StudyAbroad
Design Description

Version 1.05

Revision History

Date	Version	Description	Author
2002-00-00	0.01	Initial Draft	DSD staff
2012-11-09	1.00	First version of the document	StudyAbroad project team members
2012-11-11	1.05	Slight changes in the entire document	Branimir Lochert

Table of Contents

1. Introduction

1.1 Purpose of this document

This document defines the design of the system which will be developed during the StudyAbroad project. The information contained in this document will be used during project implementation. Next chapters describe the details of the domain model, database and interfaces that are going to be used in the StudyAbroad application. This document is written before the implementation of the system and it is expected that there will be some changes in the future.

1.2 Intended Audience

This document will be mostly used by system developers and people who monitor the project development.

Intended Audience:

- Project Leader
- Team Leader
- Supervisor
- Leader Developer
- Developers (server side)
- Developers (client side)

1.3 Scope

This document provides details of the implementation design and architecture of the system. All parts of the application that will be made must be in accordance with this document. Any changes in the implementation details of the application must be added in the future versions of this document.

1.4 Definitions and acronyms

1.4.1 Definitions

Keyword	Definitions
Visual Studio	Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft.
Object-relational mapping	Technique for converting data between incompatible type systems in object-oriented programming languages.
nHibernate	nHibernate is an object-relational mapping (ORM) solution for the Microsoft .NET platform:

1.4.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
VS	Visual Studio
DS	Data Source
OR mapping	Object-relational mapping
ER diagram	Entity–relationship diagram
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
AJAX	Asynchronous JavaScript and XML
XML	Extensible Markup Language
API	Application Programming Interface
ODSM	Object Data Source Mapping
UI	User Interface

1.5 References

Project homepage: http://www.fer.unizg.hr/rasip/dsd/projects/studyabroad

2. Software architecture

2.1 Conceptual design

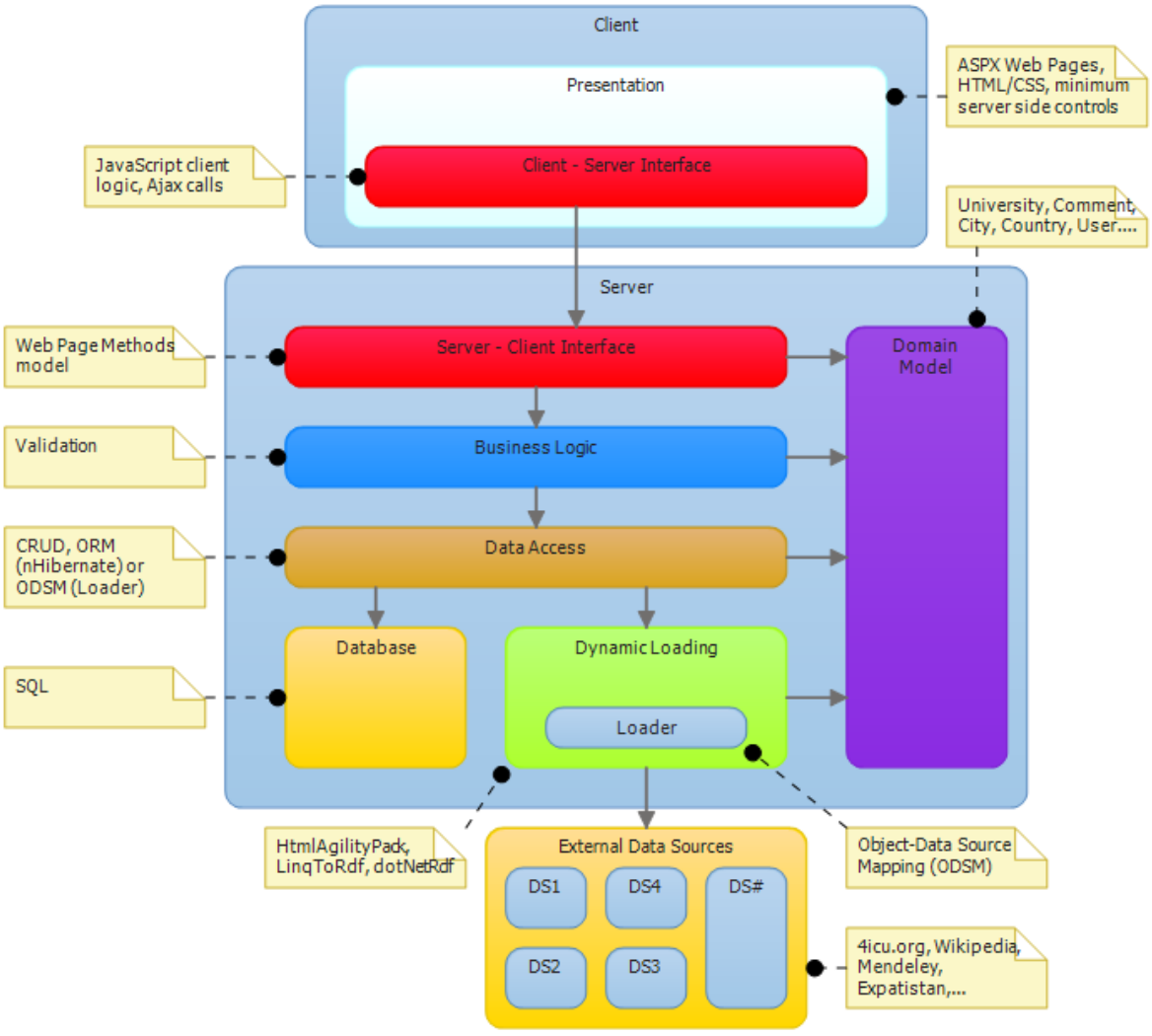

Figure 1: Conceptual diagram

System is made of two parts, client side and server side. Server side gets the information and prepares it so that the user side can get it through the interface.

2.1.1 Domain Model

Domain model is a component used by all server components. It presents a group of classes and their attributes which describe the domain and it will contain classes such as University, Comment, User, City, etc.

2.1.2 Dynamic Loader

StudyAbroad application presents some information that is dynamically retrieved from external sources and some information which is stored inside the local database. Dynamic Loader component retrieves information from external open data sources and presents it to the Data Access Layer packaged inside of domain model objects. Some external sources offer APIs for data retrieval while others need to be parsed/scraped. HtmlAgilityPack is a library which will be used for html parsing, and for data retrieval from sources which offer APIs we will use libraries such as LinqToRdf and dotNetRdf. Some APIs also provide their own set of libraries.

2.1.3 Data Access

The Data Access component queries information from either the database or from repositories returned by the Dynamic Loader component and offers it to the other parts of the application. Other components don't need to know where the information is coming from, as they are using them in the same way through the Data Access interface.

Data stored in the database tables (relations) is mapped to objects with nHibernate (ORM tool), and the data from external data sources (OData) is mapped to objects with the Dynamic Loader (ODSM tool).

2.1.4 Business Logic

The Business Logic component contains all the classes and methods which constitute the server side interface endpoint towards the client. It is used as a wrapper for the Data Access component by forming methods of the server side interface and as a validation checkpoint for all data persisted in the local database.

2.1.5 Server – Client Interface

The server – client interface is presented by the business logic methods and properties which encapsulate the inner workings of the server side components and externally expose only what is needed by the presentation component to accomplish the system functionalities.

2.1.6 Presentation

Client side holds the components and modules which present data to the user. Presentational component retrieves the data from the server through the client – server interface and presents it on the user interface. User interface uses HTML for defining the structure, CSS for defining the presentation and JavaScript (jQuery) for defining the client logic of the pages which constitute the presentational component.

The client – server interface is implemented by a set of JavaScript functions which are using Ajax calls for communicating with the server exposed interface via ASP.NET web page methods model.

2.2 System specification

Operating system that is required on the server is Windows Server 2008. The user interface of the application is accessed through commonly used web browsers such as Internet Explorer, Mozilla Firefox and Google Chrome. Programming languages that are used are C# on server side and JavaScript (with jQuery) on the client side.

2.3 External Components

IIS 7 is used for the web server application, SQL Server 2008 for the database management system and nHibernate for object relational mapping. We will use .NET Framework, and for data gathering we will use HtmlAgilityPack library for HTML parsing and dotNetRdf library for data sources with API. Besides that we will use ASP.NET Framework, jQuery, JQuery UI libraries and Ajax technology.

3. External interfaces

3.1 Software Interfaces

System is communicating with external data sources through the interface. Certain data sources offer APIs, but some data sources we need to parse to get requested information. Dynamic Loader is our component for communicating with external data sources. It retrieves exactly the information we want and exactly when we want it. The way the Dynamic Loader works is described in section 4.3.

Application depends on the availability of external data sources, because of the fact it is retrieving the data at the moment when the user requests it.

3.2 User Interfaces

As shown in figure 2. we divided the pages of our application in three groups: Introduction, Choice, Content.

3.2.1 Introduction

This group refers to the main page. Here we explain to the user what our application is about and how it can help him to reach his goal (to find a university). We also provide here the entry point for the navigational paths.

3.2.2 Choice

The choice group of pages leads the user to reach his decision providing him suggestions or other general information about cities and universities to narrow down his research. These pages contain the links to the actual content pages.

3.2.3 Content

The content group is composed of pages describing specific cities or universities. Here the user will find most of the data we retrieved from various data sources.

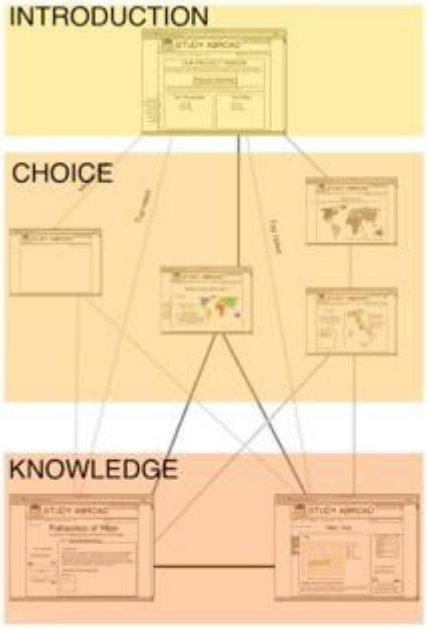

Figure 2: Application pages groups

In the home page we want to show the navigational paths that our users can take in order to access the content pages (city and university) starting from the entry point of the application. The paths are: Suggestion, Explorative and Direct. Every path allows the user to reach the content page easily with a minimum of three clicks for from the *Suggestion* and *Exploration* and two clicks for the *Direct* path. The Suggestion path strongly depends on the number of questions used to understand user's preference and the way it will be displayed.

Figure 3: Home Page

Another functionality which will be implemented is browsing the universities by location and a wizard to guide users towards the decision on where to study abroad.

Figure 4. Browsing and wizard page

4. Detailed software design

4.1 Component model

Figure 5 shows the component diagram of the entire system. As you can see the architecture is a typical client server architecture with multi layers server side.

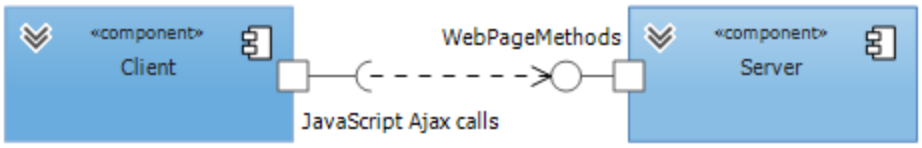

Figure 5: Component model of system

Next figure shows the detail decomposition of the server on major components.

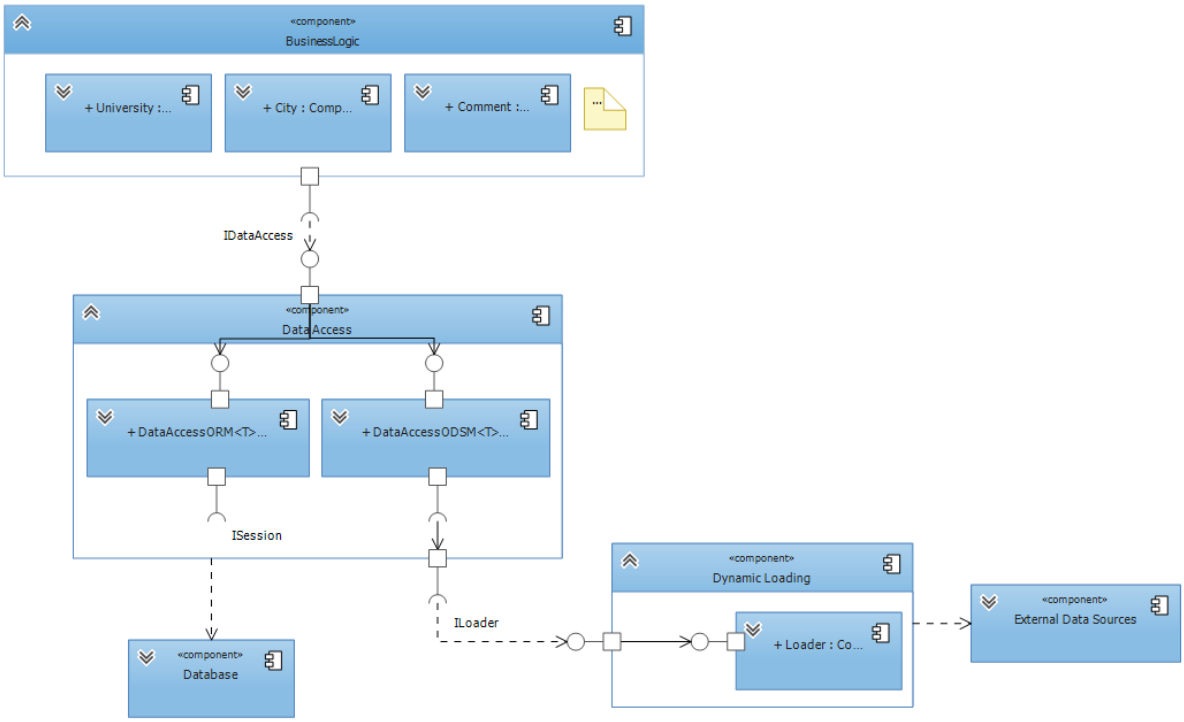

Figure 6: Component model of server side

4.2 Solution overview

Visual Studio solution is separated into several projects which tasks are to implement layers from the conceptual diagram. Each individual layer is responsible for storing or processing data in a way that provides a service to other layers. The idea of the concept in the real implementation enviroment is shown in the following picture.

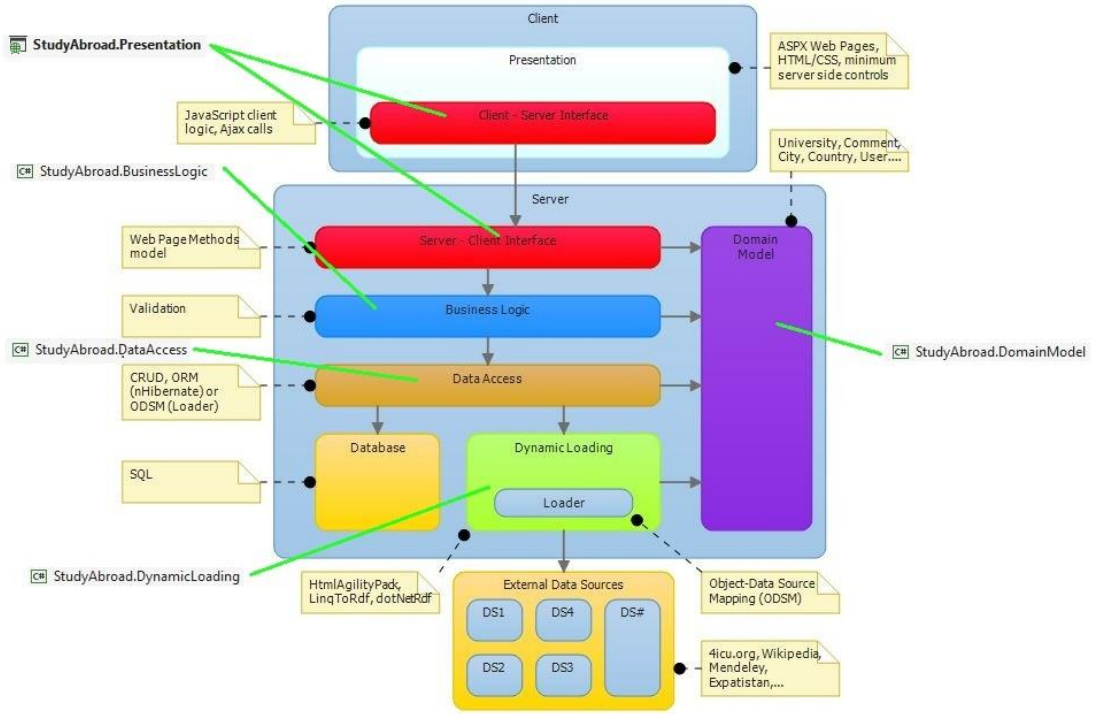

Figure 7: Mapping conceptual layers to VS solution projects

As you can see in the attached figure, layers from conceptual diagram correspond to the projects in Visual Studio. Each layer from the concept corresponds to the actual implementation in the solution.

4.2.1 StudyAbroad.Presentation

The presentation project is responsible for presentation of information that is provided by the Business Logic layer. There will be websites on the client side which will call the web methods from the interface. That websites will be made in HTML/CSS technology and the methods will be called using Ajax. Services, which will be provided by web page methods, will depend on the services of other layers in the system. There are two parts of this project. One is the HTML/CSS website presented to users with background JavaScript logic and the other one is C# code behind which provides web page methods.

4.2.2 StudyAbroad.BusinessLogic

The business logic project is responsible for the coordination between Data Access layer and Server – Client interface. It usually calls the methods from the lower layers and coordinates the data flow. This project will also perform data validation before storing it into the database.

4.2.3 StudyAbroad.DomainModel

The domain model project contains the implementation of the Domain Model layer of this project. It contains basic classes which describe the domain of the system. Details of the Domain Model will be explained in section 4.4.

4.2.4 StudyAbroad.DataAccess

The data access project represents Data Access layer from the conceptual diagram. It encapsulates the details of fetching data from the lower data layers. Thanks to it, the higher layers do not have to worry about the source from which the data is coming from. Their method call is always the same regardless of whether the data is retrieved from the database or from an external source because of the interface established here.

4.2.5 StudyAbroad.DynamicLoading

The dynamic loading project is responsible for the connection between the system and external data sources. This layer contains classes which are responsible for communication with each external data source. The details of that communication are hidden from higher layers with the interface. Other (higher) layers do not have to worry about the details of how the communication with external sources is implemented. More details about this layer will be written in section 4.3.

4.3 Data Loader

The way that the application works is by loading the data either from the database or from the external data sources and mapping them to domain model objects which are used in the application. The object-relational mapper used in the application is nHibernate and the explanation of how that framework works is beyond the scope of this document. The object-data source mapper used in the application is developed so that the data gathered from external data sources such as open data sources can be mapped to domain model objects.

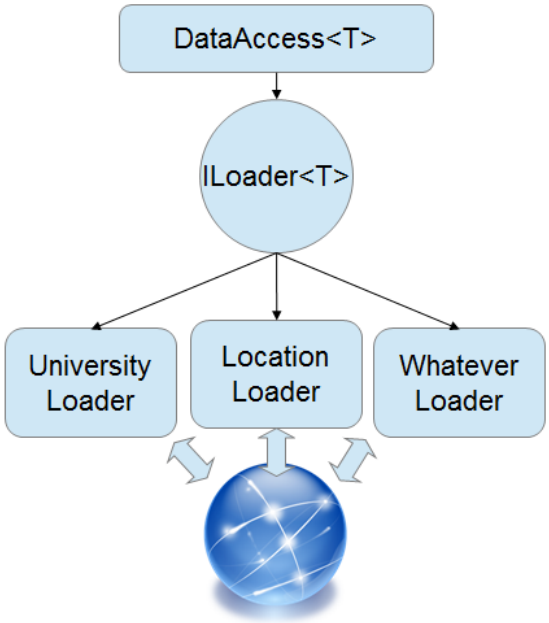

Figure 8: Loader model

The principle is simple: in order to query data the data access layer needs to be provided with a loader to fetch data into a local repository. To get only the data we need loaders have been designed to have several methods but all are accessed through a common and generic ILoader interface. For this principle to work several factories are necessary which together with configuration classes allow the programmer to create a loader which will load only the data needed and only when it is needed.

The loaders themselves are nothing more than a collection of HTML parser methods (for scraping web pages) or a collection of API access methods (for open data sources which provide an API). The loader uses methods according to the configuration settings provided and loads the local repository with data which can then be queried by the data access layer.

Loaders can be easily added once additional data sources are available and as long as they implement the common ILoader interface they will integrate seamlessly into the application.

4.4 Domain Model

The Domain Model represents a set of basic information and functionality holder classes needed to model the domain of the project. It usually contains business practices and operational details. In this project, the Domain Model classes are mostly information holders which contain collected data from the external data sources. Thus allow us to assign meaning to the collected data and classify them into appropriate groups. This way we can manage our collected data inside an object oriented environment. This is also very important in OR mapping because the classes of the domain are mapped in database tables using nHibernate. Also, the Domain Model includes the basic logic needed to provide some advanced functionalities such as the university recommendation system.

The Domain Model is a vital part of the application, so it is expected that there will be changes and upgrades in the future depending on the requirements and deadlines. Therefore, it should be noted that the attached diagram is not the final version, but is a subject of change in the future.

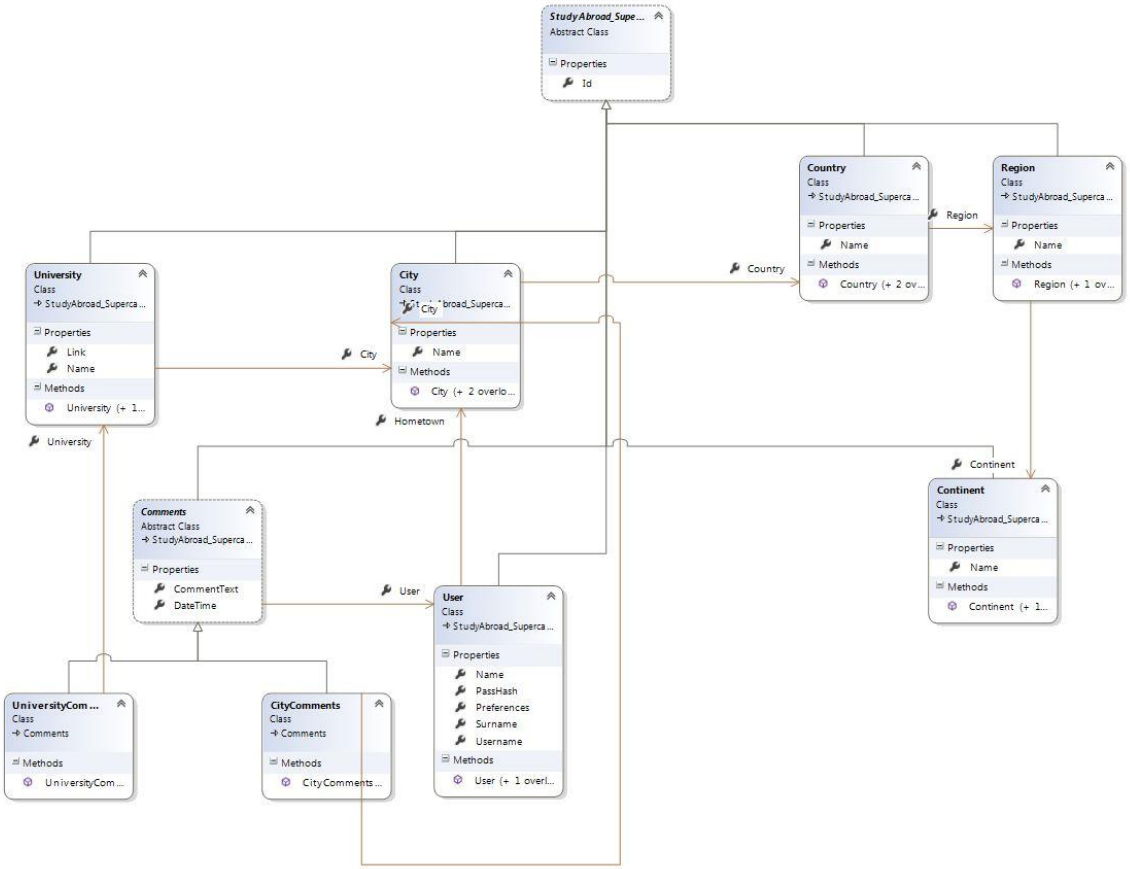

Figure 9: Domain Model class diagram

As can be seen in the diagram, there is a base super-class which is inherited by all other members of the Domain Model. It is because the class contains Id data member which is important in OR mapping using nHibernate. In this way, all the classes inherit that data member. Also, due to the mapping, data members must be public and in each class there must exist a parameter less constructor.

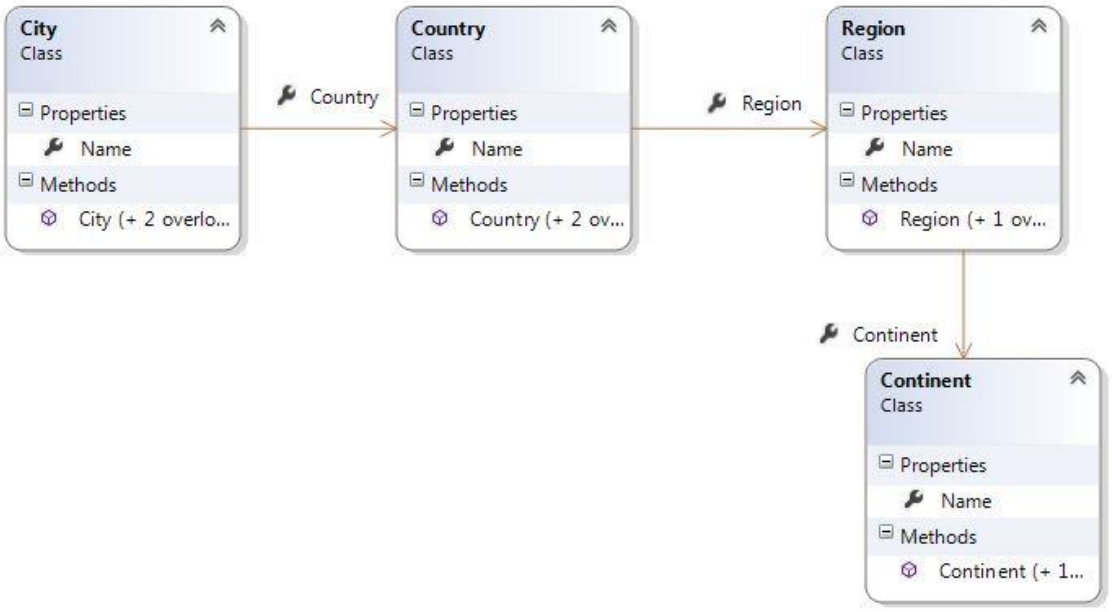

Figure 10: Geographical location classes

Regional division is presented with a series of classes that indicate the geographical location. Class City, Country, Region and Continent are representing geographic concepts and are used to precisely locate the university. These classes are also information holders for facts about the location and it is expected that they will be upgraded during development. Classes are connected with associations so every class knows to which region it belongs.

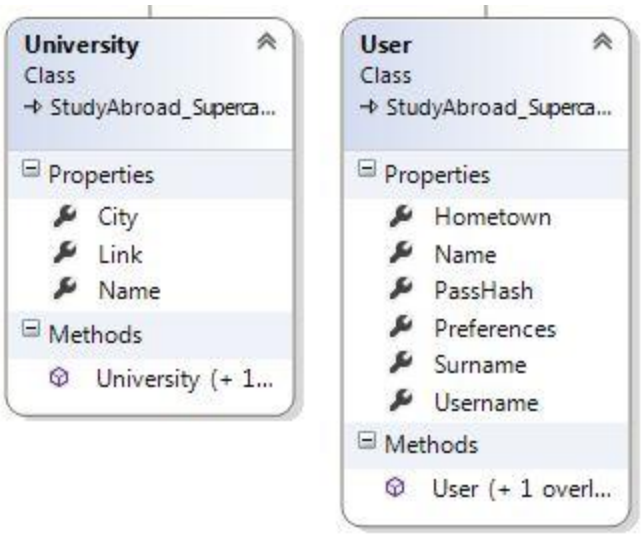

Figure 11: University and User classes

The class User stores user private data that will be used to create its profile and for his authentication. Class University stores information about every university. In the future, new data members will be added to this class if there will be more available information from data sources. Both classes have a connection to the class City. In the University class that connection represents the location of the university while in the User class it represents the hometown of that user.

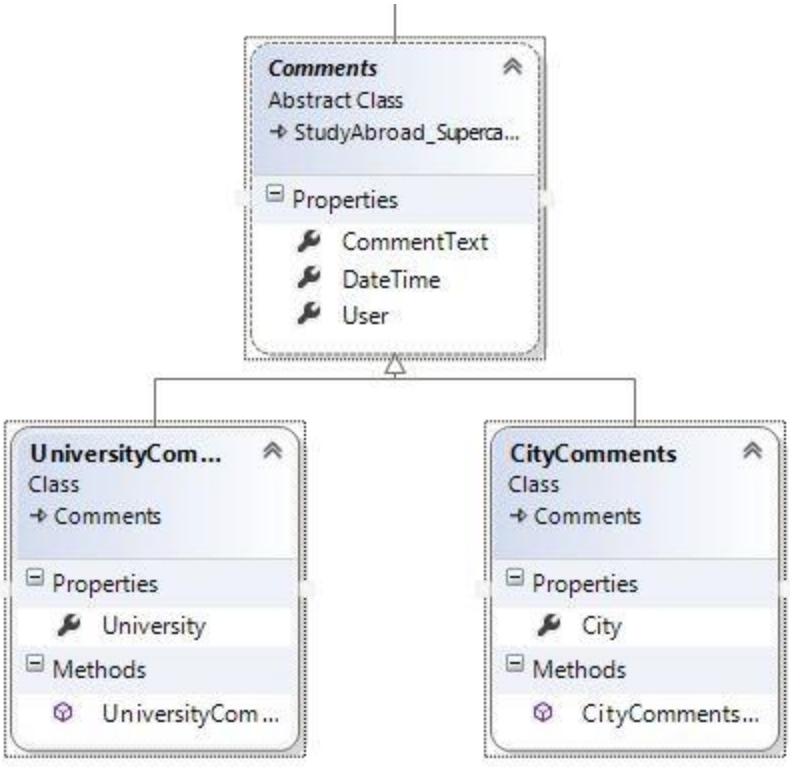

Figure 12: User comments

There are two types of comments in the system: comments of universities and comments of cities. In this hierarchy, an abstract class Comment contains information that is contained in both types of the above mentioned classes. These are: the user who commented university / city, time of commenting and comment. The derived class contains information on what is the subject of the comment (university or city).

4.5 Database definition

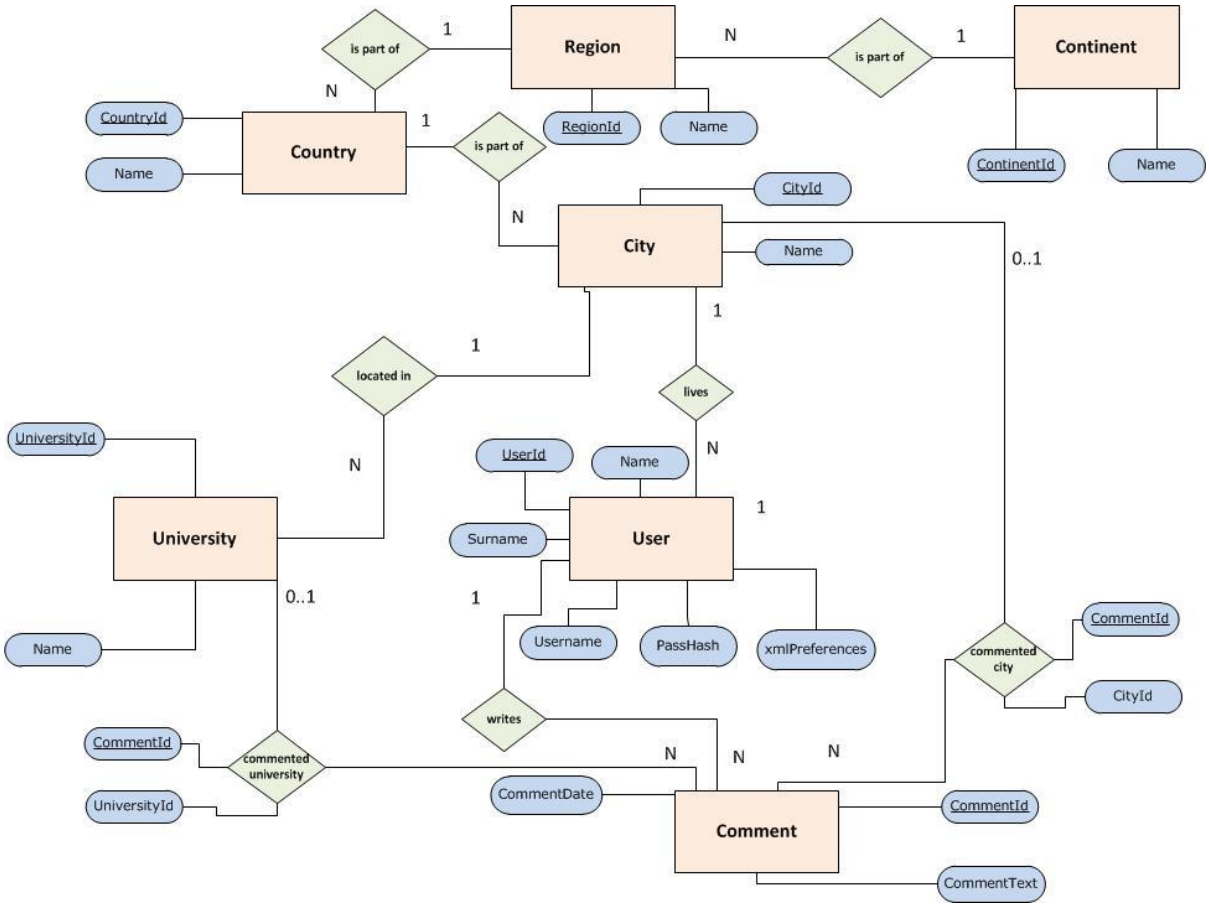

Figure 13: ER diagram

For the creation and querying of the database an OR mapping tool called nHibernate will be used together with Fluent for dynamic mapping and schema updates. nHibernate provides a framework for mapping an object-oriented domain model to a traditional relational database. Object-relational mapping (OR mapping) in computer software is a programming technique for converting data between incompatible type systems in object-oriented programming languages.

As you can see in the ER diagram, entities tables correspond to classes in the Domain Model of the system. In the Domain Model chapter it is stated that data members must be public, and that each class must have a parameter less constructor. It is very important for the realization of OR mapping.

Hierarchical model of classes is mapped in a special way. The details from the abstract class are mapped in one table and details from derived classes are mapped in separate two tables.

Preferences that the user enters into the system will not be stored in many fields but in one XML filed. Attribute xmlPreferences in the table User represents that field.

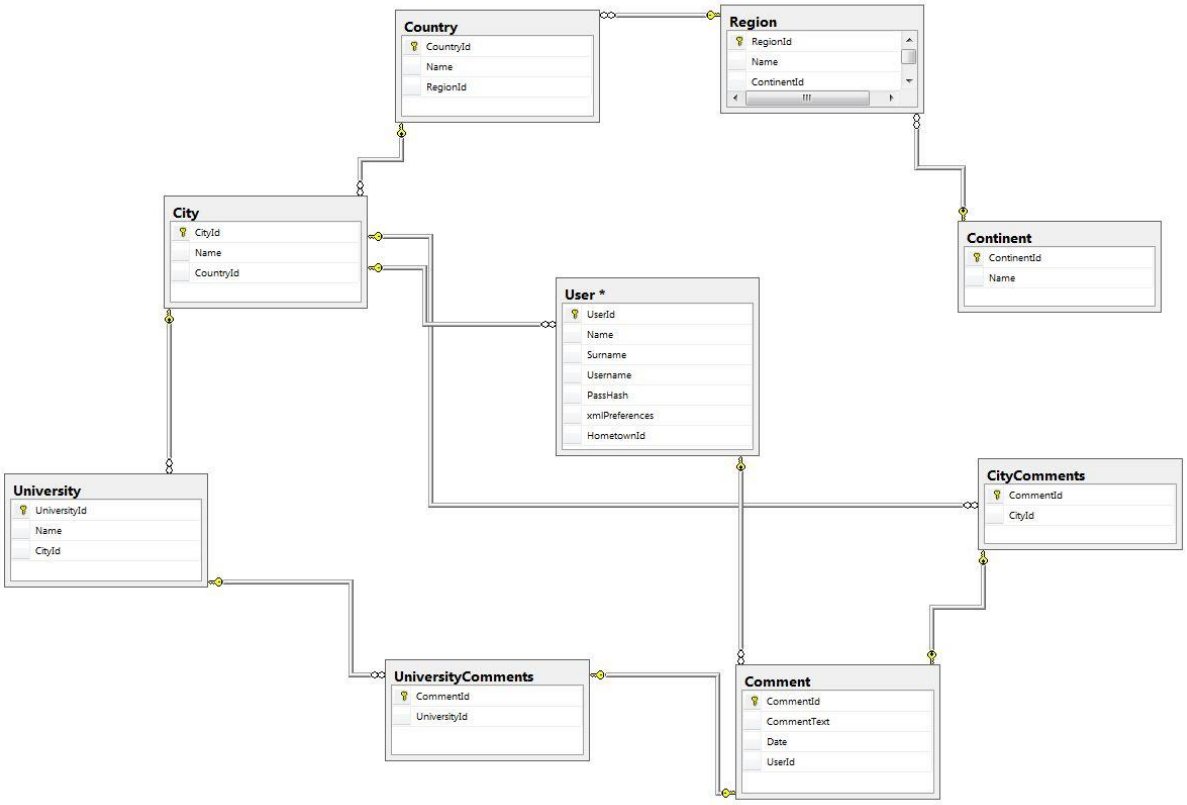

Figure 14: Database schema

In figure 13. the ER model of the database can be seen and in figure 14. the database schema is presented. The details of the database are not frozen, because some changes are expected in the future. If there is a change in the Domain Model there will also be changes in the database.

Data Access layer (StudyAbroad.DataAccess described in section 4.2.4) takes care of retrieving data in a transparent way. To higher layers it is irrelevant whether the data is obtained from the database or from external sources.