Diagram

One things that we can notice about the two approach that the first can be a subproduct of the second. Is possible, in fact, to use the Ecore framework to generate file in a specific DSL that describe an Ecore architecture. Using Ecore is also simple to create Region instances documents that are nothing more than collections of Instances of the EClasses using for described Place Classes. For example the **Fig. 6.6** show an xmi for the model of campus that it's used for example.

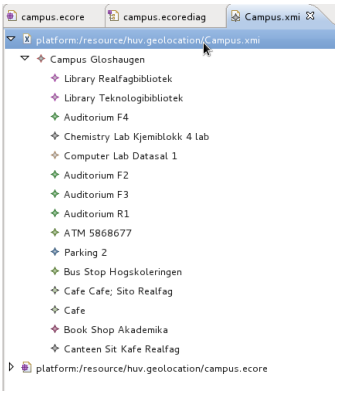


FIGURE 6.6: Xmi structure of a instances for the Campus model

In order to better integrate the production model of the region in this project will use the model form of Ecore for several reasons. First of all, the possibility of having the model in the form of Ecore provides, in an automatic manner, even the existence of a XMLSchema, which integrated with the XMLSchema of OSM or other sources, makes it virtually automatic the data extraction from these sources. Furthermore, as already said, using the Ecore model, it is possible to easily associate with such a model a DSL, and therefore, describe these models with a very precise language which returns an Ecore.
Finally, it integrates perfectly with the other sections of the project where this part fits.

At this point, defining as "middle programmer" who produce models for the various types of regions, it will have provisions to different environments in order to generate the latter. One possible solution might be to extract data sets from various sources from different regions that may be existing instances the model that you want to create, and allow the software to an intersection operation data, to find possible classes of sites, common to the various regions. Once this portion of data, manually constructing the taxonomy according to criteria defined from time to time. The software can automatically create an Ecore already with all the classes of places and give the "middle programmer" only the burden of deciding the relationship.

6.2.5 Defining Element in the Region Model

In order to create a useful model is necessary to clearly define what can be an element of the taxonomy and what not, i.e. what classify and put in the model and what to leave out.
Also, since each model's class will be associated with a collection of tags it's needed to use a systematic method to choose which tags to associate with a class.
A method to create the model is to use one or more instances of the region and consider what elements are present in these regions and how they are described (with tags which, how many nodes, etc ...). Once this study has been done, you can begin to create the desired taxonomy. More will be the instances of the region considered, there will be more likely that the model represents efficiently also other possible regions.
During this study (which can hardly be done automatically) is possible has difficulty in selecting the most appropriate tags for a given class. As mentioned in **Chapter 5.2** tags are the only way to identify the nature of a node, but they have different functions without considering that, being based on a folksonomy, it is far from said that a tag is always used appropriately or always used to identify a certain type of element.

For these reasons a detailed study on the nature of the tags for a given element is essential in order to create a usable model. To this issue is also added that the disparity of information. If for example we consider of the region as a model of the University Campus and select different campuses around the globe, we will immediately see that by looking at the data extrapolated from various sources, they can be very detailed and very poor for a campuses to another. A model that represents a region too small (in terms of geography) may be completely unusable on some instances of the region for pure lack of information linkable to classes.

In conclusion to create a model for a region is necessary to study more samples of the region and look for the best possible intersection of possible tags for each item so as to have more class-instance pairs as possible.

6.2.6 Region Base Model

Since the IDE will use directly Ecore model to represent Region Model, in order to use that we cannot image to create for each Region Model a Generation project in EMF and a plugin itself. In that case the work of the developer will be much more and there will be a specific IDE for each kind of Region Model, and that is absurd.
In order to create a unique IDE that can use ecores to represent region models we need

another model, similar to the "metamodel" showed in **Fig 6.4**, but more generic that have the only two purposes: first to give a way to recognize if a generic ecore file could be considered a Region Model; second to link the selected ecore to the file with the serialization of the instances.
This model it will call Region Base Model and will have the structure showed by **Fig. 6.7**, and will be the model using by the EMF generation project for generating the code for the IDE.

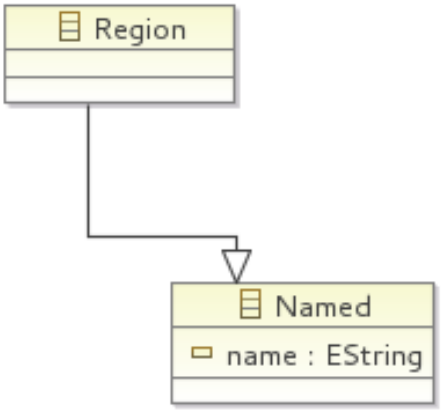


FIGURE 6.7: Region Base Model

This model imposes that every ecore want to represent a Region Model have to have at least one classes subclasses of Region that will indicate the main containment for the other classes. In other word, there will be a classes of place that represent all the region and where all the other classes are contained by that.
It also have another class that will be superclasses of all the classes will have "name" as a attribute, to make the collection of tags for that purpose not redundant inside every Region Model.

The **Fig. 6.8** will show the same structure of a model for University Campus showed in **Fig. 6.5** but with the connection between this and the base model.

6.3 Region Model Instances Document

The IDE, which will be described in **Chapter II** will use the Region Model Document described in the previous chapter, and the data retrieved from the source/s related to a specific region in order to create the so-called **Region Instances Document Model (rmi)**.
This document, then, will contain the instances of various classes of places in taxonomy of the Region Model, refer to the specific region for which data will be extracted.

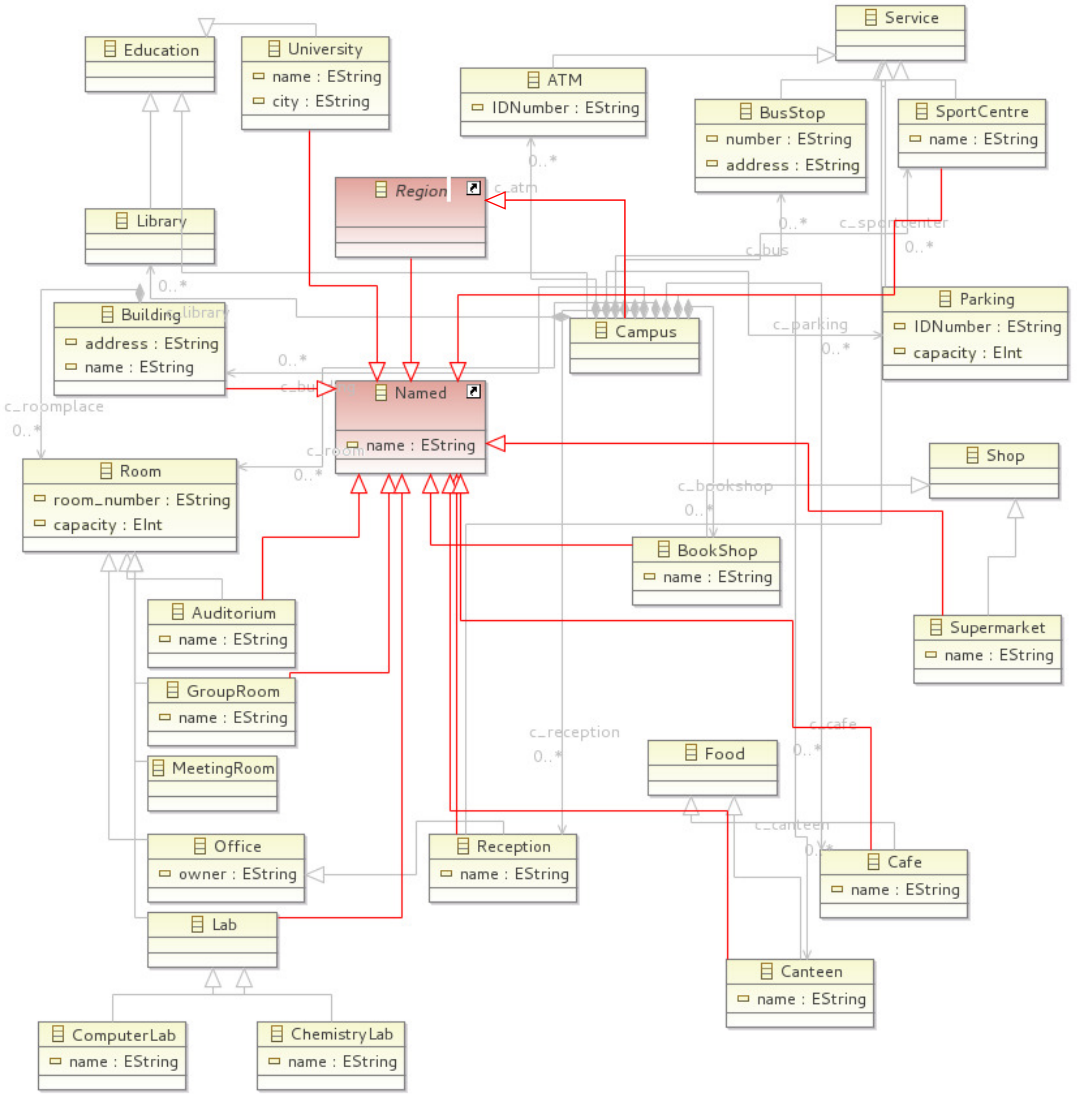


FIGURE 6.8: Campus Model with the links to the Base Model

Having decided to use as the implementation of the Region Model the Ecore model, automatically make the new document, neither than a xmi file, containing the serialization Region Model.

To better understand the nature of this document and its use is necessary to make an example. Imagine using the model of campuses displayed in **Fig 6.5** and, suppose now you want to use only OSM as data resource. Imagine, then, you want to generate the rmi on the campus of NTNU in Trondheim (Norway) called Dragvoll.
Of course, it is necessary that the classes in the region model, will be associated with tags and that them will be consist with the source and consistent with the study done on that campus and on campuses considered similar.

Once this is established the process of building the file rmi consist simply in traversing the data collected from the source and for each element of this collection attempt to

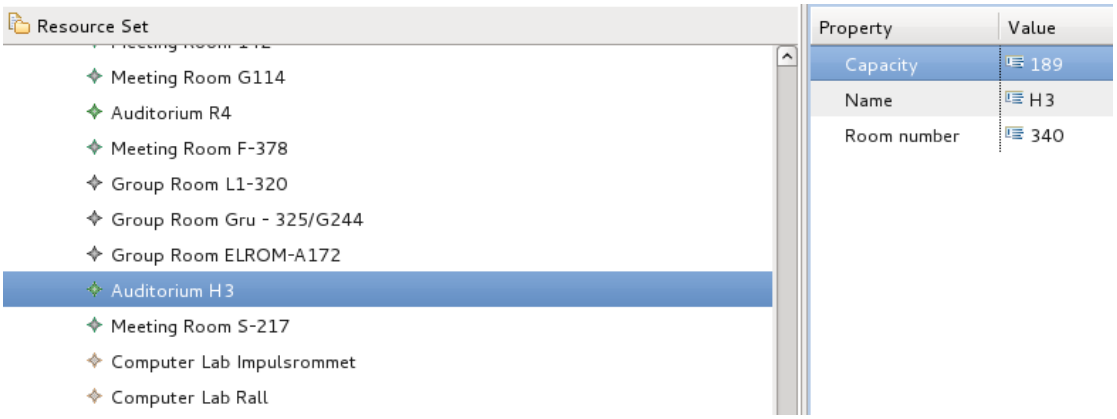


FIGURE 6.9: Example of element in the rmi

find which class of the taxonomy could be matched. Better is the creation of the Region Model and its study described in the **para 6.2.5**, much more will be the matches between the sources data and the classes taxonomy.

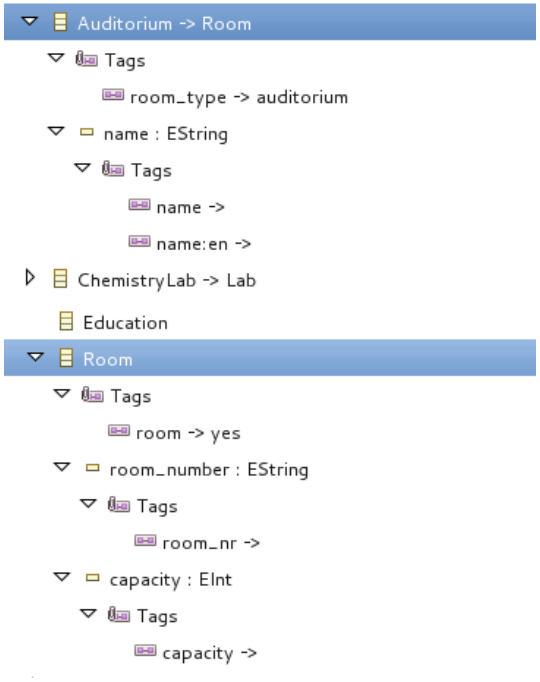


FIGURE 6.10: Auditorium class in the Campus Model

During the processor of creating the instances, for each classes the process will try also to using the tags structure in the sources data to collect information for compiling the attributes define in the class, if present. The result will be a collection of instances very different from one to another, because of the different nature of the classes. The **Fig. 6.9** shows an example of how could be a generated element in the rmi file, using the interface standard of EMF. That rmi was generated using the Campus model shows before for Region Model. As you can see not only the instance of the class will generate but also the software will collect all the possible information from the tags in order to compile the attributes specified for that class. The **Fig. 6.10** show how it's define the class in the campus model.As you can see the attributes (from Auditorium class and Room class) have indicate, like annotation, which tags have to use to collect the information. It's possible to see that the main difference between tags using for recognize the classes and tags using to fill out the attributes, is that the last one don't have the *Value*'s value. That because the value itself will be using like the information.

As described in the **para 6.2.6** the rmi is a collection of instances of the selected Region Model but have to have at least one instances the class region in the Base Region Model so it can be shown on the editor in the proper ways.

6.4 Location Document

The Location document is the final result of the whole process, which, once processed, will be joined with the other documents of the game, especially to the document that defines the various levels and tasks. Once defined, or recovered, the Model Region, and once chosen a region and generated the RMI, it possible to start drafting the Location Document.
When choose to create a new Location document, it need indicates the files of the Region Model (Ecore file) and RMI of the region selected. At this point the DSL is configured and it possible to write to the Location Document according to the structure defined in the **Code 6.2**.

```
name <name of the location document>
model <ecore file with the region model>
region <rmi with the region instances>

places {
  place <identifier of the place> is <main container class>.<subclass
    >[.place in the rmi]
    [nearest <identifier of place| <class or subclass>] [and|or]
    [width <attributes of the class> <operator> <value> ...
}
```

Listing 6.2: Structure of a general Location Document

In practice, *places* contains a collection of locations named with *place*. Each *place* has a unique identifier that is used in the other documents within the game to refer to the specific place. After is defined the type of place that can be a class in the taxonomy (the editor, once defined the Document Region, will display the possible class) but can also be a specific instance in the RMI file. This gives the possibility to choose a specific site in the target region but also to leave to a heuristic in the software to choose the instances by defining the class. The advantage of not simply declare the instance but the class is being able to reuse that file for different region who share the same Region Model.

After defining the type of place it can define additional parameters to ensure that the heuristic will select the instance in the file may have multiple rmi parameters and choose

the location closest to the idea of the programmer. These rules typically consist of two types:
The first is defined with the keyword *nearest* followed to an identifier of another Place defined in the document, of a class of the taxonomy, or of an instance in the RMI. This rule binds the software to choose between the instances of the class chosen only the one closest to the constraint. Suppose you want to define as place a bar and in the region there are more than one, and you want the bar near the bus stop, with this rule you can define exactly the bar closer to that station.
The second rule allows you to make a selection on the attributes of the selected class. In this way you can narrow the range of possibilities and possibly narrow it down to a single element if the combination of attributes is sufficient.
These rules may be more than one and joined together with the terms AND, OR, and NOT.

Once defined the place the editor provides an validator that in addition to checking the grammar, informs the programmer of the possibility that one of the places do not have a match in the file rmi. In this case it may mean that the program failed to find a correspondence between the class chosen, the rules defined and instances within the region.

Once the document is valid, you can generate an XML file containing the place referred to by their identifier, geographic coordinates, obtained from the source data, and other information useful to the game.

This is the file that the game will refer to when evaluating the position of places, compared to the player, or simply for viewing.

Chapter 7

The IDE

The project's IDE basically consists of two editors, with the corresponding wizard. The first editor mainly concerned with the management of the Region Model and generation of RMI files. The second editor instead offer a user friendly environment in order to compile the Location Document.

The first editor is a plugin generated with the EMF framework, the second one instead is a DSL design with xText.

7.1 Region Model Instance Editor

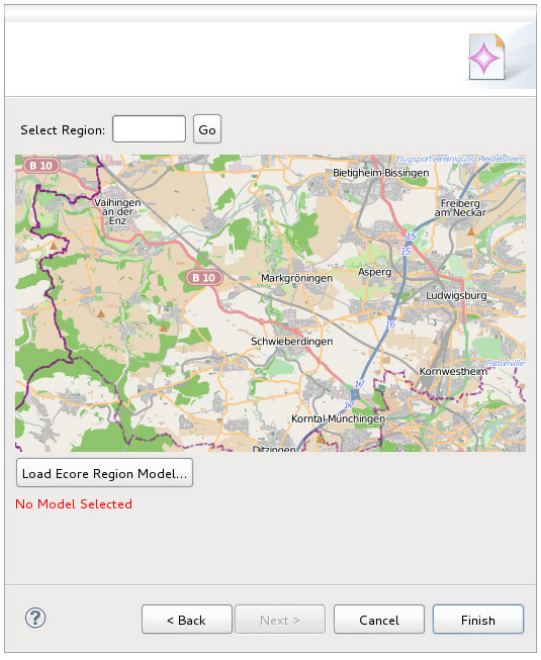


FIGURE 7.1: RMI Wizard

This editor allows you to generate the RMI for a specific region. But beyond that, has a convenient interface to manage the model selected in order to create a file as useful as possible.
Once you have decided to create a new file RMI starts a wizard that allows you to set the initial working environment.
After a page to decide the name of the file and to associate it with the project, a window shown in Fig srf fig: rmi-wizard allows the selection the Region Model and the initial region by a map.

It has a widget that displays a map of OSM (the widget is drawn from a project

that aims to create an interface for OpenStreeMap in Eclipse [25]). A text box allows you to move the map on a point, writing an identifier for that point. This function is implemented by making a GET request to the service Nominatim shown in **Sec. 5.4**, and once you got the result it is parsered by a simple XML parser that selects the first coordinates available in the document and after the move coordinates on the map above. This heuristic is based on the fact that Nominatim sort the results nodes in a descending order according to a value that represents the level of consistency to the value of the query.

Below the map there is a selector that allows to select the Ecore file with the desired Region Model. After finishing the wizard, the editor for the Region Model Instance

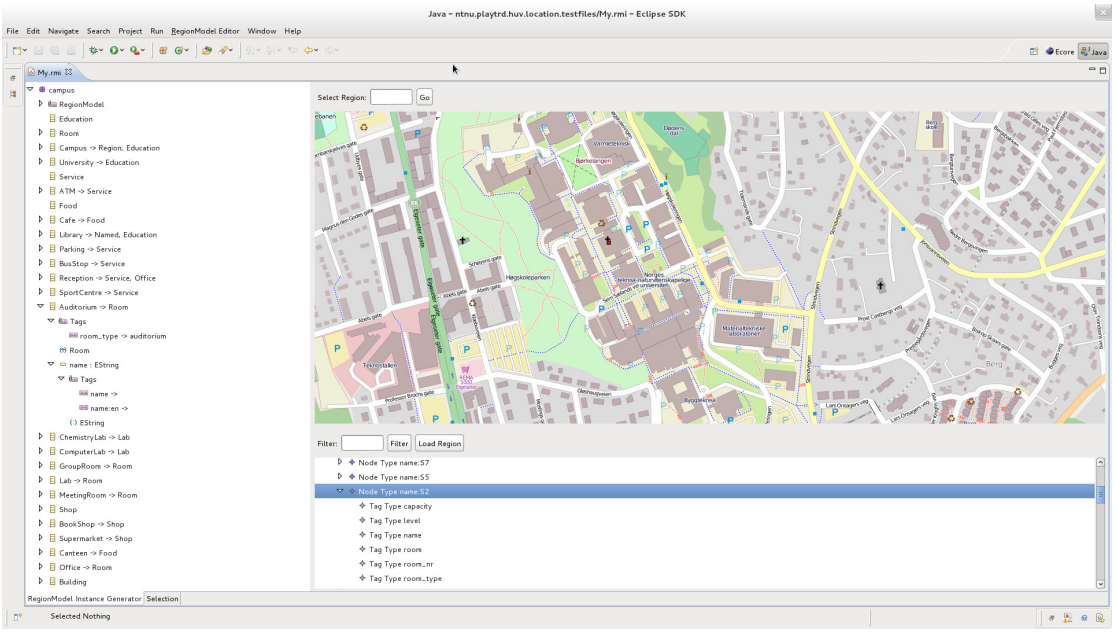


FIGURE 7.2: RMI Editor first tab

Document appears. It consists of four main parts. The **Fig. 7.2** shows the first of the two tabs that compose the editor. This screen is divided into three sections. On the left there is a tree view that give the possibility to edit the Region Model in the ecore. In the center there is a map, always implemented with the widget described above, which allows you to select the region which the data have to extract for the generation and also displays the location of the nodes selected in the section below. In this last, in fact, appear in the form of tree the data results from a request to OSM of the region that appears in the map, when you press the button "Load Region".
This part has provided the ability to filter the nodes by a string. This filter acts on the tags associated with each node and displays only the nodes that have a tag which at least the key or the value corresponds to the given criteria.

At the top there is a text box with a function identical to the one included in the wizard, that is to jump up the map to a specific point.

Once the model is defined and that the data of the desire region is loaded is possible, via an entry in the drop-down menu at the top, it's possible to populate the RMI file with instances generated by model and region.
The function, that generates such instances, work by iterating twice the osm file with the nodes of the charged region. For each node searches for a corresponding class in the model region, equating tags owned by the node with the annotations within each class. If the node match with a class, an instance of that class is generated and added to the collection, otherwise the node is discarded.
After generate all possible instances is started a second cycle, always on the nodes of the region, which aims to collect data within the tags to populate the attributes, if the class have any, of the various instances. When this cycle ends, all the generated instances are written in the rmi file.

The second part consists of an extension to the Editor *Simple Reflective Editor*, already present in the EMF framework. This is made to allow the programmer to see the instances generated by the editor.
The **fig ??** shows what it looks like. As can be seen depending on the type of class, there are icons of different color right to display that there are instances of different nature. It is possible to notice also in the box, *Properties* that all the attributes of the class are carefully compiled from the osm data.

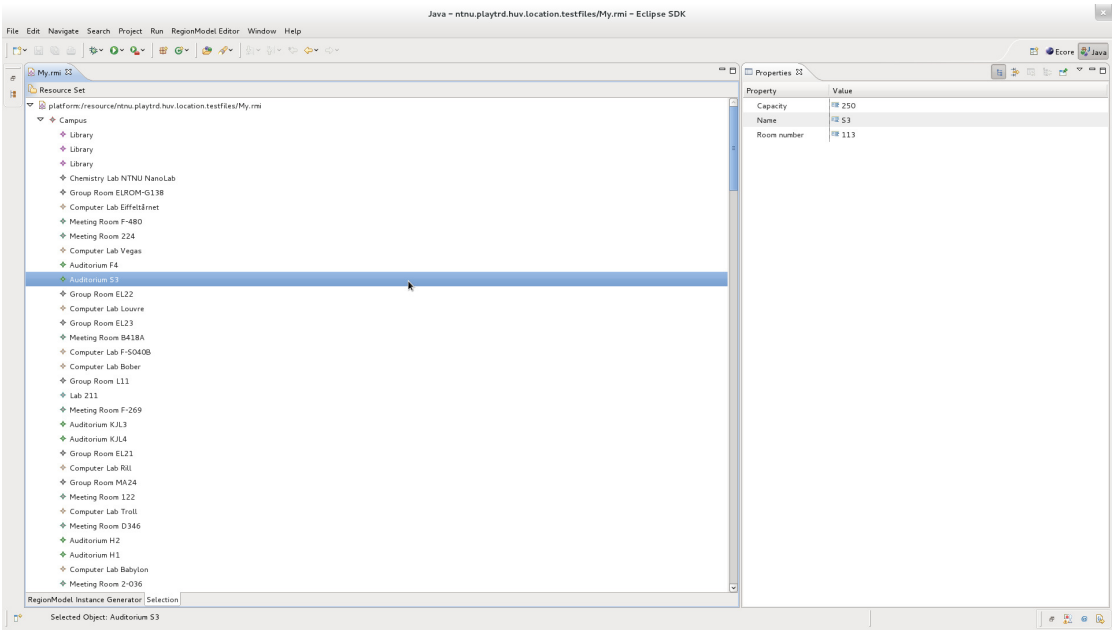


FIGURE 7.3: RMI Editor second tab

7.2 Location Document Editor

The editor for the Location document is a simple text editor with auto-complete function, evaluation of grammar function and generation of files with real physical locations data of the Places selected and entered in the Location Document, as explained in **Par ??**.

Once selected a new document a wizard appear,and, after the usual screen for entering the name and to choose of the project, there is another window that allow to select the ecore containing the Region Model and the RMI file containing the instances for the region (this selection is optional if the programmer want to work with only classes but will be request when he tried to generate the final file.

After the wizard, an editor similar to the classic Eclipse Java editor allows the developer to start compile the Location Document in accordance with the rules explained in the **Par. 6.4**.
Whenever it start to declare the class of membership for each Place, a drop-down menu will suggest which classes use, based on the taxonomy contained in Ecore through a behavior quite similar to the selection of classes during writing Java code.

Once finish to write the Location Document, through the *Validate* option int the menu at the top, it's possible to validate the document.
The validation, consider the grammar, and, also, tries to generate the file of the real locations, if an RMI file has been selected, searching for every Place an instance declared in the rmi file to associate. In case this should not possible, ie if the function of association could not find a suitable instance, an error is displayed with reference to the place that is not associated.

In particular the method that validate this document operate as followed:

- If the Place had explicitly stated the instance, and associates the following changes to evaluate the next Place.
- If only the class is declared, search for all instances with of class and will choose the first match.
  The error will be displayed only if for that class any instance are present in the rmi file.
- If there is the *nearest* rule, the function works as in the previous case, but, instead of returning the first instance, it will search the instance with the closest coordinate to the point specified in the rule.
- If there is a rule for the attributes it valuate the condition and the error is displayed if any of the possible instances is consistent with the attributes highlighted.

Obviously this control follows the previous one, though both declared, and does not run if any of the above checks has already give an error.

Finally, when the document is evaluate is possible to generate the file to associate to it with the information of physical places.

Part III

Conclusion

Chapter 8

Summary

The work described in this document had the aim to realise an IDE for the location definition, part of a entire framework for developing pervasive games.

In the first phase the work for focused in the choice and study of the tools to use, followed by an in-depth study of the data source and how to interact with that.

In the second phase there was the discover of the result to produce and the design of the production process for this result. This phase was useful also to redefine and revise the tools to use and how to implement properly the IDE. In particular to identify the elements to develop in the user interface and which function needed to develop.

In the third phase there was the production of the various parts. In particular the creation of the editor and the function for populate the Region Model Instances. After there was the creation of the wizard and the other editor for the Location Document.

Finally there was the testing phase (that was also in the other phases in order to create a functional editor step by step) and the review statements.

Chapter 9

Practical Utilization of the framework

The framework devised and described in the preceding pages has been designed, as mentioned in the introduction, in order to provide an instrument which simplifies the definition of the locations for a pervasive game. In the context of the pervasive game, this is the perfect framework to define the location of a game task, a treasure hunt or similar, i.e. those games where the achievement of a location is an integral part of the game and not just a boundary condition .
The framework may also be used in games where there is not the existence of a path or the need to achieve necessarily the locations. For example, it could be used to define the position of objects or targets in the context of a shooter pervasive. Or, again, one can use it to decide the positions of virtual opponents in an RPG with pervasive augmented reality. Extend the area of software application outside entertaining, this framework can be used to manage the positions of virtual tour guides for cities or monuments.
A possible application, for instance, would be made by defining with the framework a tourist guide of a city and use the logical information for extract information directly from Web resources for each attractions.

As can be seen from the above examples, the field of applications of this framework is quite wide and its peculiarity to provide the programmer the ability to use patterns and settings already programmed in different places, changing only the region and recreating the file locations, also makes it a very versatile tool.

Finally, the IDE design could also be used in tandem with navigation systems in order to create ”smart” routes starting from logical information. For instance, it possible to build an application that, given the point of departure and arrival, build an itinerary based on preference settings that the user chooses.

Chapter 10

Future development

This IDE project was to be part of an entire framework for the development of Pervasive Game, which is being developed within the University. The project, due to its size, has been divided into several parts, similar to this. The first development that should be implemented, would be to combine these projects into a single product, which requires a study to standardize the various products obtained from different part.

But, since this IDE is complete and working, it could be integrated into other frameworks for building other types of application, as suggested in the previous chapter. To achieve this, the possible outcome could be the addition of other sources of information, as other topographic sites, and sites containing private information, such as yellow pages or similar.

As described in the document prior to this project [6], another development could be the implementation of a system to use the *time dimension*,instead of just the spatial. Would it be possible, therefore, associate tasks of the game to cultural events or entertainment so that the game can be played in specific time periods.
Imagine, for example, the creation of an interactive digital game that takes place during a music festival. The tasks may be, for example, collecting for different events, specific pictures (such as "during the concert of that singer must be able to photograph the face of the bassist) and link the game to an image recognition system. In order to do this, added to the framework data sources like website of events or calendars, could be a remarkable development.

A further development, may improve the system of creation and modification of the Model Region.
Could be constructed a tool to build a semi-automatic skeleton of the model and optimize, then, manually. The tools would take as input XML files returned by OSM for

various regions and select the nodes that have common tags and build a class specifically for each set of nodes with common characteristics. In this way, the model, once refined, could certainly be used for those regions from which it is extracted. This would facilitate a lot the design of the region model that to the state of art is only possible from scratch.

Finally, integrate the system with the possibility of modifying the data of OSM, could facilitate the work of the developer and enrich the OpenStreetMap project. For instance, when the programmer is not able to match a model's class with the place he want to use in the game, because that place does not have the right tags suitable for this purpose, not for a lack of consistency with them, but simply because of gap in the OSM database, could be good to have an easy way to add these tags directly in the OSM data. It would benefit from this system and the entire community that uses the OSM as a simple mapping provider, or for the other possible applications.

Chapter 11

Personal Conclusion

Working in a completely different environment like the University that host my project was a very useful and constructive experience. In particular working together with highly experts was very interesting and made me able to learn how to work efficiently.

The subject of the project was very interesting and taught me a lot about sector of developing that I did not know before.

I'm grateful to had the possibility to face with a professional work and I hope that in the future my products will be useful for all the community of game developers.

Bibliography

[1] Indie game makers dominate ios and android. URL http://blog.flurry.com/bid/82758/Indie-Game-Makers-Dominate-iOS-and-Android.

[2] J. Kuittinen, A. Kultima, J. Niemelä, and J. Paavilainen. Casual games discussion. In *Proceedings of the 2007 conference on Future Play*, pages 105–112. ACM, 2007.

[3] T.T. Goldsmith and E.R. Mann. Cathode ray tube amusement device. *United States Patent Office*, 1948.

[4] M. Montola, J. Stenros, and A. Waern. Pervasive games theory and design. *MK ISBN 01274853*, 2009.

[5] C. Schlieder, P. Kiefer, and S. Matyas. Geogames: Designing location-based games from classic board games. *IEEE*, 2006.

[6] Andrea Mannarà. Location-based games and the use of gis information. Technical report, NTNU.

[7] Bian Wu and Alf Inge Wang. A pervasive game to know your city better. *International Games Innovation Conference*, 0:117–120, 2011.

[8] Pilar Castro Garrido, Guillermo Matas Miraz, Irene Luque Ruiz, and Miguel Angel Gomez-Nieto. Use of nfc-based pervasive games for encouraging learning and student motivation. *Near Field Communication, International Workshop on*, 0:32–37, 2011.

[9] About moodle. URL http://docs.moodle.org/22/en/About_Moodle.

[10] Bryan Bergeron. *Developing Serious Games*. Course Technolgy, 2006.

[11] Parallel kingdom: The iphone's first mmorpg, 2008. URL http://toucharcade.com/2008/03/21/parallel-kingdom-the-iphones-first-mmorpg/.

[12] Definition of location, oxford dictionaries. URL http://oxforddictionaries.com/definition/location.

[13] Geographic coordinate system. URL http://en.wikipedia.org/wiki/Geographic_coordinate_system.

[14] *Global Positioning System, Standard Positioning Service. Performance Standard.* U.S. Department of Defense, September 2008.

[15] About the eclipse foundation. URL http://www.eclipse.org/org/.

[16] *Eclipse Platform Technical Overview.* Object Technology International, Inc., February 2003.

[17] Definition of model, oxford dictionaries. URL http://oxforddictionaries.com/definition/model.

[18] *OMG Unified Modeling LanguageTM (OMG UML), Infrastructure version 2.4.1.* Object Management Group (OMG), August 2011.

[19] Joaquin Miller and Jishnu Mukerji. *MDA Guide Version 1.0.1.* Object Management Group (OMG), June 2003.

[20] Tim Bray, Jean Paoli, C.M. Spenberg-McQuenn, Eve Maler, and François Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Ed.).* W3C, November 2008.

[21] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0.* Addison-Wesley Professional, 2nd edition, 2009. ISBN 0321331885.

[22] *Xtext 2.1 Documentation.* The Eclipse Foundation, October 2011. URL http://www.eclipse.org/Xtext/documentation/2_1_0/Xtext%202.1%20Documentation.pdf.

[23] Open mapquest http://open.mapquestapi.com/xapi/. URL http://open.mapquestapi.com/xapi/.

[24] Nominatim: Development overview. URL http://wiki.openstreetmap.org/wiki/Nominatim/Development_overview.

[25] Swt mapwidget. URL http://mappanel.sourceforge.net/swt/.