

Social Network System Design

ALBERZ AKRAWI



**KTH Computer Science
and Communication**

Master of Science Thesis
Stockholm, Sweden 2010

Social Network System Design

A L B E R Z A K R A W I

Master's Thesis in Computer Science (30 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2010
Supervisor at CSC was Dilian Gurov
Examiner was Stefan Arnborg

TRITA-CSC-E 2010:168
ISRN-KTH/CSC/E--10/168--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

Abstract

A stable and well-functional *Social Network System* that is flexible through market changes and market needs, requires a well-defined development environment infrastructure. All levels of a social network's development environment have to be analyzed to find out the best techniques and tools, for instance, programming language, application server, web server, frameworks and database solution.

Interest in social networks is growing more and faster than before, and because of that, the data volumes of such platforms increase drastically. A fundamental question is which data storage system is the most suitable one for a social network system according to data volume changes and increasing of functionalities in such platforms. To answer the question, you have to find out the best suitable data model for your application. You can then find out the best database solution for your data model based on the properties like changeability, performance, scalability, and easy-to-use. Relational databases, Cassandra as a key-value store and Neo4J as a graph database are compared to each other in this paper. A *Proof-Of-Concept* application shows which database system is the best solution for a social network system.

Design av sociala nätverk Sammanfattning

Ett fullt funktionellt och stabilt socialt nätverkssystem kräver en väldefinierad infrastruktur för utvecklingsmiljön. För att uppnå ett sådant system måste man undersöka alla aspekter som är involverade i en utvecklingsmiljö, för att sedan hitta de bästa teknikerna och verktyg som finns ute i marknaden.

Intresset för sociala nätverk växer mer och snabbare än förut och det gör att datamängden ökar drastiskt för sådana system. En viktig aspekt är att ta reda på vilken databasteknik som skulle kunna vara den bästa lösningen för ett socialt nätverkssystem som kan svara till systemets ökande datamängd och funktionalitet. För att ta reda på den bästa databastekniken måste man först ta reda på systemets datamodell och baserad på datamodellen jämföra de databaslösningar som stödjer datamodellen och därefter hitta den bästa utav dem. Jämförelsen kan baseras på egenskaper som databaslösningens förändringsbarhet, användarvänlighet, skalbarhet och prestanda. Relationsdatabaser, Cassandra som en nyckel-värde databas och Neo4J som en grafdatabas jämfördes med varandra med en prototyp i detta arbete.

Contents

- I Specification & Market Researches And Analysis 1**
- 1 Introduction 2**
 - 1.1 Method And Purpose 2
 - 1.2 Limitations 4
 - 1.3 Chapter Description 4
- 2 Project Specification and Analysis 7**
 - 2.1 What Is Value4All? 7
 - 2.2 General Problem Description 8
 - 2.3 Problems And Alternative Solutions 10
- 3 Market Researches 13**
 - 3.1 Market’s Database Solutions 13
 - 3.1.1 Results And Analysis 15
 - 3.2 The Popularity Of The Market’s Social Platforms 16
 - 3.3 Market’s Development Environment Solutions 19
- II Theory 20**
- 4 Web Development 21**
 - 4.1 Client Side Programming 21
 - 4.2 Server Side Programming 21
 - 4.3 Persistence Layer 22
 - 4.4 Security Considerations 22
 - 4.5 Value4All Three Tier Design 23
- 5 A Social Network System 25**
 - 5.1 Design Considerations 26
 - 5.2 Social Network As A Living System 27
 - 5.3 Social Network System Functionalities 30

5.3.1	Social Network System Dangers	30
-------	---	----

III Development Environment And Database Solution Proposal 32

6	Development Environment	33
6.1	Development Language	33
6.2	Integrated Development Environment	33
6.3	Build Tools	34
6.4	Versioning Tools	35
6.5	Applications Server, Web Server	36
6.6	Persistence technique	37
6.7	Java Frameworks	37
6.8	Test automation control	37
6.9	Other Techniques and Tools	37
6.10	Results	38
6.11	Download Pages	39
7	Persistence Layer - Solution Proposal	40
7.1	Introduction	40
7.2	Value4All Data Structure	40
7.3	Data Models	41
7.3.1	Flat model	41
7.3.2	Hierarchical model	42
7.3.3	Network model	42
7.3.4	Relational model	43
7.3.5	Entity-relationship model	44
7.3.6	Object model	44
7.3.7	Graph model	44
7.3.8	Key-Value model	44
7.3.9	Document model	45
7.4	Data Models Comparison Analysis	45
7.5	Data Models - Comparison Result	46
8	Why Not Relational Databases For Value4All?	48
8.1	Relational Databases - Scalability Analysis	49
8.2	Types Of NoSQL Databases	53

9	Why Choose Cassandra?	56
9.1	Introduction	56
9.2	Distributed hash table	56
9.3	Consistency Levels	57
9.4	Example of Key-Value Stores	58
9.5	Cassandra	59
9.5.1	Read And Write Operations	61
9.5.2	Partitioning	63
9.5.3	Replication And Consistency Level	63
9.5.4	Cluster Membership	64
9.5.5	Sorting	64
10	Why Choose Neo4J?	66
10.1	Introduction	66
10.2	What Is A Graph?	67
10.2.1	Traversing A Graph	68
10.3	Graph Databases - Analysis	69
10.4	Neo4J	71
10.4.1	Traversal	74
IV	Proof-Of-Concept	75
11	Proof-Of-Concept Application	76
11.1	Development Method	76
11.2	Prototype's Iterative Stages	76
11.3	Choosing Softwares and Tools	77
11.4	Softwares And Tools Installation and Configuration	77
11.5	Designing A Data Model For Value4All In Cassandra and Neo4J	77
11.5.1	The Data Model In Cassandra	79
11.6	Prototype Implementation	80
11.6.1	Transactions in Neo4J	81
11.6.2	Transactions in Cassandra	82
11.6.3	Data Model Implementation In Neo4J	83
11.6.4	Data Model Implementation In Cassandra	85
11.7	Comparison Test Cases	87
11.7.1	Changeability	87
11.7.2	Easy-To-Use	95

11.7.3 Performance	98
11.7.4 Scalability	103
11.7.5 Conclusion	105
V Future Of Value4All & Final Words	107
12 Which Social Networks Can be Integrated In Value4All?	108
12.1 Facebook	108
12.2 Twitter	109
12.3 LinkedIn	109
12.4 Result	110
12.5 OAuth Protocol	110
13 The Future Of Social Network Systems	113
14 Final Words	115
VI References & Appendices	117
References	118
Appendices	
A Graph Databases - A Complete Comparison	122
B Social Network Creators - Comparison	125
C NoSQL Databases - Comparison	127
D The Popularity Of The Market's Social Platforms	130
E Market's Development Environment Solutions	131
F Application Installations	132
F.1 What we'll need?	132
F.2 Applications And Their Versions	133
F.3 IntelliJ IDEA Installation	133
F.4 Git Installation	134
F.4.1 Git Installation in IntelliJ IDEA	136
F.5 Maven and Junit Installation	136

F.6	Subversion Installation	138
F.7	Selenium Installation	141
F.8	Jetty Installation	142
F.9	Neo4J Installation	143
	F.9.1 Testing Neo4J Installation By An Example	145
	F.9.2 Indexing In Neo4J	146
F.10	Cassandra And Cassandra Client Installations	147
	F.10.1 Cassandra Installation	148
	F.10.2 Cassandra-Thrift Installation	149
	F.10.3 Cassandra-Hector Installation	155
F.11	Master Thesis Hardware Environment	157

Part I

Specification & Market Researches And Analysis

Chapter 1

Introduction

My name is Alberz Akrawi, and I am a student of School of Computer Science and Communication at KTH Royal Institute of Technology. From the beginning, my thoughts were to take an extensive master thesis within an interesting area of technique in expansion and because of that, I took this master thesis that is about design of a *Social Network System*. Today, there are many social network systems that are growing fast, which causes some difficulties for them to manage the huge amount of data. It is also critical for them to choose the right developing environment to maximize the durability and minimize the risks.

Today, there is no social network platform for:

- **Entrepreneurs** with no experiences to write a business plan or run a company.
- **Investors** who looking for other profitable investments.
- **Service Providers** that looking for new customers offering their service packages.

Value4All has planned to be a such durable social network system dedicated to entrepreneurs, investors and service providers. Value4All has also planned to be integrated with already stabilized popular social network systems in Sweden, for instance, LinkedIn, Twitter or Facebook to create a more valuable environment for their members.

1.1 Method And Purpose

There were several comparison studies during the master thesis:

1. Find out the best data models suitable for Value4All based on the structure of Value4All's data.

2. Find out the best database solutions for the results from the first comparison.
3. Compare the databases from the second comparison.

The second and the third comparisons were based on properties:

- *Easy-To-Use*
- *Changeability*
- *Performance*
- *Scalability*

These comparison studies took also advantages of the market's available social network systems to find out how they had solved their database needs and which database models are used by them. For more information about that, see the chapter 3.1 (Market's Database Solutions). Comparison analysis was based on the results from different test-cases from a "*Proof-Of-Concept*" application. During the whole work, I made notes from analysis of different test-data.

A very important comparison property between the database solutions was how easy or difficult it was to migrate from a presentation of one database model to another changed one. To understand the *changeability* power of databases, I presented a model to show how much work there were needed to migrate from one version data model implementation to another changed model with a description on what we needed to do manually by a script or something else. This fundamental property was very important for Value4All to become a durable system through market changes and market needs. This is also an important property that most companies take it into consideration when they design their applications. On the other hand, there are many systems that are implemented without thinking about this property and because of that they maybe need to change the whole system to overcome the complexity problems which is a difficult task and require much money.

The points that are mentioned below are also included in this paper.

- A recommendation on which developing environment are the best choice for Value4All.
- A description on what a three tier architecture is and how Value4All can be developed as a such system.
- A description on what a social network system is.

- Introducing some popular social network platforms in Sweden and how they can be integrated with Value4All.
- A description of the prototype, installation and configuration guide.
- A description about the future of social networks.

The purpose of the master thesis is to propose a durable, well-defined solution with high market acceptance to the entire development environment of Value4All, including a scalable database solution to be used as a basic data for decision-making.

1.2 Limitations

The paper doesn't contain detailed information about every technical term and all techniques used during the master thesis. All data models and all database solutions were not described in this paper in details. Only some database solutions were compared to each other to find out the best database solution for Value4All. A "Proof-Of-concept" application was developed for comparing only two database solutions with different comparison properties like easy-to-use, scalability, changeability and performance without thinking about other design aspects like interface design or security measurements.

1.3 Chapter Description

Here is a summary about the contents of all chapters in this paper.

PART I - Specification & Market Researches And Analysis

- Chapter 1. This chapter contains a brief introduction of what the master thesis is about, background, method and purpose of the master thesis.
- Chapter 2. This chapter contains the specification of the master thesis, a general solution proposal and the problems that I faced during the master thesis.
- Chapter 3. This chapter is about some market researches that were done during the master thesis. The researches are about the market's most used database solutions and their popularity and also market's development environment solutions.

PART II - Theory

- Chapter 4. Value4All is a web application with a three tier architecture. This chapter contains a brief introduction about what web development is and how Value4All can be developed as a three tier architecture.
- Chapter 5. This chapter contains a brief introduction about what a social network system like Value4All is.

PART III - Development Environment And Database Solution Proposal

- Chapter 6. A solution proposal for the entire development environment of Value4All.
- Chapter 7. A solution proposal for Value4All's database solution.
- Chapter 8. A description about why relational databases are not enough for Value4All.
- Chapter 9. A description about what Key-Value stores are, what are the market's most popular key-Value store and why we choose Cassandra?
- Chapter 10. This chapter contains an introduction about what graph databases are, what are the market's most popular graph databases and why we choose Neo4J?

PART IV - Proof-Of-Concept

- Chapter 11. This chapter is about the implementation of the prototype (Proof-Of-Concept application), result, analysis and conclusion of Neo4J and Cassandra comparison test-cases.

PART V - Future Of Value4All & Final Words

- Chapter 12. This chapter is about which popular social network systems can be integrated with Value4All.
- Chapter 13. A brief description about the future of social networks.

- Chapter 14. This chapter contains final words of the master thesis.

At the end of this paper, there is a presentation of the references to information resources and appendices.

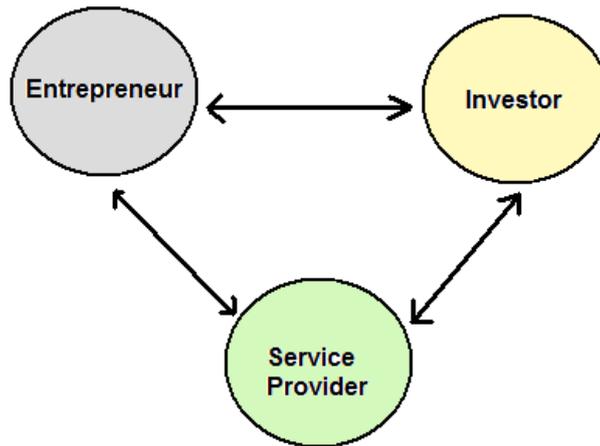
Chapter 2

Project Specification and Analysis

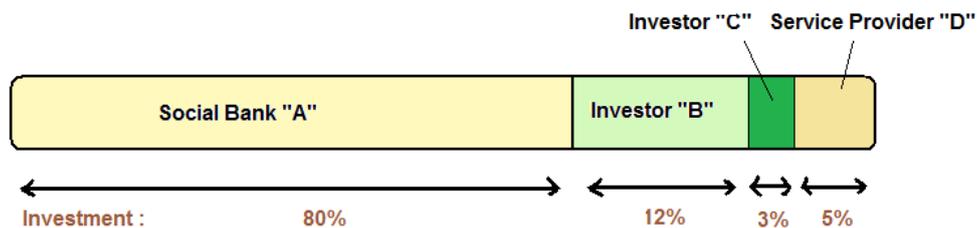
2.1 What Is Value4All?

The first version of Value4All contains three groups of people. That was important to take this into account, to make the first prototype interested to all of these three groups of people, which means that, the application had to contain some functionality to all of these three groups.

- **Entrepreneurs.** Entrepreneurs are the people with an idea who look for investors and a place to learn how to start and realize their idea (a lifestyle idea or an academic startup idea) to a real business. Value4All helps them to create a business plan from the business idea and find capital to their business.
- **Investors.** They want to see how an entrepreneur has thought about the incomes and if the idea is profitable. The other important aspect for them is to know if the idea was scalable (the ability to grow).
- **Service Providers.** In Value4All, service providers can help other people with different services. For example, they can offer an entrepreneur with the creation of a home page, a company logotype, a company name.



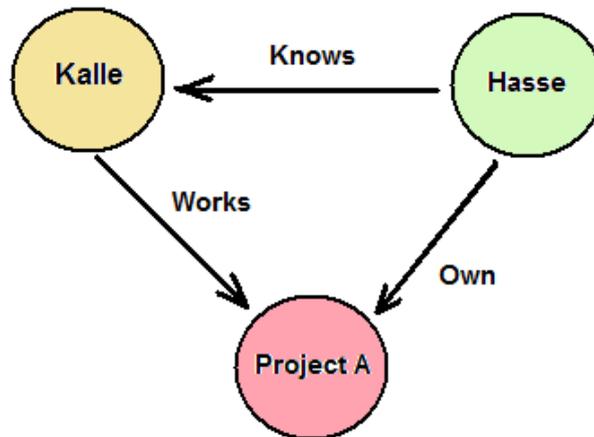
The picture above, shows the dependencies between these groups of people based on their interest in different projects in the first version of Value4All. When these groups of people agreed on the idea then they can agree on the profit percentage point between each other. Value4All is also connected to social banks to help entrepreneurs with some investment opportunities like non-risk capital investments.



2.2 General Problem Description

To make it possible for Value4All to success in the market and become a popular, well-defined social network system, there was important to look for solutions used in the market by other social networks for the entire development environment. For instance, which programming language was the best choice (Java, .Net, Scala, Ruby, and, etc.) or maybe a mixed of programming languages could be a better solution.

An easy example that shows how Value4All's data model can look like is as follows.



"Kalle", "Hasse" and "Project A" are the nodes of the graph. "Knows", "Works" and "Owns" are the edges of the graph which indicate the relationships between the nodes. Of course, this data structure can be designed by other data models, such as a relational model. For example:

Name	Knows
Hasse	Kalle

Name	Works
Kalle	Project A

Name	Owns
Hasse	Project A

Person - name
Hasse
Kalle

Project - name
Project A

Three tables are created for designing three relations (Knows, Works and Owns) for two persons ("Kalle" and "Hasse") and a project ("Project A"). A table for persons and a table for the project.

The question was which data models were the best choices for Value4All's data. In this paper, there is comparison analysis by a prototype to find out a good database solution for Value4All.

2.3 Problems And Alternative Solutions

During the master thesis, I came across some problems. Some of these problems are described here:

1. There is some social network creator in the market. The question was if we could use a such system for development of Value4All instead of developing it from scratch?
2. How to find good material about NoSql databases?
3. How to decide the development environment without any inputs from the companies that were asked by a questionnaire?
4. Writing the chapter 13 (The Future Of Social Network System) without any inputs from the companies that were asked by a questionnaire?

Problem Solution:

1. I found some open source social network creator in the market that are developed for a different purpose. I couldn't find any information about these application's performance and scalability strategies and also what will happen in the case who, the amount of data increases drastically and how difficult it was to migrate from one database solution to another one. Almost all of these applications are based on relational databases and because of that, there will be a limitation for the growth of the Value4All since relational databases are not good enough to be used as a database solution in social network systems. For more information, see the chapter 8.1 (Relational Databases - Scalability Problem). Another problem with these applications is that there are a limitation on how much influence you can have on the interface of your application. A complete comparison between these social network creators is available in the appendix B (Social Network Creator - Comparison).

There is some free and open source software for creation of social networks on the internet with different functionalities.

- Elgg. Elgg won the best open source social network creator for the year 2008, and is a powerful free software that offers a blog, networking, community, collection of news using feeds aggregation and file sharing. Elgg works on a LAMP (Linux, Apache, MySql and Php) environment.

Download page: <http://www.elgg.org/>

- Mahara. The purpose of Mahara is to allow its user to view their life-long learning, skills and development to their friends. Some functionalities in Mahara include a blog, file managers, and, maintaining a list of friends. Mahara runs on LAMP (Postgres is preferred over MySQL).

Download page: <http://mahara.org/>

- Lovd By Less. Lovd By Less is built on Ruby on Rails. Some functionalities of Lovd By Less include a blog, photo gallery, searching for friends, user-to-user messaging with Flickr and YouTube integrated on it.

Download page: <http://lovdbyless.com/>

- Xoops with Yogurt extension. XOOPS is an extensible Content Management System (CMS) that allows you to build pages based on your needs. You can first build a blog and then include social activities, forum and, etc. Yogurt is the module that allows to build a social site with XOOPS with functionalities like a personal album of pictures, videos from YouTube, mp3 files, and a list of friends. XOOPS is based on php in a LAMP environment.

Download page: <http://www.xoops.org/>

- AROUNDMe. AROUNDMe is different from the other social network creators. Instead of building a social network platform, it allows you to set up your servers who enable your users to create their own social network page, community or webspace like Google Groups, Snappville, CollectiveX. It allows your users to have access to a guest book, a blog, a forum and, etc. PHP is used as the programming language.

Download page: <http://www.barnraiser.org/aroundme>

AROUNDMe was not what we were looking for. The question was if any of the other social network creators that were mentioned above were suitable for Value4All to become build upon it, instead of developing it from scratch. As mentioned above, the best idea is to build Value4All from scratch because of the inflexibility of the changeability property, unscalability and limited performance of these network system creators when the amount of data increases.

2. There is much different literature about relational databases, but I couldn't find any good literature about NoSQL databases. My first information source for this part was the internet because NoSQL databases are relatively new.

3. From the beginning, I thought that I could use a questionnaire to ask some companies about which applications they are using. In that situation, I could see which applications were popular and were most-used in the market, but I got almost no answer from the companies that were asked by a questionnaire. It was maybe, the company secrets that, they didn't want to talk about. I solved the problem by doing some researches on the internet and ask Crisp about it.

4. That was hard to write this chapter because I hoped to get some answers from companies that were asked by a questionnaire. I made some research on the internet and also asked some private persons by a questionnaire.

Chapter 3

Market Researches

3.1 Market's Database Solutions

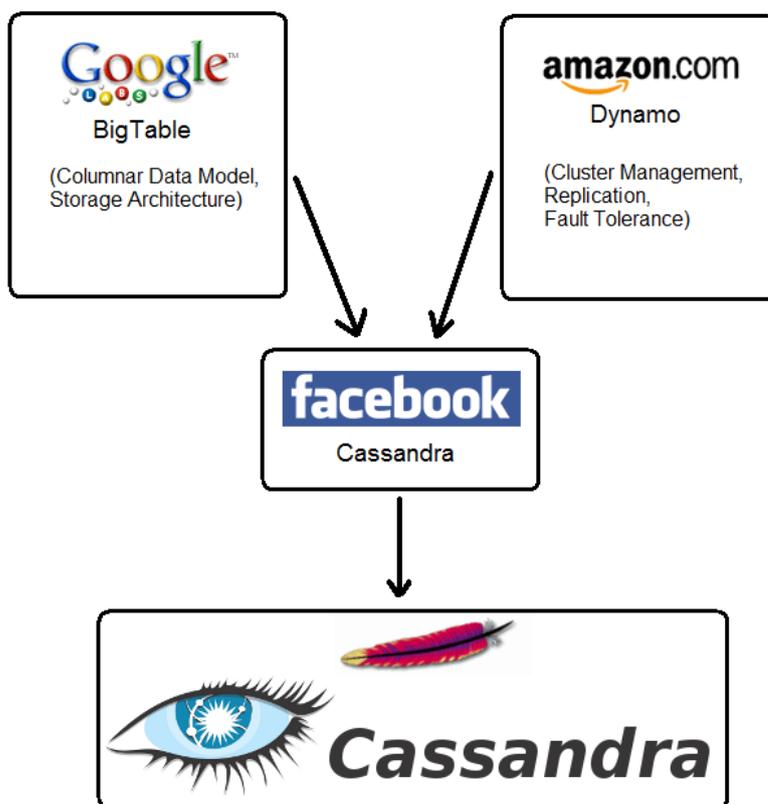
When developing a system, one of the most important cases we analyze is if the system scale. Scalability means the ability of an application to handle increasing load by, for instance, adding additional nodes (*horizontal scaling*) or upgrading existing machines (*vertical scaling*). Horizontal scaling becomes more important with Web 2.0 (distributed systems, cloud computing, social network systems, and, etc.). There are many companies that solved their scalability problems and many other companies tries to find a good solution to their application infrastructures. Here is the result of some researches about market's database solutions for the scalability problem.

- *Amazon*, (www.amazon.com/). Amazon is the world's first e-commerce site in the internet, and they have solved the horizontal scalability with their own solution by a distributed key-value store (Dynamo).
- *Twitter*, (www.twitter.com/). Twitter is one of the biggest social network systems today, and it has been considered that Twitter has a problem with slow response time and downtime. Twitter has switched over to a distributed key-value store (Cassandra, for storing all tweets) because it can be run on large server clusters and is capable of handling huge amounts of data at a time.
- *LinkedIn*, (www.linkedin.com/). LinkedIn is a popular network system in Sweden that uses a distributed key-value store (Voldemort) to their high-scalability storage problems.
- *Google*, (www.google.com). Google has solved the problem of horizontal scaling in their infrastructure, with an own solution. They have solved the problem by a distributed key-value store (Google BigTable).

- *Facebook*, (www.facebook.com/). Facebook is another popular social network system in the market and is also another example of what happening if you don't plan your infrastructure orderly from the beginning. They have big problems with their storing infrastructure, which don't scale as it required. Facebook has huge clusters of MySQL and Memcached servers to deal with storing user data and serving user queries to overcome the scaling problem. As it is reported nearly a year ago, they already had close to 2,000 MySQL servers and 1,000 Memcached servers in addition to their 10,000 web servers to be able to handle their enormous data. They have looked at other NoSQL solutions to solve their problems, and they have implemented Cassandra from the scratch as a distributed key-value store.
- *Cisco's WebEx*, (<http://www.mahalo.com/>). Cisco's WebEx providing online meeting and services for business worldwide, including web and video conferencing. They use Cassandra to store user feeds and activities in real time.
- *Ooyala*, (<http://www.ooyala.com/>). Ooyala is a leading provider of end-to-end video platform applications. They have moved to Cassandra as its new Database system. That will enhance Ooyala's current analytic offering and create the backbone for its new Business Intelligence platform for video.
- *Rackspace*, (www.rackspace.com/). Rackspace also uses Cassandra in their Cloud Computing and Hosting.
- *IBM*, (www.ibm.com). IBM has done researches using Cassandra building a scalable email system.
- *Digg*, (www.digg.com/). Digg, on March 8, 2010, uses Cassandra as their database solution.
- *Reddit*, (www.reddit.com/). Reddit uses Cassandra as their database solution from memcacheDB on March 12, 2010.
- *Cloudkick*, (https://www.cloudkick.com/blog/2010/mar/02/4_months_with_cassandra/). Cloudkick's primary use of Cassandra is for storing monitoring data of different metrics, checked by their system.

3.1.1 Results And Analysis

There are some important properties when designing an application, such as, performance and scalability. SQL databases like, MySQL or Oracle database is widely-used in the market because of performance, security, simplicity and flexibility but scalability is another important property that SQL database can't support in an easy way. Scaling is available by special solutions in SQL databases. Scaling is very important in social network system but scaling solutions in SQL databases are very expensive and complex. As you can see almost all popular companies with huge amounts of data looking for other solutions than NoSQL databases.



Cassandra is one of the best and popular NoSQL databases used in the market because, the database brings together the scalability and reliability of both Amazon's Dynamo and Google's Bigtable storage systems in a single database model. There is a new type of NoSQL databases, Graph Databases, like Neo4J, which are worth looking at in this paper. With other words, web application nowadays, have different needs than applications that, SQL databases are designed for, such as:

- Low Response Time. The response time to the visitors has to be very low. This is one of the most important properties of a social network system. The visitors want to see the information really fast otherwise you risk that they don't come

back. The response time of SQL databases increases when the amount of data and tables increase in the database because of many joins and unions that are very costly.

- Scalability. It is very important that a social network system scale well because of the huge amount of data that increase drastically. This solution must be easy, fast and also at a low cost. The available scalability solutions of relational databases are not good enough because of the complexity with special solutions.
- High availability. This is the trade-off between ACID property of relational databases and the high availability of NoSQL databases in case of not providing *high consistency* (means that you can't find any information in a database that is against each other). Generally, web applications can function without high consistency.
- Schema-less databases (NoSQL databases) provide the flexibility to take the real world's data as they are into the databases. In large scale application, it is difficult to achieve the schema flexibility in SQL databases because of a high number of tables (causing many joins and unions with no good performance). Also changeability of schema-based databases based on a big amount of tables is not easy.
- Geographical distributions are also another good property of NoSQL databases in that way that, you can place your databases in different data centers.

Social network systems are not as a banking system, and we don't actually need the ACID property of relational databases in such applications, instead we need internal scaling of data size, high read and write operation rates and frequent schema changes. In these paper, there is a comparison analysis about why relational databases are not good enough to be used as a database solution in development of social networks. For more information see the chapter 8.1 (Relational Databases - Scalability Problem).

3.2 The Popularity Of The Market's Social Platforms

According to researches that were done by a questionnaire, the most popular social networks in the market in Sweden are listed below. You can also see, how popular it is to use social platforms among people.

- Facebook. Facebook connects people with friends and live around them.

<http://www.facebook.com/>

- Twitter. With Twitter, you can share what's happening right now.

<http://twitter.com/>

- LinkedIn. Here you can share your academical studies, CV and other skills.

<http://www.linkedin.com/>

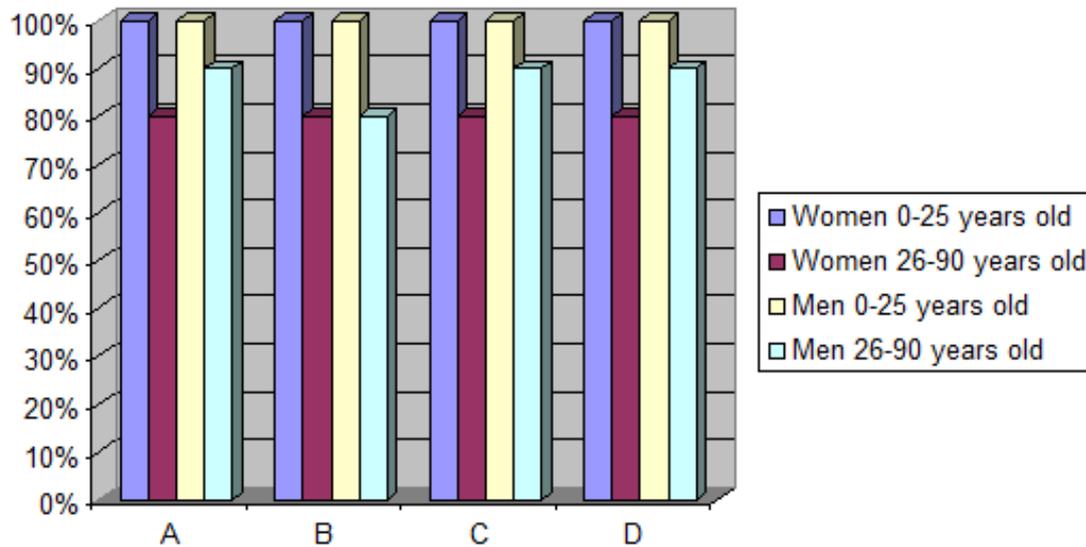
A questionnaire was sent to more than 100 private persons, and I could use 46 answers, of which 26 was women and 20 was men. 10 of men and 10 of women who answered were over 25 years old.



You can find the questionnaire in appendix D (The Popularity Of The Market's Social Platforms).

Group	A Do you know what a social network (Facebook, Twitter, LinkedIn) is?	B Are you using Facebook, Twitter, LinkedIn and etc.?	C Is it interesting for you?	D Are the services user friendly?
Women 0-25 years old	100%	100%	100%	100%
Women 26-90 years old	80%	80%	80%	80%
Men 0-25 years old	100%	100%	100%	100%
Men 26-90 years old	90%	80%	90%	90%

In the table above and the diagram below, there is a number in a percentage point for all of them who answered within each group of questions (A-D). For instance, 100% of women 0-25 know what a social network system (Facebook, Twitter, LinkedIn or something like that) is.



In appendix D (The Popularity Of The Market's Social Platforms):

- Questions 1 fits in "Do you know what a social network (Facebook, Twitter, linkedIn) is?".
- Questions 2 and 3 fit in "Are you using Facebook, Twitter or LinkedIn?".
- Question 4 and 5 fit in "Is it interesting for you?".
- Question 6, 7 and 8 fit in "Are the services user friendly?".
- Question 9 was about the future of social network systems. I used the result of this question in chapter 13 (The Future Of Social Network System).

As the result shows, almost all people know what a social network is. This result shows the popularity of social network platforms like Facebook, Twitter and LinkedIn. I think the popularity and the usage of such systems will increase dramatically in the future and the possibility to get Value4All integrated with these systems will improve the popularity of Value4All in the future. Almost all of them who answered, were very interested on what will happen in the future and what improvements they are going to see in the future on these social network systems.

I can mention that, some older people that knew what a social network system was but refused to participate and answer the questions, were worried about their children who use such systems and what will happen in the future.

3.3 Market's Development Environment Solutions

A questionnaire was sent to some big and popular companies, both social networking companies and other big companies with a three tier architecture solution. Unfortunately, I am not going to show any results here since only one company answered to the questionnaire. The questionnaire is available in the appendix E (Market's Development Environment Solutions).

Part II
Theory

Chapter 4

Web Development

Web development is a broad term in development of web sites for the Internet or the intranet that includes, web design, web content development, client/server scripting, web server, application server, network security, etc. Web development uses different techniques like Java Enterprise Edition (JEE) and Microsoft .NET to create interactive web pages and consists of many areas and some of them are listed below.

- Client Side Programming.
- Server Side Programming.
- Persistence Layer.
- (Security Considerations for all levels).

Such architecture is called a *three-tier architecture*.

4.1 Client Side Programming

Client side programming refers to techniques and programs that can be used in a web application to be viewed by users' web browsers. These techniques and programs are an important part of an interactive HTML, for instance Javascript and Ajax. See the chapter 6.9 (Other Techniques And Tools) for more information.

4.2 Server Side Programming

Server side programming refers to actions that are performed by a server who is called from client side programs. Client requests are performed by the server because they require access to information from databases that is not available by clients directly. Some of the server side techniques are listed below.

- ASP. Active Server Pages, is Microsoft's server side script programming for dynamic web pages. ASP is included as a component of Windows Server 2000 and higher server versions and is usually called as ASP.NET.
- Java EE. Java Enterprise Edition, is used for server programming in the Java programming language. J2EE includes libraries, that provide functionalities to develop secure, reliable and multi-layer Java applications.
- PHP. Hypertext Preprocessor is a server side programming language that is designed to develop dynamic web pages. PHP code is embedded into the HTML documents and interpreted by a web server with a PHP interpreter.
- Perl. Perl is a Unix server side programming language that is designed to develop dynamic web pages.
- Websphere. IBM WebSphere is a set of software that is designed to manage enterprise applications across multiple platforms, using Java web technologies.
- Microsoft .NET Framework is a framework for Microsoft Windows operating systems that contains a set of library solutions and a virtual machine that manages the execution of applications.

4.3 Persistence Layer

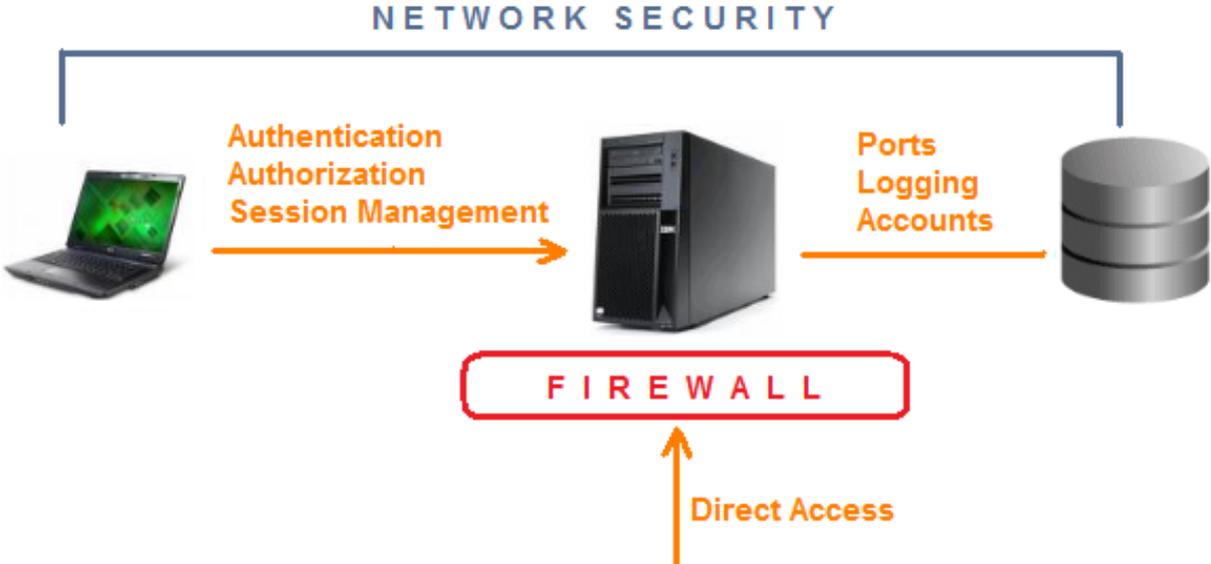
Persistence software is an application designed to store, organize and manage information, for instance, MySQL, Oracl DB, Neo4J and Microsoft SQL server.

4.4 Security Considerations

There are many security considerations when designing web applications because it is critical to protect the organizational data by controlling the information into the company databases and out from the databases. For instance, data entry error checking through forms applied by a user, filtering output to protect critical data, and encryption of data. It is important to test web applications before the applications are public released in a production environment.

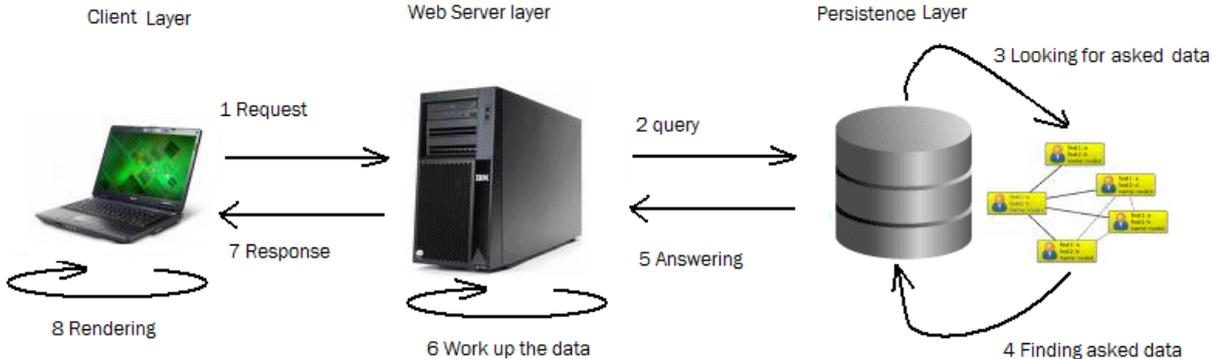
Web servers and databases have to be kept safe for threats outside when transmitting data. Many technologies exist to keep the information on the internet safe when it is

transmitted from one location to another, for instance, Secure Socket Layer Encryption (SSL) or Certificates that are issued by certificate authorities to help prevent internet fraud. A weakness in any layer of a web application makes the application vulnerable to attack.



4.5 Value4All Three Tier Design

The information flow of Value4All as a three tier architecture can be illustrated in the picture below.



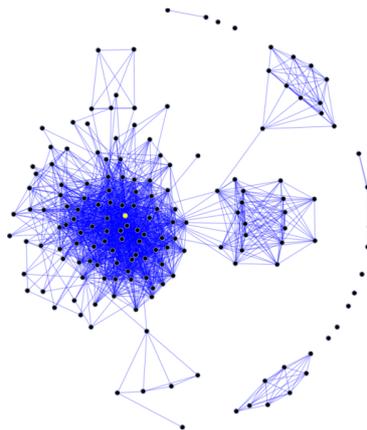
- 1 Request. A Value4All user, sends a request (for instance, find a project) by its web browser to Value4All Web server (for instance, Jetty Web server).

- 2 Query. Value4All Web server sends a query to Value4All database to find the requested project.
- 3 Looking for asked data. Neo4J looks for the project.
- 4 Finding asked data. Neo4J finds the project.
- 5 Answering. Neo4J sends the project to the Value4All Web server.
- 6 Work up the data. Value4All Web server, if needed, manipulates the answer.
- 7 Response. Value4All Web server sends the project to the Value4All user's web browser.
- 8 Rendering. User's web browser renders the result and shows it.

Chapter 5

A Social Network System

There are many social network platforms in the market that are developed for different purposes, for instance, Facebook, Twitter or LinkedIn. In such platforms, people can join, write to each other, create different groups of people. We can say that social network systems in terms of network theory consist of nodes (for instance, an individual or a group) and connections between them (for instance, an individual knows another one or an individual is a member of a group, financial exchange, friendship, knowledge or religion). In a typical scenario, you register on a web page and send invitations to your friends to become part of your network or community. Your friends, in turn, invite their friends and so on. Only after joining the network, someone can see your information.



According to Nielsen O. (Online Measurement Services, <http://en-us.nielsen.com/>), social networks are the fourth most popular kinds of online activities with 67% of the world *online* population. Different purposes of creating social networks can be, for instance:

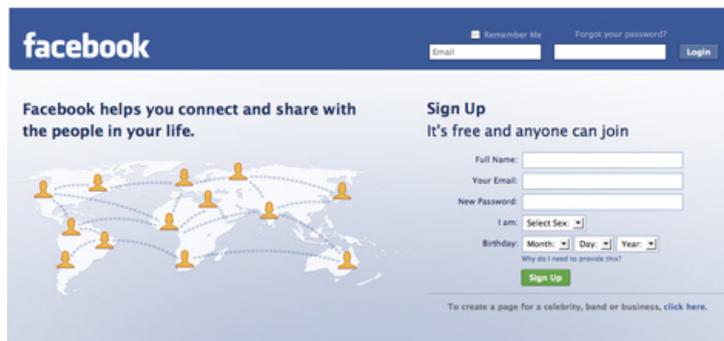
- Personalization or friendship approaches like Facebook, Twitter and MySpace.

- Professional approach likes LinkedIn. Value4All will fit in under this category.

5.1 Design Considerations

Designing social network systems are complex and need many researches. One important thing, when designing such systems are to consider that, all functionalities have to be easy to use by users. There are many aspects to think about when designing social networks and some of the most important aspects are listed below.

- Clear information for visitors. When people visit a social network site, they have to know all about the purpose of the site within seconds because most people don't have time to find out what a site is about, if they can't see that from the beginning. The site's title, graphics, logotype and other elements of the site have to be obvious and self-speaker for visitors.



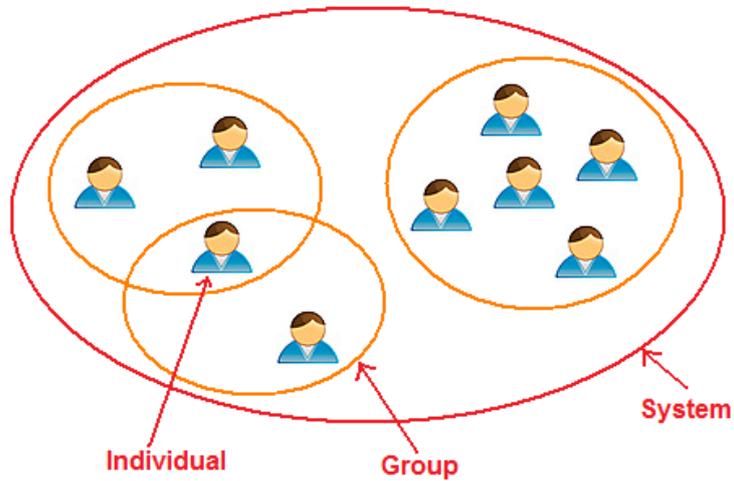
- Clear information for users. That is important to have clear and updated information for users of the site to make the trust between the site and users stronger and make them more interested in the site.
- Users and visitors should be able to do some options. A social network site should give both visitors and the site's users something professional to do. For example, logging in for users and signing up for visitors. The site should give visitors other options too, for instance, gives them the opportunity to know what the site is about quickly or describes the requirements before they sign up, let them search for people they already know on the site and give them the opportunity to see why they should sign up in the site before they sign up, because it leads to trust between the site and visitors.

- Provide users with interesting information of their friends or groups. It is important that users can see what their friends are doing or what is new within their groups from the moment they log in to the system.
- Make it easy to find friends. The users of a social network site should be able to find friends easily. The site should let users to search by different search options like an email, school, company, name, age and other options.
- Configurable Profile pages by users. A social network system needs security considerations but on the other hand, the site should let users at least do some ability to make their site reflect their personality. For example, through color schema, backgrounds, adding information about themselves, comment on their activities.
- Regular information changing. The characteristic of a social network design is that, the user's information can change constantly. Updating individual contents are important because this keeps users coming back, because there is always something more and new to do.
- Let users to create own groups. A successful social network lets users to create different groups to organize their friends, such as business contacts, friends, and family. The site should let users to add their contacts to more than one group simultaneously.
- Search functionality. A social network site should let users to find information they are looking for.

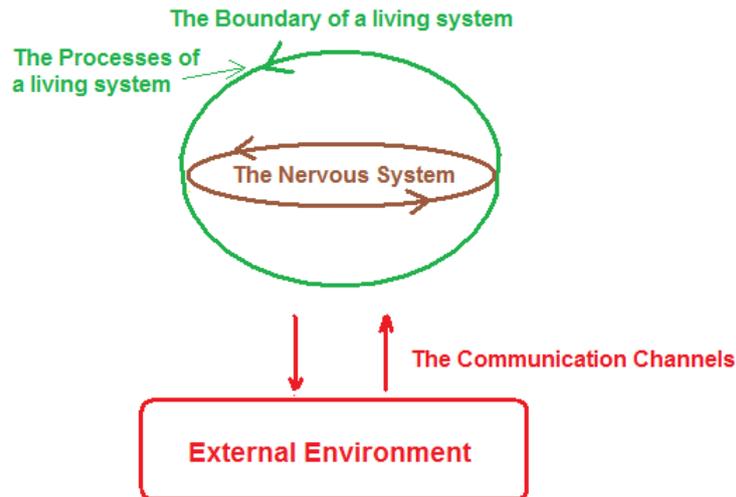
5.2 Social Network As A Living System

Ken Thompson (a leading expert in the area of virtual enterprise networks http://en.wikipedia.org/wiki/Ken_Thompson), suggests that social network developer should look at a social network system as a living system. He states that, "it is possible to successfully apply living systems design to social systems but only if, living systems theory is applied to each one of the three nested systems making up a living unit: the individual, the group, the system itself".

A Social Network System As A Living System



“Living system theory is about the existence of all living systems, their structure, their behavior and development”. This theory is created by James Grier Miller (an American biologist, who originated the modern use of the term "behavioral science", and originated the living system theory, http://en.wikipedia.org/wiki/James_Grier_Miller). Another theory are defined by Maturana and Varela (Principles of Biological Autonomy, http://en.wikipedia.org/wiki/Humberto_Maturana, http://en.wikipedia.org/wiki/Francisco_Varela) that, have defined living systems in a more philosophically way: “A living system is one whose only products are itself”. They suggested also that there are four key-aspects of a living system that are viewed in the picture below.



- The Boundary: represents the boundary of a living system. This boundary is open to energy from outside, but it is closed to external materials.
- The Processes available are represented by the arrow on the boundary. We can say that:
 - The processes are the *doing* of a living system.
 - The boundary is the *being* of a living system.

According to this Theory, a living system must have a complete set of different processes to be able to function as a living system.

- The Nervous System. The Nervous system of a living system is like a connection between external events and the processes within a living system.
- The Communication Channels which represents a two way communication between a living system and its external environment.

Value4All application is a social network system and by this theory can represent a living system (Value4All as the boundary with its own available functionalities). Value4All users can perform different actions by functionalities defined in Value4All (the processes). A user can introduce new projects in Value4All, which means that Value4All is open to new energy but users can't perform actions that aren't defined in Value4All, which means the system is closed to external materials. The communication channel between Value4All and the world outside is the site available online with its users.

5.3 Social Network System Functionalities

There are many functionalities that can be implemented in a social network system. The functionalities are many and the set of functionalities are increasing all the time. Here are, some of the basic functionalities.

- Identity Management. For instance, in Facebook, users can fill in information about themselves to access the site. Once you are granted access to the site you can begin to *socialize*, which means that you can create different groups of friends or family and search friends.
- Contact management. For instance, you can decide your family group or friends group. You can also read the profile pages of other members and possibly even contact them and put them in a specific group.
- Exchange. For instance, in Twitter you exchange information about what you are doing right now or in Facebook you can exchange pictures. You can learn new things about new cultures or new languages and learning is always a good thing.
- Expert Finding. For instance, in LinkedIn, companies can see people's CV and find new experts for their companies.
- Share common interests in business, hobbies, politics or religion.

5.3.1 Social Network System Dangers

Social networks are great opportunities for *socializing*, finding friends and communicate to them but on the other hands, there are many dangers within social network systems, that we have to be aware of. I had sent a questionnaire to more than 100 private persons, and *I can mention that, some older people that knew what a social network system was but refused to participate and answer the questions, were worried about their children using such systems.* I think it is important to remember that, there are many dangers in the real life too, not only within the social networks. Some of the dangers and how to protect oneself from them within social networks are:

- Data theft. Don't share sensitive information.
- Viruses. Install anti-viruses program.

- The most prevalent danger is individuals that claim to be someone whom they are not. The important things here are to be aware of these dangers precisely in the same way when you are meeting strangers at clubs, bars, school, or work in the real life.
- Decide to meet someone from social networks. It is important to be able to make a clear decision before you can meet someone from the site.

Part III

Development Environment And Database Solution Proposal

Chapter 6

Development Environment

6.1 Development Language

Value4All is a web based three tier application, and the development environment has to be a flexible and well-defined environment with good market acceptance. For that purpose, it's important to choose a good programming language for development of Value4All. The possible web programming languages are, for instance, ASP.NET, PHP, Ruby, Java, Python, Ruby on Rails, Scala. The question is which of them can be suitable for Value4All? A debate between these programming languages had gone on for years, and you'd never find a good answer. That's because there are so many pros and cons of all these programming languages that make it impossible to get closer to the truth. Developers are building web applications using .NET, Java, Ruby, PHP all the time and almost none of them are failing because of the choice of the language. All of these programming languages are large and complex and need some experiences because otherwise you'll maybe do things wrong. A popular programming language is Java, and I think Java can be a good programming language for Value4All. You can find many free, popular and open source applications used in many three tier platforms with Java's support. I used Java as the programming language for programming of the prototype. Java is a programming language, developed by James Gosling at *Sun Microsystems* and released in 1995. Java is one of the most influential programming languages of the 20th century, and is widely used from application software to web applications.

6.2 Integrated Development Environment

An *Integrated Development Environment*, IDE, or an integrated debugging environment is a software application that provides complex integrated functionalities to develop

pers, for instance, a source code editor, a compiler, a debugger, a version control system, a class browser, a class hierarchy diagram and a refactoring system. The most popular Java based IDE in the market are listed below.

- Eclipse. Eclipse is a multi-language development environment, free and open-source IDE.
- Borland JBuilder. Borland JBuilder is a Java IDE for Windows, Linux, and Solaris. JBuilder Foundation Edition is a free version of JBuilder, which offers most of the JBuilder capabilities.
- Sun Java Studio Creator. Java IDE for Windows, Solaris, and MacOS. Java Studio makes it easy for relative junior developers to create a complex server-side applications, but will be less popular for experts who prefer to work directly with the code. Sun Java Studio is built on the free, open-source NetBeans IDE.
- IntelliJ IDEA. A powerful IDE that is popular among people who like a smart editor and many integrated Java related tools.
- IBM WebSphere Studio. Java IDE for Windows and Linux. It is expensive but very powerful IDE for servlets, JSP, and other J2EE development.
- Oracle Weblogic Workshop. Oracle-BEA WebLogic Workshop is a very powerful IDE for developing applications, which runs on Windows, Linux and Solaris, and requires a Weblogic Server.

That was difficult to choose an IDE. For me, Eclipse is the favorite IDE but Eclipse is hard to set up and there are many extensible plug-in systems for Eclipse to configure. Crisp recommended IntelliJ IDEA because it's easy to set up, configure, and they had good experiences within IntelliJ IDEA. IntelliJ IDEA has many integrated Java related tools such as refactoring system, versioning systems and Maven, which make the development easier. IntelliJ IDEA is a commercial Java IDE by JetBrains. Also an open source Community Edition is available. I used IntelliJ IDEA for development of the prototype.

6.3 Build Tools

A Java Build tool is a compiler for the Java applications. The output file of a Java build tool are Java class files containing platform independent Java bytecode. The Java Virtual Machine (JVM) loads, the class files and either interprets the bytecode or compiles it to machine code. There are two popular java build tools, Ant and Maven.

- Ant. Historically, Ant is an older tool than Maven. Ant is something similar to Make which most of C++ developers use. Like Make, Ant is an extremely powerful tool which can do whatever developer wants. Ant use XML-tree instead of writing shell commands and the developers have to create own build process.
- Maven. Maven requires a clear definition of what the project consisted of. Maven does everything Ant does and is an easy way to publish project information.

Ant and Maven are different. Ant is a building tool but Maven is more than just a building tool. For instance, maven has a standard way of how to handle a project and project dependencies are clearly defined that are downloaded automatically by Maven (this property helped me to integrate Cassandra Hector application to Value4All application. For more information see the chapter 11 (Proof-Of-Concept Application)). Relations between different projects and sub projects are based on a single source of a XML file (Maven POM.xml). Maven is also integrated in IntelliJ IDEA. I used Maven for programming of the prototype.

6.4 Versioning Tools

Version Control System is the management of changes to documents, applications, and other files. Version Control is used in different enterprise application development, where a team of people may change the same files. Changes are usually identified by a version number. Each version is associated with a timestamp and the person making the change. Different versions of a file can, for example, be compared and restored. The choice was between CVS and SVN.

- CVS. The Concurrent Versions System, is a free open source software version control system and is released under the GNU General Public License. The server side normally runs on Unix and CVS clients may run on a different operating system.
- SVN. Subversion is a version control system which is more flexible and featured than CVS. Most open source applications use Subversion as a repository such as SourceForge, Apache, Python and Ruby. There are many different SVS clients, for example, for a Windows user, Tortoise SVN is a great file browser for viewing, editing and modifying.

SVN is also integrated in IntelliJ IDEA and supports a flexible, easy to use software environment to keep a truck of the application files. SVN works faster than CVS and as well version management is better in SVN than CVS. It transmits less information

through the network and supports more operations for offline mode. I used SVN as versioning tool for programming of the prototype.

6.5 Applications Server, Web Server

An application server is a middleware server who is designed to run specific applications and might be used to run only one application, for example, a huge enterprise application which needs the entire server resources as RAM , ROM and CPU. On the other hand, a Web Server is a server application that delivers content, such as Web pages by using the Hypertext Transfer Protocol (HTTP), over the World Wide.

The question here is, what is the difference between a Web Server and an Application Server? Basically, a Web Server manages HTTP requests, while an Application Server manages business logic to different enterprise applications through any number of protocols. Based on the company need and functionality requirements of Value4All, a web Server with application server functionalities will be suitable for Value4All in the first version. Here is a list of web servers that were studied.

- Jigsaw. Jigsaw is W3C's Web server, which provides a HTTP 1.1 implementation and a variety of other functionalities implemented in Java.
- Tornado. Tornado Web server is multi threaded written in Java. Tornado is a secure, efficient, portable Web server and also provides a complete implementation of HTTP 1.1.
- Pygmy. Pygmy is a small Web server, with a core source around 40kB, for embedding into applications. Features can be optionally added and removed to reduce the pygmy's already small binary size.
- Jakarta Tomcat. Tomcat is the servlet container that is used in Java Servlet and Java Server Pages, JSP. The Java Servlet and Java Server Pages specifications are developed by Sun under the Java Community Process.
- Jetty. Jetty is a powerful and flexible Java HTTP Server and Servlet Container, which means that you do not need to set up, configure and run a Web server and a Servlet container separately to use Java, Servlets and Java Server Pages to generate dynamic content.

Jetty can be integrated on IntelliJ IDEA in an easy way and is a very flexible Web server with integrated Servlet Container, which makes it easy to configure and run in the developing side of the application. Tomcat is also suitable and can be used in the production side of the application. I used Jetty for development of the prototype.

6.6 Persistence technique

The comparison was between relational databases, Neo4J as a graph database and Cassandra as a Key-Value store. According to analysis available in the chapter 8 (Why Not Relational Database For Value4All) and in the chapter 11 (Proof-Of-Concept Application), Neo4J is the winner database which is a powerful, easy to use graph database with an extremely good open source implementation in Java. I used Cassandra and Neo4J for development of the prototype.

6.7 Java Frameworks

A framework makes it easier to work with complex applications because there are many predefined codes, which are reusable and also extensible. There are many java frameworks, which can be used in Value4All based on needs. Here, you can find a list of frameworks suitable for Value4All.

- Tapestry. Tapestry is a powerful, open source, Java framework for creating web applications in Java. Tapestry is an alternative to Java Server Pages, JSP, or Velocity, which provides to create extremely dynamic applications with minimal amounts of coding.
- JUnit. JUnit is a powerful, free, open source Java framework that makes the application unit testing easier and more effective. JUnit is developed for test driven development, which is useful for larger enterprise applications.
- Java Server Pages (JSP). JSP is also used for development of web applications.

I used JSP for development of the prototype.

6.8 Test automation control

Test automation checks an application's outcome with predicted outcomes by different test cases and test logging functions. Selenium is a powerful open source program for test automation of web applications, which runs the test cases directly in a browser, just as real users do.

6.9 Other Techniques and Tools

Here you can find a list of other techniques that can be used in Value4All based on needs:

- HTML. Hyper Text Markup Language is a programming language used for pages that are viewed by web browsers.
- JavaScript. JavaScript is an object oriented scripting language that provides HTML pages to interact with clients and respond to what they do dynamically.
- Cascading Style Sheets, CSS. CSS is used for styling the content of HTML pages.
- XML. EXtensible Markup Language, XML, is a way of how data can be transformed between different layers of an application or across the internet to be used by a web browser and viewed to visitors.
- Ajax. Asynchronous JavaScript and XML, is used to design and implement dynamic web applications that are actually a group of web development techniques used on the client side creating interactive web applications.
- Photoshop. Adobe Photoshop is a graphics editing tool developed by Adobe Systems.

I used JavaScript, HTML and Paint.Net instead of Photoshop for development of the prototype.

6.10 Results

Here, you can find a complete list of the solution proposal for the Value4All's development environment.

- Web Development Language: Java.
- Integrated Development Environment: IntelliJ IDEA.
- Build tool: Maven.
- Versioning tool: Subversion, SVN.
- Web Server: Jetty
- Database Server: Neo4J.
- Java Frameworks: Tapestry, JSP and JUnit.
- Test automation tool: Selenium.
- Other techniques and tools: HTML, JavaScript, Servlets, Java Beans, XML, Ajax, CSS and Fotoshop.

6.11 Download Pages

In the appendix F (Application Installations), you can find all download pages to the applications mentioned in the chapter 6.10 (Results).

Chapter 7

Persistence Layer - Solution Proposal

7.1 Introduction

Using social networks are growing fast, and the data volumes are growing drastically. On the other hand, the current tools, data storage models and management of data for social network systems still are not good enough. Today there exist different data models and some of them are suitable for a social network system. The best choices of data model implementation for a social network, take into considerations what social network needs, for instance, automated data management, the integrity management and security management of social network's information even against system crashes or attempts at unauthorized access. Also, if data are to be shared in a social network system among several users, the system must provide a concurrency mechanism (the property of an application where many computations are executing simultaneously, and interacting with each other) avoiding possible results in error states. Some data models are discussed in this chapter, and the best data models suitable for Value4All are selected. The discussion is as follows:

1. Understand the data structure of Value4All as a Social Network System.
2. Analyze all possible data models and based on the Value4All data structure choose which of data models can be suitable for Value4All.

7.2 Value4All Data Structure

Basically, a simple example of how the data structure of Value4All can look like is described in the chapter 2.2 (General Problem Description).

7.3 Data Models

Data models are used for representation of information in a machine language. A *database model* is a developed application of a specific data model which is managed by a *database management system* (DBMS is an application that controls the creation, maintenance, and the use of a database). Database models are powerful tools for organizations and for representation of information. There are different developed data management systems for different database models that are based on different data models, for instance, in a relational database, there are different tables for an organization data or a XML (extensible markup language) can be used as a document database. With a database, we can retrieve information and manage the information. There are different data models, which are based on different data structure needs in the market.

- Flat model
- Hierarchical model
- Network model
- Relational model
- Entity-relationship model
- Object model
- Graph model
- Key-value model
- Document model

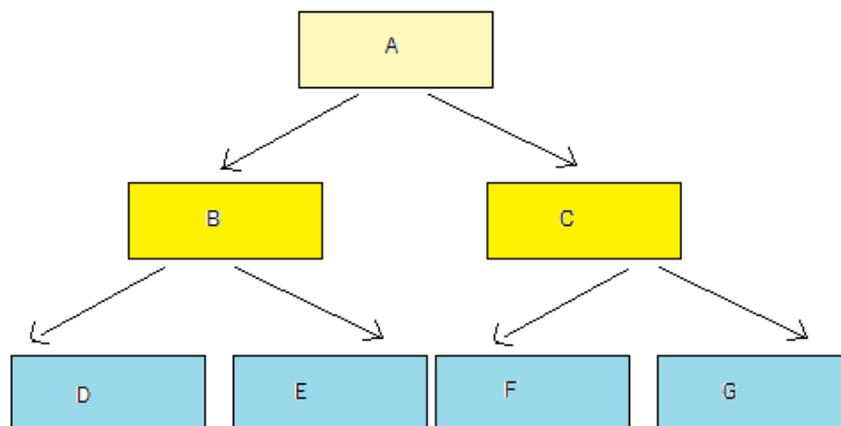
7.3.1 Flat model

The flat model (flat-file model) consists of a single table (two-dimensional array of data) saved, for instance, as a .txt file, an .ini file or a .xml file. In such table all data of a column are assumed to be similar values (similar structure of data) and also all data of a row are assumed to be related to each other. For instance, (name/password columns) with similar data in each column and each row is associated with an individual user. XML is a popular and well-used format for storing data in plain text files, but XML allows also very complex data structures to be represented with definition of the data.

	Name	Product	Price(Kr)
Record 1	Kalle	MP3	1 200
Record 2	Johan	Laptop	12 500
Record 3	Stefan	MP3	1 200

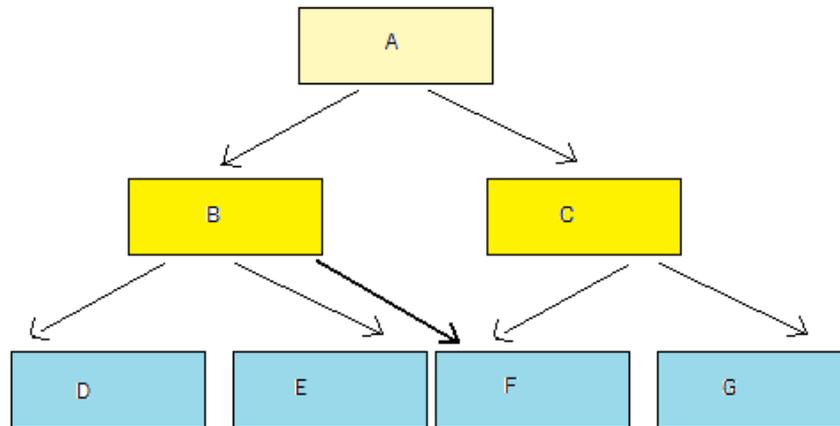
7.3.2 Hierarchical model

In this model, data is stored into a tree-like structure, with each node related to its child nodes and its parent node indicating the hierarchical relationship between them, for instance, a child can only have one parent but a parent can have multiple children. A Parent has a list of all of its children. For instance, a XML document follows this hierarchical data model.



7.3.3 Network model

The network model consists of records with different fields that define relationships between records. The network model is a variation of the hierarchical model. The model differs from the hierarchical model in that, nodes from different sets can be related to each other.

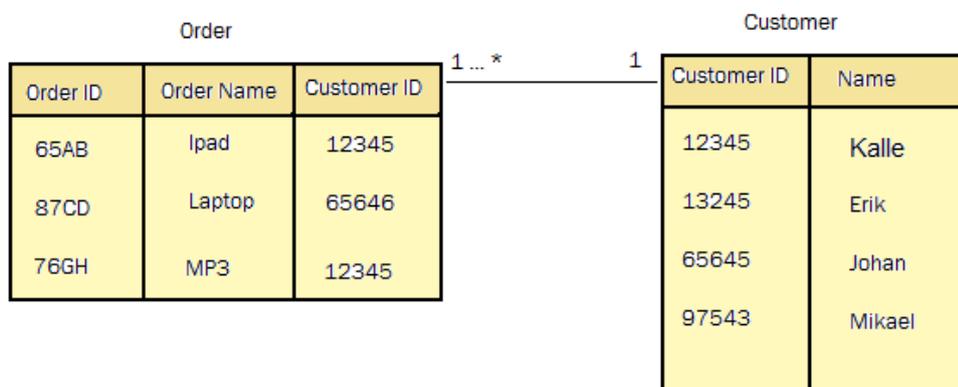


7.3.4 Relational model

The relational model is a mathematical model defined in terms of predicate logic and set theory. Three key terms are used in relational database models:

- Relations
- Attributes
- Domains

A relation is a table with columns and rows. The named columns of the relation are called attributes, and the domain is the set of values the attributes are allowed to take. Relational databases consist of a collection of tables that store different sets of data instead of placing all data in one large table. Typically, a relational database consists of many related tables. Each table contains specific data, for example, name and address. Relational databases are the most used database today.



7.3.5 Entity-relationship model

An entity-relationship model (ERM) is an abstract representation of data. Entity-relationship modeling is a database modeling method, used to produce a schema of a system. Diagrams created by this process are called entity-relationship diagrams, ER diagrams, or ERDs.

7.3.6 Object model

An object model (object-oriented model), the information is represented in the form of objects from an object-oriented programming. Object model has object-oriented aspects like class, attributes, methods and integrity constraints. Object model also supports multiple inheritance and abstract data types.

7.3.7 Graph model

A graph model uses nodes, edges and properties to store information and represent the information. This model is an alternative to relational databases, which use tables, document-oriented databases, which use documents to store information or other types of structured storage database systems. With other words the databases based on a graph database model is characterized by three properties:

- Data structures are graphs or any other structure similar to a graph.
- Data manipulation and queries are based on a graph-oriented operations (graph traversal properties).
- There are data constraints to guarantee the integrity of the data and its relationships.

Compared to the relational databases, graph databases are often faster for associative data sets. They can scale more flexible to large data sets without any join operations. This model is one of the best choices for designing a social network based on the structure of data with offering high performance functionalities for querying the data of a social network system.

7.3.8 Key-Value model

Key-Value model is based on a hash table of (key, value) pairs. In this model, any participating node can efficiently retrieve the value associated with a given key. This model is very powerful and efficient because adding or removing a record is extremely

flexible and it scales to extremely large numbers of nodes. This model is a part of a NoSQL database model. A database based on a distributed key-value model can be also a good choice for Value4All because it is suitable for a large amount of data, high availability, no single point of failure (fault tolerance) and high performance.

7.3.9 Document model

The Document model is designed for document-oriented applications. Document databases store each record as a document with different properties (for instance, when the document was created and by who) instead of storing data in tables.

7.4 Data Models Comparison Analysis

The *flat-file model* of database is ideal for small amounts of data that needs to be human readable or edited by hand. It consists of a set of strings in one or more files that can be retrieved easily. This model is great for storing simple data values, but can get complicated with more complex data structures because the file gets more and more unreadable, and at the end we lose the meaning of such a solution. It is still possible to store such complex data structures but doing so can be more costly in time and performance, for instance, compared to a relational database. A flat model is unsecured and is very sensitive to corruption because it is very difficult to detect when a file is being used or modified. A flat-file model was unsuitable for Value4All, but I have to mention that a flat-file solution is very flexible and fast to redeploy at the first stages of development of an application. Value4All will use a XML file solution as its first version database solution. An *entity-relationship model* (ERM) which is an abstract representation of data is not either what we are looking for since it is more to present diagrams created from a database system.

A *hierarchical model* is unsuitable for Value4All and it can be explained by an easy example. For instance, an organization may have information about an employee in a table (Employees) with columns like Name, Responsibility and Wage. The company may have another table for all employee's children (Children) with columns like Name, Data of birth. We have a 1:N relationship between them (an employee may have no children or one or more children). We have a hierarchical structure in our database, which is not applicable to a social network system. In a social platform, we have something like a meshed structure. The *network model* was the first attempt to address the problems and inefficiencies of the hierarchical model but network model is also

a type of a hierarchical model which is unsuitable for Value4All. The *relational model* has proven to be the most efficient and most flexible database model in use today. There are many advantages of the relational model over the other models, which is why the most popular databases in use today employ this methodology. A relational database model like Mysql and Oracle are used by many companies. A relational database contains multiple tables with methods for the tables to work together. The relationships between table data can be retrieved by SQL query-methods, data can be merged and displayed in database forms in application interfaces. Many of the today social network systems like Twitter and Facebook are designed upon a relational database system. Based on that One of the best choices for Value4All was a relational database model to be compared to other choices.

Object model databases extend the functionality of object-oriented programming languages (e.g., C++, Smalltalk, Java) to provide database programming functionality. The result is a high level of compatibility between the application data type and the database data type, resulting in less code, and better reusability of code. For instance, C++, Java, and Smalltalk programmers can write complete database applications with using the same data type as in the programming languages they use. Object model databases offer the ability to reduce code. This is a good idea if the application data model contains many different complex data types, which require creating different classes to take care of the data and this ability of an object oriented data models to support many classes of objects, each one with their own methods and properties, do not offer any special advantage for data types of social networks, which are homogeneous, with a small number of classes. Another aspect is that, an object model is suitable for fewer records (rows in a database) and a big number of different objects, instead, Value4All will have a few objects (like user object, project object, and, etc. based on needs) but probably it will have a huge amount of data in the database. As mentioned in the chapter 7.3.7 (Graph model) and 7.3.8 (Key-Value model), a *graph model* and a *key-value model* can also be two good data model solutions for Value4All. A *document model* is not a good choice for Value4All, because a document model can't handle references between nodes that are related to each other.

7.5 Data Models - Comparison Result

The chosen data models were:

- Relational model (Relational Database or SQL database).

- Graph model (NoSQL database).
- Key-Value model (NoSQL database).

The comparison analysis of these models is discussed in chapter 8 (Why Not Relational Databases For Value4All).

Chapter 8

Why Not Relational Databases For Value4All?

Social network platforms are not only about developing a website but also managing a large amount of data that requires much knowledge about data management and load balancing (technique to distribute the workload across several database servers, in order to get optimal resource utilization, avoid overload, minimize response time and maximize throughput). Relational databases (described as table-based databases or SQL databases), like MySQL or Oracle Database, are widely-used databases in the market. In these databases you can add many records at the same time but these databases are not a good solution when the amount of related tables increase, because the data access and the data management require many joins and unions (this operations are high performance consumers). Of course, there are ways to come around such limitations by scaling the relational database system. For instance, Adam Wiggins of Heroku in his article (*SQL Databases Don't Scale*, <http://adam.heroku.com/>), describes some techniques (vertical scaling and sharding) for management of scalability and performance of relational databases for huge applications along with listing their downsides. According to the points mentioned below, relational databases are not enough as a database solution for Value4All.

- Value4All is a *relationship* platform, for instance "Kalle" *knows* "Hasse". The verb "knows" indicates a relationship between two persons. Relationship in a relational data model is weak, for instance, you may define a new relationship as a one or more tables, more constraints or more codings. This is a strong reason to why relational databases are not a possible solution for Value4All. Basically, the *changeability* property is not strong enough on relational databases.

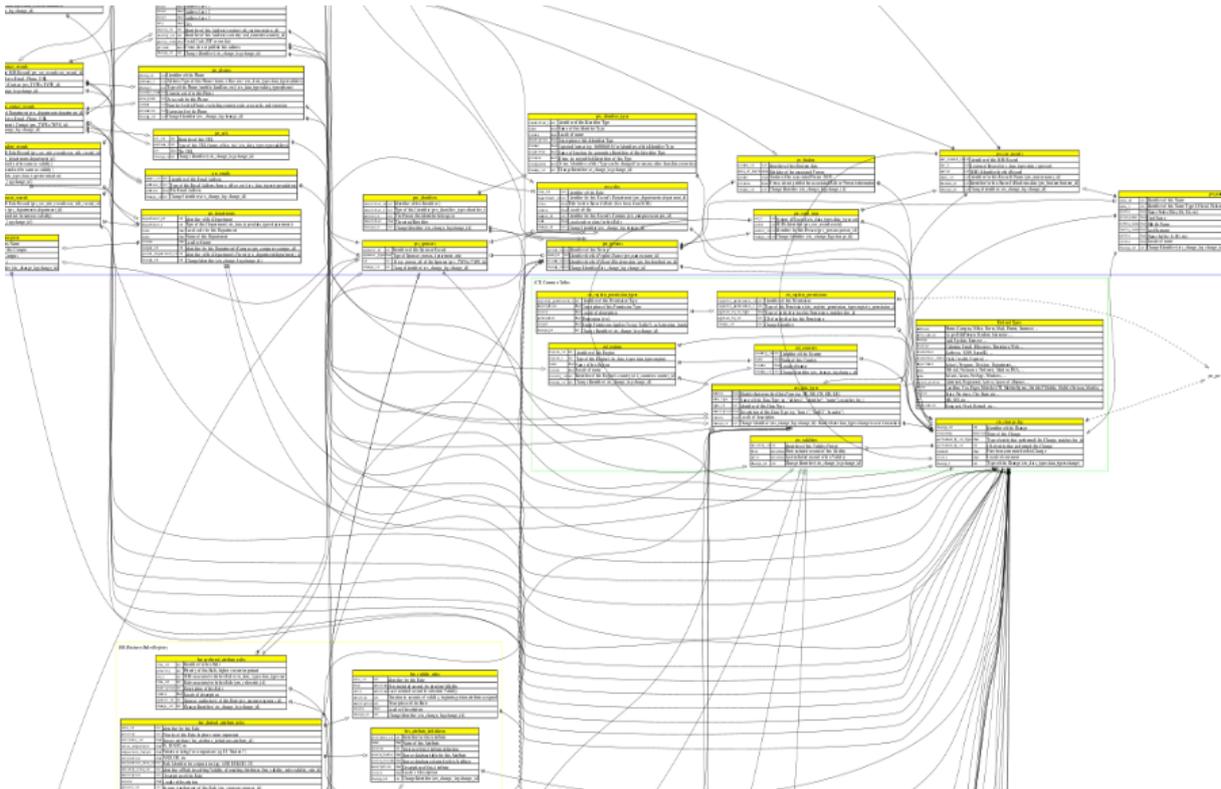
- Relations can also be coded easier in a graph database than a relational database. It is not easy to understand the data model when the number of tables increases based on the increasing number of relations. Because of that, the *easy-to-use* property is not strong enough on a relational database to be used as a possible solution for Value4All.
- Extremely Query loaded. A relational database is not a good choice compared to a graph database or a key-value database because it is very hard to get efficiency doing joins in such a high level. Because of that, the *performance* property is not strong enough on a relational database to be used as a possible solution for Value4All.
- Huge Data Volumes. For example, in a social network system (Facebook) there are massive data volumes, which have to be stored. Such systems are usually distributed across a cluster of servers. Huge amounts of data are a problem for relational databases because of the scalability problem. Relational databases don't follow the definition of a true scalable system (see the next section 8.1 for more information), and this is another reason to why we canceled relational databases off the list. In loosely comparison of *scalability* we can see in the picture below that relational databases will not follow the scalability requirement of a social network system.



Another type of databases is NoSQL databases (Non-Relational Databases), which is described as the next generation of data storage technologies that are designed for web applications that need to scale effectively. These databases don't require fixed table schemas, avoid joining operations, avoid primary keys and foreign keys.

8.1 Relational Databases - Scalability Analysis

There are many relational databases, for instance, MySql and Oracle. MySQL is a relational database management system that provides multi-user access to a number of databases.



MySQL was owned by the Swedish company MySQL AB, and now owned by Sun Microsystems. MySQL is also used in many popular, large-scale web applications like Wikipedia, Google and Facebook.

The improvement of hardware help relational databases like MySQL to become more *scalable* with improved *performance*, for instance:

- Improvement of the capacity.
- Improvement of speed of CPU. For example, by multi-core CPUs, 64-bit micro-processors, and, etc.
- Improvement of speed of networking devices. For instance, Network-attached storage (NAS), Storage area networks (SANs) with fast local area networks and Fibre Channel technology.

But still, there are different *scalability problems* that facing relational databases. Adam Wiggins says that “SQL databases are fundamentally non-scalable, and there is no magical pixie dust that we, or anyone, can sprinkle on them to suddenly make them scale”. According to Wiggins A., a technique must fit the following criteria to be a true scaling system:

- *Horizontal scaling.* Horizontal scaling means adding more computer systems to the environment which increase the capacity.
- *No single point of failure.* No single point of failure means, in case of using some devices or communication lines to perform a function, ensuring continuous operation without no downtime. Any server in a distributed/cluster system creates a single point of failure.
- *Transparent to the application.* In case of a three tier architecture, a well-designed application has a strict partition between the presentation layer, business layer and database layers. For instance, scaling database servers or other types of servers to the system have to be transparent to the application.

There are different techniques that address the scalability problem of Relational databases:

- *Partitioning.* Partitioning is a logical division of a database into separate independent databases for improving the manageability, performance, scalability and availability. The partitioning can be done by either adding additional databases with its own tables or by splitting tables between the additional databases. There are two different partitioning of data (*Horizontal partitioning* and *Vertical partitioning*). Horizontal partitioning means to put different rows into different tables, for instance, if the rows increase more than 10000 rows, the table can be partitioned in another table. A view with a union of these two tables can be created. Horizontal partitioning requires deep modifications into the application itself because the object models of the database tables have to be modified. Vertical partitioning means dividing tables' columns to different tables with a union view of them. Horizontal partitioning and vertical partitioning cause more problems if you have many relationships (many related tables) in your SQL server. (*According to definition of a true scaling system that was mentioned above, you have no transparency in this case, between the database layer and the business logic*). This problem also causes that the changeability of relational databases is very hard to achieve because of the complexity of the data model. For instance, if you want to add a new property "age" to your user tables, you need to change your data model by SQL script, and also the code modifications have to be done. Such changes are not flexible and are very hard to achieve when the number of relations increases. Social networks are relational systems and because of that a relational database is unsuitable.

- *Vertical Scaling* (referred as Moore's law). In this case, you can have a bigger server to scale your server but this solution is not recommended because usually such servers are old and the old server is useless, and you have to buy a new bigger one and maybe after some years you have to buy a new one. Also to understand when you need a new server require professional skills. The installation of the new one needs professional skill too.

The question is what is the alternative to the SQL database scaling problems?

SQL databases are widely-used in the market by the popular and big companies, and they maybe have solved their scaling problems with one of the solutions presented above but these solutions are very expensive and very complex to manage. Another solution is to use a *NoSQL* database instead of a SQL database which addresses these problems. Several NoSQL databases have a *distributed* architecture by distributed hash table solutions. In this way, the system can be scaled up easily by adding more servers with a guarantee for no single point of failure. There are some other differences between NoSQL and SQL databases:

- SQL databases have good availability but NoSQL databases have very high availability.
- SQL databases provide SQL-queries (SELECT, INSERT and, etc.) while NoSQL databases use other techniques like graph traversal for graph databases that, is a very flexible solution suitable to the structure of social network's data.
- ACID (*atomicity, consistency, isolation, durability*) are four fundamental properties defined in a SQL database. (*Atomicity* means that transaction must go all the way, or it has to be rollback. *Isolation* is the requirement that other operations cannot access data that has been modified during a transaction that has not been completed yet. *Durability* is the DBMS's guarantee that once the user has been informed by a transaction's success, the transaction will not be lost.) A strong property of SQL databases is *strong consistency* that means you can't find any information in a database that is against each other. That means any transaction from one consistent state will end to another consistent state. The thing is that ACID doesn't scale well, and the consistency is the tradeoff to get the scalability to work in relational databases. Social network system is not like a banking system and because of that they don't need the strong consistency as the banking systems do. Several NoSQL databases provide eventual consistency, which

is a weak consistency. The term means "when no updates occur for a long period of time, eventually all accesses will return the last updated value". Some NoSQL databases provide full ACID guarantees by additional configurations. Werner Vogels (The Chief Technology Officer and Vice President of Amazon.com, http://en.wikipedia.org/wiki/Werner_Vogels) says "Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability".

According to the *CAP theorem* (Brewer's theorem), it is impossible for a distributed computer system like a NoSQL database, to provide all three of the following guarantees and achieve the scalability at the same time (these three requirements are needed for a distributed system):

- *Consistency* (all nodes see the same data at the same time in the case of strong consistency). The distributed database guarantees that, users see the latest information but there is no guarantee that all nodes of the distributed system have the latest information at the same time. Eventual consistency is tradeoff between strong consistency and high availability.
- *Availability* means no downtime in case of a node failure.
- *Partition Tolerance* (the system continues to operate despite split into disconnected subsets). For example, not only for reads, for writes as well.

According to the theorem, you can satisfy at most *two* of the three requirements with *eventual consistency against high availability*. For instance, Cassandra guarantees the AP (Availability and Partition Tolerance) and eventual consistency providing high scalability.

8.2 Types Of NoSQL Databases

Non-Relational Databases (NoSQL), are databases without tables, joins, unions, primary keys, foreign keys and schema. For instance, Google's BigTable, Amazon's Dynamo and some other open source implementations as Neo4J, HBase and Cassandra which fit into different types of NoSQL databases.

- *Document based*. A document database is designed for storing documents (for example, research papers). These databases may be implemented as a layer above a relational database or an object database, for example, CouchDB, MongoDB, Terrastore and XML databases as eXist.

- *Graph*. A graph database (for example, Neo4J, HyperGraphDB and Allegro-Graph) uses nodes and edges to represent and store the information. Graph databases are an alternative to relational databases, document databases or other structured storage systems based on key-value store.
- *key-value store*. In such databases, you have only one key and its value with no duplicates with high accessibility and performance. For example, Dynamo, Voldemort, Cassandra and BigTable which store key-value (s) on disk, Redis and Hazelcast, which store key-value (s) in RAM.
- *Column Database*. Column databases are a distributed multi-dimensional storing-map and are referred as *BigTable Clone*, for example, BigTable, HBase and Cassandra. Column databases store their content column-wise instead of row-wise. In this way, the data similar to each other are stored together and accessing the data are more efficient if we are looking for some specified columns (a subset of columns) or specific data. Row-wise store is good if you want to access the whole row information. For example, if we have a table of employees:

ID	Name	Salary
1	Hasse	50 000
2	Kalle	7 000

This information is serialized to the hard drive or to the RAM. If these series of bytes are stored row-wise, the information is serialized, for example, something like this:

1,Hasse,50000;2,Kalle,7000;

If these series of bytes are stored column-wise, the information is serialized, for example, something like this:

1,2;Hasse,Kalle;50000,7000;

Based on the results available in the chapter 7.4 (Data Model Comparison analysis), a document database was not a solution that, we were looking for. On the other hand, a graph database, a key-value store and a column database were three NoSQL database solutions that are suitable for Value4All. Some databases like Cassandra and BigTable

are both a column database and a key-value store. From this chapter, these databases are referred as key-value stores. Based on analysis available in the chapter 11 (Proof-Of-Concept), graph databases and key-value stores are compared to each other by a prototype as two NoSQL databases and based on the analysis available in this chapter 8.1 (Relational Databases - Scalability Problem) and the chapter 3.1 (Market's Database Solution) relational databases are out of the game. In appendix C (NoSQL Databases - Comparison), you can see a complete comparison between different NoSQL databases.

Chapter 9

Why Choose Cassandra?

9.1 Introduction

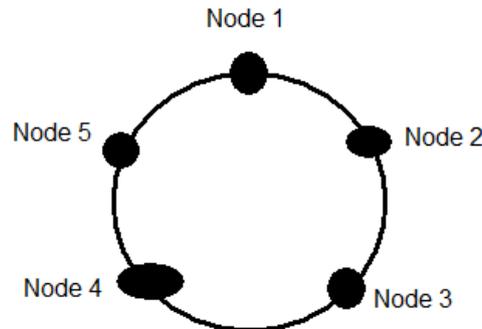
Key-Value stores have a very powerful data model with the fundamental mapping keys to a list of values as its basic structure. With key-value stores we can store schema-less data as a key-value pair with high performance as an alternative to relational databases. Key is usually a string and value can be a primitive data type, an array, a HashMap or an object. A good choice, to store values as they are without needs to create different tables in a relational database based on a distributed hash table. With a key-value store, it is easy to achieve high performance, no single point of failure and high availability because of its flexible schema-less data models and fine granularity partitioning of the data (load balancing of the data).

9.2 Distributed hash table

Distributed hash table (DHT) is a type of decentralized distributed systems with no master node. Any member node in the distributed system, can efficiently retrieve the value associated with a specific key. All *nodes* in the distributed system have the responsibility for the mapping keys to values, in such a way, a change in the set of nodes causes a minimal disruption (no single point of failure) to the entire system. This property gives the DHT, the ability to *scale up* to millions of nodes in the node network.

A key technique that is used in DTS to achieve the properties like decentralized, scalability and no single point of failure, is that any node involved in the DHT, needs to communicate with only a few numbers of nodes (most commonly, results in, $O(\log n)$ performance with n nodes). That means, only a little work needs to be done for each change in the DHT topology because every node has a list of some other nodes (for

example, their neighborhoods by a routing table). Every node in such a topologies can find out, for example, which node has what particular key-value pairs based on the id of available nodes in its list.



Imagine that you have a circular, *double-linked* list of nodes (each node is a machine). Each node has a reference to some other nodes in its available list (a hash table of other machines). If we want to lookup the value of a key, we can do that by the id of its node (machine). For example, look up of a key (K), the node either has a node id that has the K stored in it or send the lookup to the closest node that can lookup the value of K in case, the current node itself doesn't have the key (K). At each step, the current node forwards the lookup message to the node whose id is closest to the key K until it reaches the node that has the value of K (*Key-based Routing*). There are many popular projects that use DHT, for instance, Apache Cassandra, BitTorrent, Tapestry.

9.3 Consistency Levels

There are different levels of consistency. Cassandra provides eventual consistency.

- Strong Consistency. Strong consistency guarantees that you can't find any information in the database that is against each other, for instance, RDBMS, Local Disk and RAM.
- Eventual Consistency. When no updates occur for a long period time, eventually all accesses will return the last updated value.
- Weak Consistency. Weak Consistency offers no guaranties.

9.4 Example of Key-Value Stores

There are many popular key-value stores, like:

- *BigTable*. Google uses the own key-value store BigTable, to map URLs to multidimensional data, such as keywords, links, Timestamps, Versions, and, etc.
- *Dynamo*. Dynamo is created by Amazon to address the need for an *incrementally* scalable, highly-available key-value storage.
- *Cassandra* (an Apache Software Foundation top-level project). According to Cassandra Project homepage: “Cassandra (implemented in Java) is a highly scalable, eventually consistent, distributed, structured key-value store. Cassandra brings together the distributed systems’ technologies from Dynamo and the data model from Google’s BigTable. Like Dynamo, Cassandra is eventually consistent. Like BigTable, Cassandra provides a ColumnFamily-based data model richer than a typical key-value systems (but Cassandra uses a Distributed hash table (DHT) instead of a central server)”.
- *Redis*. Redis is an in-memory key-value store design to achieve high performance, implemented in C. Redis uses RAM as the head storage area and only flush out its data to disk asynchronously. This is not ideal for important data, which can be lost in case of a server crash.
- *Voldemort*. Voldemort is a distributed key-value store that combines in-memory-caching with the storage system. In that case, a separate caching system is not required (instead the storage system itself is very fast). According to the Voldemort project homepage (<http://project-voldemort.com/>): “It is used at LinkedIn for certain high-scalability storage problems where simple functional partitioning is not sufficient. It is still a new system which has rough edges, bad error messages, and probably plenty of uncaught bugs. Let us know if you find one of these, so we can fix it”.
- *Hazelcast*. A fast, in-memory and highly scalable key-value store in Java.
- *HBase*. HBase is an open source, distributed database similar to Google’s BigTable in Java.

Among these databases Cassandra has already many strongness that are needed to be widely used as a key-value store in Value4All based on the researches that are described in the chapter 3.1 (Market’s Database Solutions) many companies use Cassandra as its storage solution.

9.5 Cassandra

Cassandra is a NoSQL database that was developed by Facebook from the beginning to improve their Inbox Search feature. One of the most widely-used NoSQL database is Cassandra, which is a column-oriented database and also a distributed key-value store based on a distributed hash table. Keys are distributed to nodes according to a hash function, and values are retrieved with $O(\log N)$ messages by performing routing tables. Cassandra is also a P2P system because every node is aware of all other nodes in the cluster.



According to Avinash Lakshman (one of the authors of Amazon's Dynamo), Cassandra is designed for:

- *High Availability.* Losing a node doesn't take down the cluster and the functionality of the cluster.
- *Eventual consistency* (Trade-off between strong consistency in SQL databases and high availability in Cassandra).
- *Scalability.* Huge amounts of data (the ability to scale to many millions of data) distributed across many servers (the ability to scale across many nodes).
- *No single point of failure* (means the ability to adapt to cluster-network changes), which means that Cassandra is decentralized, and you can read and write any data to anywhere in the cluster at any time. No single point of failure is important to achieve high availability and load balancing.
- *Low total cost of ownership.* It is cheaper to use Cassandra than other relational databases.
- *Minimal administration.* It is also fewer administration needs for Cassandra than other relational databases.
- *Continuous Deployment.* Cassandra is *schema-less*, you do not have to decide what fields you need in your records when you start designing. You can add and remove arbitrary fields at any time. This is an incredible opportunity to increase the productivity, performance and availability.

Scale is the primary reason to why you would choose Cassandra. Traditional RDBMS starts to struggle when you want to use more than one node, and creating big clusters are currently only available with special solutions like, using for example expensive shared-disk technology. The scalability is provided in Cassandra by default without special solutions, for instance, Facebook is running a 150 node Cassandra cluster and others have 30+ node clusters in production also. Cassandra's powerful *data model* is built up of:

- *Key-Value* pairs of data, for instance, (name: "Kalle").
- *Columns*. *Columns* are the next smallest piece of data in Cassandra, and it contains of a name, a value and a timestamp, (also a key-value pair with a timestamp). For example:

```
{name:"Salary", value:50000, timestamp:201009101230}
```

For simplicity I refer to the above example as:

```
Salary: {50000, 201009101230}
```

The next level of the Cassandra's data model is a *SuperColumn*, which is also a key-value pair (without a timestamp) but the value is a *map containing an unbounded number of Columns* (column-name is the key for the value). For example:

```
{
  name:"address",
  value: {
    street: {name:"street",value:"Sveav.", timestamp:1234567}
    zip: {name:"zip",value:"12345", timestamp:1234567}
    city: {name:"city":,value:"Stockholm", timestamp:1234567}
  }
}
```

For simplicity I refer to the above example as:

```
Address: {
  street: "Sveav.",
```

```
zip: "12345",  
city: "Stockholm",  
}
```

- *ColumnFamilies*. ColumnFamily group both the Columns and SuperColumns with an infinite number of Rows (each row is a key-value pair with value as a map of Columns, and name as the name of the column). There are two types of ColumnFamilies, standard and super. In a standard ColumnFamily, each key-value map contains of only normal columns and in a super ColumnFamily, each row contains of only SuperColumns. Each Column family has also a specific unique Key as the name of the ColumnFamily. *Each ColumnFamily is stored as a separate file on disk*. This design allows for a certain *data model design* which has to be done before the start of the application in the "storage-conf.xml" file of Cassandra. This means that when you select an existing ColumnFamily in Cassandra's Config file, you will receive a group of key-value pairs of data. The data is automatically indexed and allows for fast data access.
- *Keyspace*. The Keyspace is the highest level of data in Cassandra data model. All ColumnFamily is packed inside a Keyspace (can be the name of the application defined in the "storage-conf.xml" file before startup), but it doesn't mean that, there is a *relationship* between ColumnFamilies (Keyspace can be compared to a RDBMS schema and ColumnFamily to a RDBMS table). ColumnFamilies can't be joined like tables in a RDBMS system.

9.5.1 Read And Write Operations

Write Operation

A client sends a write request to a *random node* in the Cassandra cluster. The Cassandra cluster of nodes is stored as a "*ring of nodes*" and the write requests are replicated to *N* nodes for redundancy using a *replication placement strategy*, for instance, *RackAwareStrategy* based on the *distance*. Cassandra determines the "*distance*" from the current node in three levels (achieving reliability and availability): same rack as the current node, same data center as the current node, or a different data center. Each node in the cluster when receiving a write request (a RowMutation message) performing two actions: the data first added to the *commit-log* for transactional purposes and then to an *in-memory* table (Memtable). There are some additional *asynchronous* operations (high efficiency) after that: the data is stored from *commit-log* on a dedicated disk locally (SSTable) so

we don't get too much data in-memory. Appending to the disk is the slowest part and because of that, unlike a relational database, Cassandra does not update data or indexes in-place on disk, so there are no intensive synchronous disk operations to block the write operation. According to Cassandra the write operation time in comparison with MySQL for more than 50 GB data is approximately as follow:

- Cassandra: 0.12 ms.
- MySQL: 300 ms.

Read Operation

Consider a write operation and a read operation that is very close to each other in time. For example, we have a key "Year" with value "2009" in your Cassandra cluster, and now you want to update the key "Year" to value "2010". The write request are sent to all N nodes in the Cassandra cluster and in each node it takes some time to write the data. In this moment, you send a read request for key "Year". In this case, some of the nodes have the new value "2010" and some of them have the old value yet "2009". They will all eventually return the new value "2010" but it's not guaranteed for just a few milliseconds. A client sends a request to a *random node* (called the Storage Proxy) in the Cassandra cluster. The proxy determines the N nodes in the ring based on a *replication placement strategy*, that can hold a copy of the data. The proxy forwards the read request to all this nodes having the data. Because of the eventual consistency limitations, Cassandra allows the client select the strength of the read consistency:

- Single read: the proxy returns the first response it gets.
- Quorum read: the proxy waits for a majority to respond with the same value which is a little slower.

In the background, the proxy also performs *read-repair* on any inconsistent responses. The proxy will send a write request to any nodes returning older values to ensure that the nodes return the latest value in the future. Since Cassandra has two levels of storage, Memtable and SSTable, first the data is read from Memtable (in-memory), if the data is incomplete then the data is read from the SSTables (in disk). To scan the SSTable, Cassandra uses a row-level column index and bloom filter to find the necessary blocks on disk, deserializes them and determines the actual data to return. Cassandra does provide some row caching, which minimizes the disk read operations. Cassandra has also bloomed filters for efficient SSTable lookups. Read operation time in comparison with MySQL for more than 50 GB data is approximately as follow:

- Cassandra: 15 ms.
- MySQL: 350 ms.

As you can see the read operation is quite slower than the write operation in Cassandra. It happens in case of uncached data, which is slower than cached data. As mentioned above, the reason is first the data reads from the memtable (in-memory) and in incompatibility case of data, then the data have to be read and merged from SSTables (in disk) to complete the request (SSTables are not updated in place in Cassandra. They are updated by asynchronously operations).

9.5.2 Partitioning

Designing a Cassandra cluster is very important. There are two methods for partitioning of Cassandra cluster:

- RandomPartitioner (RP).
- OrderPreservingPartitioner (OPP).

These two methods control how your data is distributed over nodes in the Cassandra cluster.

9.5.3 Replication And Consistency Level

As mentioned above, when writing in Cassandra, a client sends a write request to a *random node* in the Cassandra cluster. The Cassandra cluster of nodes is stored as a *ring of nodes* and the write requests are replicated to N nodes for redundancy using a *replication placement strategy (RackAwareStrategy)*.

The question is how many copies of each data we want to store in the Cassandra cluster (Redundancy)? Cassandra solves this problem according to strength of read and writes consistency:

- *Single read/write*: one node is updated or the data is read from one node.
- *Quorum read/write*: read and write by $(N/2)+1$ nodes.
- *All read/write*: read and write by all nodes in the cluster.

9.5.4 Cluster Membership

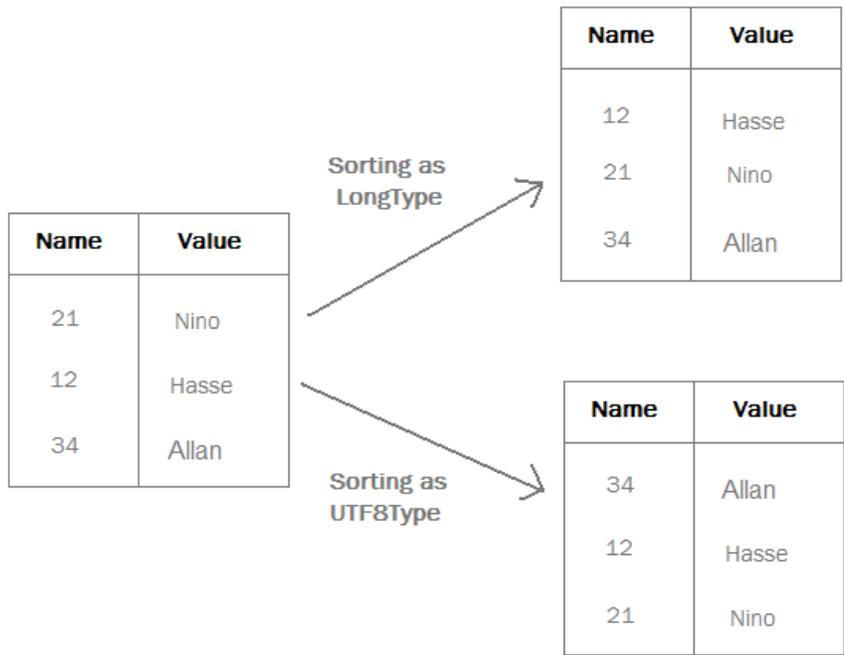
Cassandra uses the *gossip protocol* for node's membership management in the cluster. Every node gossips to some subset of other nodes (1-3) about the state of the cluster. Changes in such a cluster like adding nodes, removing nodes and failure of nodes' propagates quickly with $O(\log n)$ performance, n is the number of nodes in the cluster. Every T seconds, each node in the Cassandra cluster chooses a node in their node list, to send their node list to and then that node merges the node list with its own list.

9.5.5 Sorting

Unlike relational databases Cassandra has no capability of querying so you are not able to specify sorting when you retrieve data. Cassandra sorts the data when you store it by default. However, you need to think before storing data. There is an attribute (*CompareWith*) on the ColumnFamily, which can be used for sorting the data by data types. The options are:

- BytesType.
- UTF 8Type.
- LexicalUUIDType.
- TimeUUIDType.
- AsciiType.
- LongType.

Each of these types' treats the contents of your Columns' name as a different data type, for example, the LongType treats your Column name as a 64 Bit long value.



Here we can see an unsorted column family that are stored as two different data types (LongType, UTF8Type) in Cassandra. Longype is the natural ordering of numbers in comparison with UTF8Type with UTF8 String ordering.

```
<ColumnFamily CompareWith="LongType" Name="Persons" />
<ColumnFamily CompareWith="UTF8Type" Name="Persons" />
```

The sorting can be applied to *SuperColumns* too with the attribute *CompareSubcolumns-With*.

Chapter 10

Why Choose Neo4J?

10.1 Introduction

Social Network Systems, route planning systems or the World Wide Web itself is examples of graph data structures. In this chapter, there is information about some graph databases, the idea behind them and why we choose Neo4J. There are different terminologies for nodes and edges in different graph data storing systems and contexts:

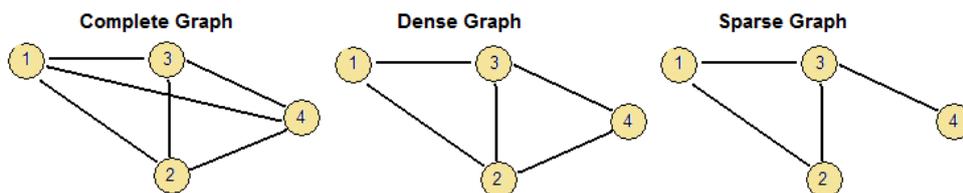
- Node and Edge in Neo4J graph database.
- MeshObjects and Relationships in InfoGrid database.
- Instances and Links in UML.
- Boxes and Arrows as we draw them on a white board.

In a relational database, we can have a table called "friends" with a field "personid" and another field "friendid" where friendid shows the friends of a person (personid). Both friendid and personid can be records in another table called "persons". The weakness with such solutions appear for many-to-many relationships. A person can have many friends, and you might want to track the date that, the friendship was created, or whether the two people are married. These solutions result in an increasing number of tables in databases, which are inefficient and complex for maintenance. Specially, in case of recursive joins, which tend to lead to long, incomprehensible SQL statements and unpredictable performance. A graph database is designed to prevent these types of problems and also to introduce a data model that is very close to the structure of relational data.

10.2 What Is A Graph?

Graph theory is the study of graphs in both mathematics and computer science. A graph contains of a set of nodes that are connected by edges. If the edges have no direction, we have an undirected graph. In contrast, a graph where the edges have a direction from a node to another is called a directed graph. Just like a relational database operations (for instance, CREATE, INSERT, SELECT and. etc.), there are many operations you can do on a graph database, for instance, graph traversing. Another grouping of graphs is:

- Complete Graph: When all nodes are connected to each other by edges the graph is called a *Complete Graph*.
- Dense Graph: When The number of edges that connect nodes are close to the maximum number of nodes the graph is called a *Dense Graph*.
- Sparse Graph: When the connected graph has only a few numbers of edges the graph is called a *Sparse Graph*.



There is two main data structure for representing a graph (a list or a matrix). List structures are often preferred for sparse graphs as they have smaller memory requirements. Matrix structures, on the other hand, provide faster access for some applications but can consume huge amounts of memory. There are different List structures (Incidence List, Adjacency List) and different matrix structures (Adjacency Matrix, Incidence Matrix, Kirchhoff Matrix, Distance Matrix). The two main data structures we are looking at, are:

- Adjacency List. An adjacency List is an array. Each node has one linked list that, contains the destination node of each edge, which is connected to it. Adjacency Lists are preferred for Sparse Graphs.
- Adjacency Matrix. An Adjacency Matrix is a boolean two-dimensional matrix (for instance, the $n \times n$ matrix A , where n is the number of vertices in the graph), in which the rows are the source vertices, and columns are the destination vertices.

Entries in the array indicate whether an edge exists between the vertices with boolean values. If there is an edge from some vertex x to some vertex y , then the element (x,y) is 1, otherwise it is 0. In computing, this matrix makes it easy to find sub graphs, and to reverse a directed graph.

The examples' below shows the Adjacency List for the Sparse Graph above and the Adjacency Matrix for the Dense Graph above.

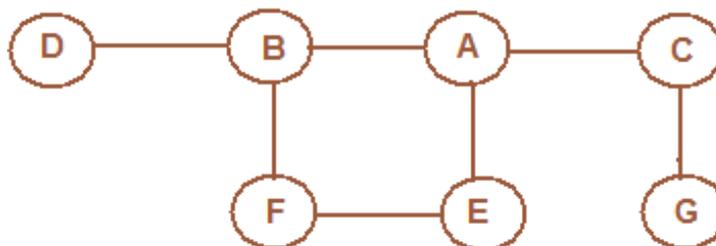
Adjacency List	Adjacency Matrix																									
1 ----> 2, 3	1																									
2 ----> 1, 3	2																									
3 ----> 1, 2, 4	3																									
4 ----> 3	4																									
	<table border="1" style="border-collapse: collapse; margin: 0 auto;"> <thead> <tr> <th style="padding: 2px 10px;"></th> <th style="padding: 2px 10px;">1</th> <th style="padding: 2px 10px;">2</th> <th style="padding: 2px 10px;">3</th> <th style="padding: 2px 10px;">4</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px 10px;">1</td> <td style="padding: 2px 10px;">0</td> <td style="padding: 2px 10px;">1</td> <td style="padding: 2px 10px;">1</td> <td style="padding: 2px 10px;">0</td> </tr> <tr> <td style="padding: 2px 10px;">2</td> <td style="padding: 2px 10px;">1</td> <td style="padding: 2px 10px;">0</td> <td style="padding: 2px 10px;">1</td> <td style="padding: 2px 10px;">0</td> </tr> <tr> <td style="padding: 2px 10px;">3</td> <td style="padding: 2px 10px;">1</td> <td style="padding: 2px 10px;">1</td> <td style="padding: 2px 10px;">0</td> <td style="padding: 2px 10px;">1</td> </tr> <tr> <td style="padding: 2px 10px;">4</td> <td style="padding: 2px 10px;">0</td> <td style="padding: 2px 10px;">0</td> <td style="padding: 2px 10px;">1</td> <td style="padding: 2px 10px;">0</td> </tr> </tbody> </table>		1	2	3	4	1	0	1	1	0	2	1	0	1	0	3	1	1	0	1	4	0	0	1	0
	1	2	3	4																						
1	0	1	1	0																						
2	1	0	1	0																						
3	1	1	0	1																						
4	0	0	1	0																						

The most used structure, for data storing in databases is the adjacency list.

10.2.1 Traversing A Graph

Traversing a graph is one the most basic operation on a graph database. Traversing refers to the problem of keeping track of which nodes we have visited and which node we don't have visited yet. There are different traversing techniques on a graph:

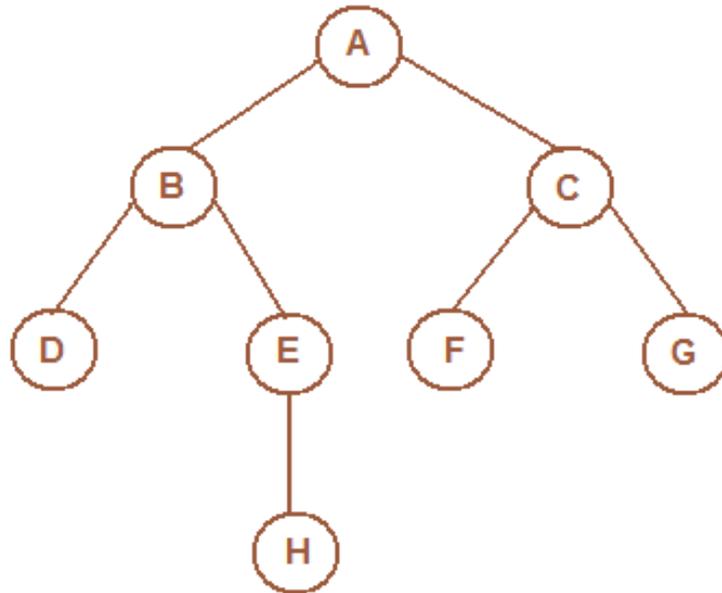
- Depth-first Search (DFS) is a technique for traversing a graph or a tree. DFS starts from the root, and then it visits the child nodes as far as there is a child node to visit before it goes back and visits the sibling nodes and do the same operation. DFS means that it traversing the depth before the breadth.



In the example above if we want to do a DFS search from node A from left, assuming that we remember the states of the visited nodes. The search will be in the

following order: A, B, D, F, E, C, G. If we don't remember the visited nodes, we'll get a loop as: A, B, D, F, E, A, B, D, F, E, A, B, etc.

- Breadth-first Search (BFS) is also another technique for traversing a graph. BFS visits the sibling nodes first before it visits the child nodes. That means BFS, for instance, begins at the root node and visits all the neighboring nodes and after that for each of those neighboring nodes, it visits their neighbors and so on.



In the example above if we want to do a BFS search from the root node (A) from the left assuming that we remember the states of the visited nodes. The search will be in the following order: A, B, C, D, E, F, G, H. If we don't remember the visited nodes, we'll get a loop as: A, B, A, B, etc.

10.3 Graph Databases - Analysis

The graph databases are optimized for highly related data with high performance of using graph traversal technique. The following is a list of some graph databases.

- InfoGrid, an open-source, commercial (AGPLv3) graph database, known as the *internet database* with many additional functionalities that make it to an ideal database for the development of fully functional web applications.
- Neo4J, an open-source, commercial (AGPLv3) graph database. Neo4J is an embedded, disk-based, fully transactional Java persistence engine with data struc-

ture as graphs. Neo4J provides different API's for Ruby, Python, and Java with support for various web technologies.

- DEX, a high performance graph database for huge amounts of data that is developed by DAMA-UPC (a research group of the Technical University of Catalonia). Dex is suitable to store huge amounts of data and when the high performance queries are mandatory for applications.
- HyperGraphDB, a distributed, embedded, open-source graph database that is designed specifically for *artificial intelligence* and *semantic* web projects.

Dex is not open-source but a restricted version is available only for personal use. HyperGraphDB is designed first for artificial intelligence. InfoGrid and Neo4J are very similar to each other with differences that InfoGrid is an internet-based graph-database and Neo4J that support the most programming languages, including Java. Some code differences between these two databases are illustrated below. The first one is a transaction-code and the other is a property-code.

Example 1. Transactions

Neo4j	InfoGrid
<pre>Transaction tx = neo.beginTx(); try { // do something tx.success(); } finally { tx.finish(); }</pre>	<pre>Transaction tx = null; try { tx = mb.createTransactionNow(); // do something } finally { if(tx != null) { tx.commitTransaction(); } }</pre>

Example 2. Properties

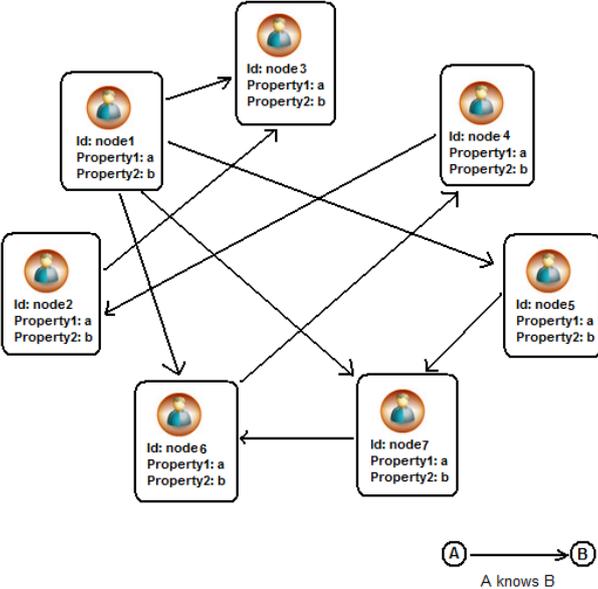
Neo4j	InfoGrid
<pre>firstNode.setProperty("Name", "Alberz");</pre>	<pre>firstObject.setPropertyValue(MySubjectArea.PERSON_NAME, StringValue.create("Alberz"));</pre>

There are not many differences between InfoGrid and Neo4J. The central objects of InfoGrid are a little bit unusual like *MeshObject*, *RelateAndBless*, while Neo4J has *Node*

and *Relationship* as its central objects. I think personally Neo4J is easier to code, read and understand because of its data model. As you can also see everything in Neo4J must happen inside a transaction, even if it is a graph traversal operation (this gives a very strong Isolation level). The InfoGrid traversal code seems to happen outside the transaction, so it sounds like it supports a more relaxed isolation level. In that situation, you can do for instance, the traversal operation inside a transaction if you like, but you are not required to. See the appendix A (Graph Databases - A Complete Comparison) for a complete comparison of the graph databases mentioned above.

10.4 Neo4J

The example below shows how a graph can be represented in Neo4J.



The basic operations on Nodes and Relationships are as follows:



The characteristics of Neo4J are:

- Disk-based database. Neo4J is a graph database engine with high writing capacity to disk instead of main memory because of the huge amount of data. Caching techniques provide even in memory data storage.
- Transactional. Neo4J has full support for:
 - *JTA*, Java Transaction API that allow distributed transactions across multiple resources in a Java environment.
 - *JTS*, Java Transaction Service that together with the Java Transaction API, enable to build distributed applications that are robust to all sorts of system and network failures.
 - *2PC* distributed ACID transactions: two-phase commit protocol is a distributed algorithm that coordinates all the processes that participate in a distributed transaction on whether to commit or abort (roll back) the transaction.
 - Configurable *isolation* levels: isolation is an ACID property that defines how/when the changes made by one operation become visible to other concurrent operations' properties.
 - *Transaction* recovery: the process to recover downtime and loss of good data.
- High performance and Scalable. In each JVM, you can save several billions of nodes and relations and properties because Neo4J has been written from scratch with performance and scalability in mind.
- Robust system. Neo4J had been in production for more than 2-3 years in a high demanding environment.
- Semi-structure. Neo4J supports the individualization of the data.

The best part when dealing with Neo4J is that as a developer, there's no need to think much about the representation of data. Nodes and edges (relationships) are represented in Neo4J, and you need only to think about to have all relations between nodes in place, and then you can focus on manipulation and less on how to represent it.

For example, "*find all nodes that node1 know directly*" in the picture above. We can see that node1 knows node3, node5, node6 and node7 directly.

node1 —→ *node3, node5, node6, node7*

This example can be coded as follows (<http://api.neo4j.org/current/>).

```
for (Relationship relation: node.getRelationships(RelationshipTypes.KNOWS, Direction.OUTGOING))
{
    Node otherNode = relation.getEndNode();
    //do something
}
```

Example A

There is three different direction we can look for:

- Direction.OUTGOING. For a directed graph (A KNOWS B), this direction will find the end-point of a directed edge (B).
- Direction.INCOMING. For a directed graph (A KNOWS B), this direction will find the start-point of a directed edge (A).
- Direction.BOTH. For an undirected graph (A — B), this direction will find both nodes (A, B) which mean, for instance, for Relationship "KNOWS", (A KNOWS B) and (B KNOWS A).

Another example is to *"find all nodes that are reachable from a node"*. This example requires an object called *Traverser*, who is described in the next section 10.4.1 (Traversal).

```
Traverser travers = node.traverse(Order.BREADTH_FIRST,
                                StopEvaluator.END_OF_GRAPH,
                                ReturnableEvaluator.ALL_BUT_START_NODE,
                                RelationshipTypes.KNOWS,
                                Direction.OUTGOING );

for(Node friendNode: travers)
{
    int depth = travers.currentPosition().depth();
    // do something
}
```

Example B

In this example, the traverser (Traverser) uses a breadth first algorithm that was described in the chapter 10.2.1 (Traversing A Graph). In that situation, we can traverse into the depth of the graph for each node. The traverser will not stop until the end of the graph and will visit all nodes. It will follow the "knows" relation in the outgoing direction. Finally, the loop fetches the result from the traverser.

10.4.1 Traversal

The two examples A and B, that were described in the previous section 10.4 (Neo4J), show how we get information out of Neo4J. According to database theory, Neo4J is categorized as a navigational database that means, you navigate from an arbitrary node(start node) via relationships to the nodes that match the specified search criteria. For that reason Neo4J provides a traversal API (a key to effective development by Neo4J) with an easy way of coding. The traversal object is called "Traverser" that is basically a Java Iterator. For example, for the Node "node", in the example B, the traverser has the following components:

- Traversal Order (BREADTH_FIRST). In that situation, the program, first returns friends and then friend's friend.
- The relationship as our search criteria (KNOWS).
- A stop criterion to know when we have to stop traversing (END_OF_GRAPH). In that case, searching in the whole data.
- A selection attribute to know which nodes to return as a consequence of the traversal (ALL_BUT_START_NODE that returns all nodes visited except the first node).
- Traversal direction (OUTGOING).

Part IV

Proof-Of-Concept

Chapter 11

Proof-Of-Concept Application

11.1 Development Method

During this phase, a prototype was designed and implemented to compare the database solutions Neo4J and Cassandra. There were many things to think about, for instance, which applications were needed, application installation, configurations and made the applications to work with each other. That was important to find out a development method that could help me to manage this phase in time.

Agile software development method refers to a group of development methodologies that are based on iterative development. Each iterative development is seen as a mini-project that runs through a period of time. I followed agile software development method because of the method flexibility. The entire implementation process was divided into the different mini-projects. Each mini-project was designed, developed and tested before the next mini-project started.

11.2 Prototype's Iterative Stages

The development stages (mini-projects) were:

- Choose software and tools.
- Installation, configuration, testing of software and tools and finally integration of the software and tools to each other.
- Designing a data model for Value4All in both Neo4J and Cassandra.
- Studying and designing comparison test-cases.
- Implementation was divided into different implementation phases.

- Comparison Analysis was also divided into different comparison phases.

11.3 Choosing Softwares and Tools

You can read about the whole process of choosing software and tools in chapter 6 (Development Environment - Solution Proposal).

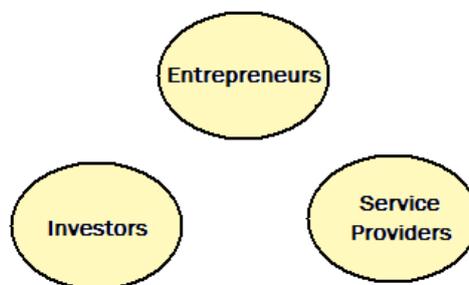
11.4 Softwares And Tools Installation and Configuration

You can read about all software and tool installations and configurations in the appendix F (Application Installations).

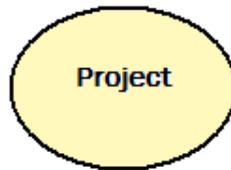
11.5 Designing A Data Model For Value4All In Cassandra and Neo4J

During this stage, a data model was designed to be used by the prototype. The first version of Value4All will contain three groups of people. All these groups of people have to be presented in Value4All and all of them will have different relations to each other. These groups of people are:

- *Entrepreneurs*
- *Investors*
- *Service Providers*



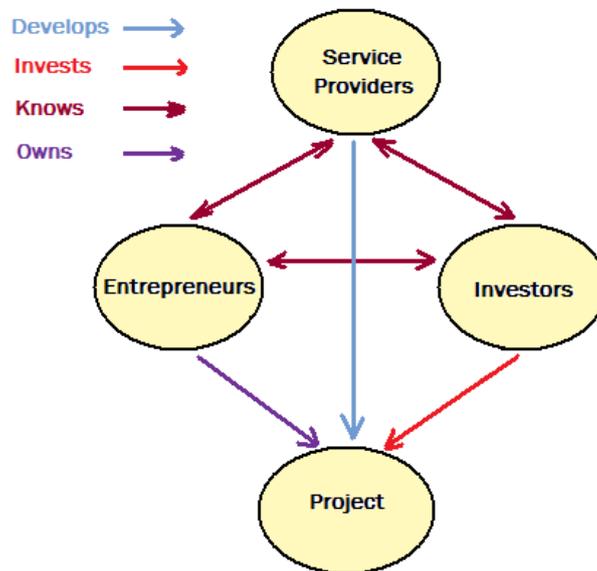
All these groups of people have something in common based on their interest in different projects.



The relations between these groups of people are as follows.

- **Owning:** a entrepreneurs *owns* a project.
- **Investing:** an investor *invests* in a project.
- **Developing:** a service provider *develops* in a project.
- **Knowing:** a person from one of these groups *knows* another one from other groups or in the same group.

A complete data model can be designed as follows.



A possible situation that can exist in Value4All is, for instance, "Kalle" is defined as an entrepreneur, "Hasse" as an investor, "Project A" as a project and Crisp as a service provider in Value4All. The relations can be defined as follows.

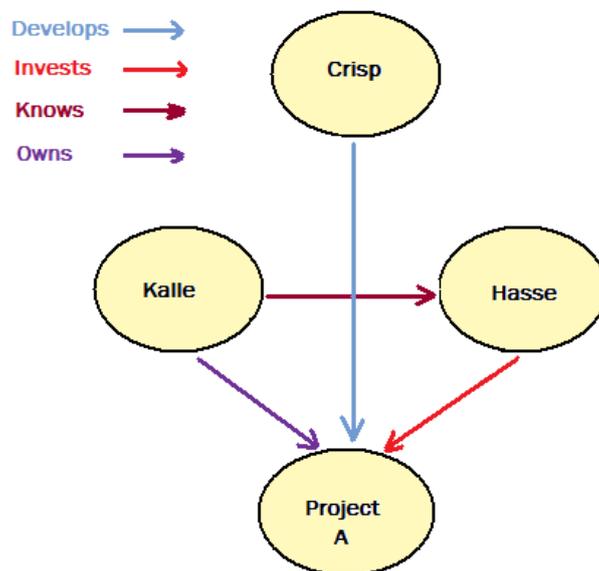
- "Kalle" *owns* Project A.
- "Hasse" *invests* in Project A.

- "Crisp" *develops* in Project A.
- "Kalle" *knows* Hasse.

With other words:

"Project A is owned by Kalle and Crisp develops it. Kalle knows Hasse, who invests in the project A."

The possible situation that, is derived from the general data model that was described above can be designed as follows in Neo4J:



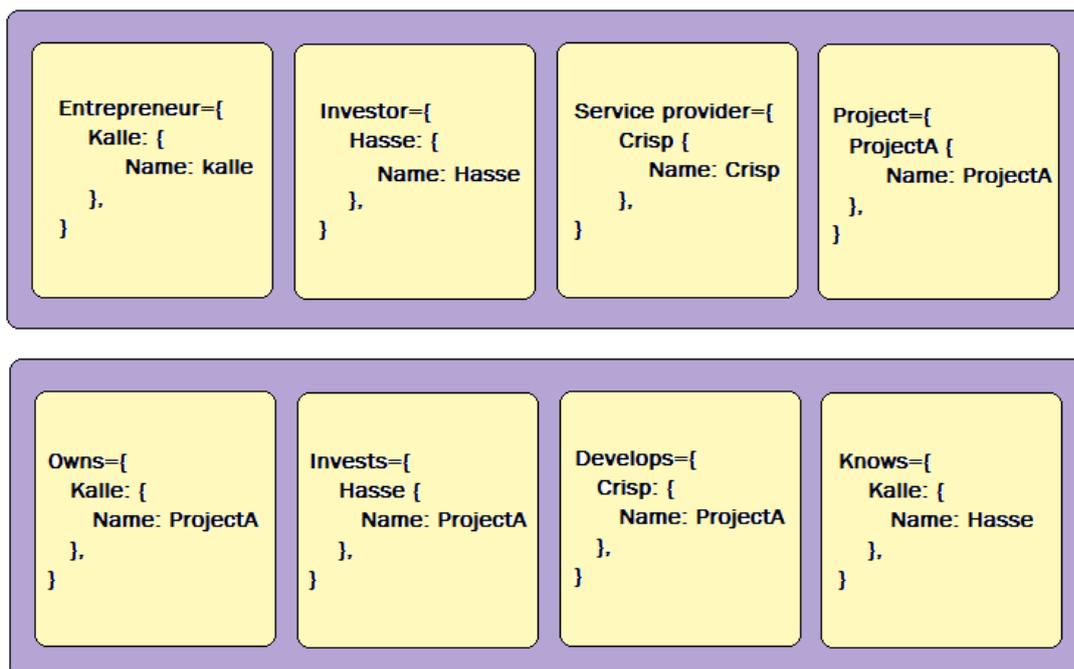
11.5.1 The Data Model In Cassandra

It is totally different to design the data model in Cassandra from Neo4J. There are other terms you talking about in Cassandra. In Value4All, there are three groups of people, one project group and four relationships. It can be designed as eight different Column Families. You can save an arbitrary number of columns in each column family.

- *Column Family: Entrepreneur.* Store information about all entrepreneurs existing in Value4All.
- *Column Family: Investor.* Store information about all investors existing in Value4All.
- *Column Family: Service Provider.* Store information about all service providers existing in Value4All.

- *Column Family: Project.* Store information about all projects existing in Value4All.
- *Column Family: Owning.* Store information about which entrepreneur owns which project.
- *Column Family: Investing.* Store information about which investor invests in which project.
- *Column Family: Developing.* Store information about which service provider develops in which project.
- *Column Family: Knowing.* Store information about which peoples from these groups knows each other.

The possible situation derived from the general data model that was described above can be designed as follows in Cassandra.



11.6 Prototype Implementation

It is two totally different implementation methods between Cassandra and Neo4J in terms of how to implement the data model and how to manage the establishment of communication and transactions between the business layer and the database layer.

11.6.1 Transactions in Neo4J

There are two rows of code, you need to implement for a successful transaction communication with Neo4J.

```
GraphDatabaseService graphDb =  
    new EmbeddedGraphDatabase( DB_PATH );
```

```
Transaction tx = graphDb.beginTx();
```

GraphDatabaseService is the main Neo4J object that acts as an access point to a running Neo4J instance from *DB_PATH*. With a *GraphDatabaseService* instance, you can create nodes, get nodes by an id and finally close the Neo4J instance.

```
graphDb.shutdown();
```

Transaction object is needed to transact the information between node objects and the Neo4J database instance. First, you have to create an instance of *Transaction* object. After that, you can perform all your operations based on your needs, for instance, search nodes. The method, *success()*, performs the transaction job, and it can results in *failure()* or *finish()* states:

```
Transaction tx = graphDb.beginTx();  
try {  
    ... //any operation based on your needs  
    tx.success();  
}  
finally  
{  
    tx.finish();  
}
```

As you can see in the code below, you first create your transaction object, implement your requirements, perform your transaction and finally close the transaction.

```

public class FindFriendNeo4jServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        System.out.println("Creating FindFriendNeo4jServlet.");

        String userName = request.getParameter("user_name");
        String knows = request.getParameter("knows");
        System.out.println("Group: " + knows);

        ArrayList result = new ArrayList();

        GraphDatabaseService graphDb = new EmbeddedGraphDatabase("var/graphdb");
        Transaction tx = graphDb.beginTx();

        IndexService index = new LuceneIndexService(graphDb);
        try {
            for (Node hit : index.getNodes("group", knows)) {
                String res = hit.getProperty("name").toString();
                System.out.println("Node name: " + res);
                result.add(res);
            }

            tx.success();
        } finally {
            tx.finish();
        }
    }
}

```

Searching

11.6.2 Transactions in Cassandra

This part is based on Cassandra-Hector API. Hector is a high level client (API) for accessing and managing information in Cassandra database. It is easy to create a Cassandra communication pool and transact the information to Cassandra database by Hector. The main Objects here are *CassandraClientPool* and *CassandraClient*:

```

CassandraClientPool pool = CassandraClientPoolFactory.INSTANCE.get();
CassandraClient client = pool.borrowClient("localhost", 9160);

```

A Cassandra client instance is retrieved from a *CassandraClientPool* object which can be instantiated by a Cassandra factory object (*CassandraClientPoolFactory*). First you retrieve a pool for your Cassandra clients for management of your Cassandra clients for reusing. You can then create Cassandra clients for the local machine on port 9160, where your Cassandra server is already listening and waiting for requests. After all operations done in Cassandra database based on your needs, you can finally release the Cassandra client to the pool with *pool.releaseClient(client)*:

```

try {
... //any operation based on your needs
}
finally
{
pool.releaseClient(client);
}

```

As you can see in the code below, you first create your transaction object, implement your requirements and finally close the transaction.

```

try{
    CassandraClientPool pool = CassandraClientPoolFactory.INSTANCE.get();
    CassandraClient client = pool.borrowClient("localhost", 9160);

    try{
        Keyspace keyspace = client.getKeySpace("value4all");

        // setup column parent (CF)
        ColumnParent cp = new ColumnParent(knows);
        // setup the slice predicate
        SlicePredicate sp = new SlicePredicate();
        SliceRange columnRange = new SliceRange();
        columnRange.setStart(new byte[0]);

        ...

    } finally {
        // return client to the pool. Finally block is used to make sure it's executed
        pool.releaseClient(client);
    }
} catch (Exception e) {}

```

Coding your functionalities

11.6.3 Data Model Implementation In Neo4J

There are two head objects of the data model implementation in Neo4J, the *Node* object and the *Relationship* object. Nodes are instantiated by:

```

Node kalle = graphDb.createNode();
Node hasse = graphDb.createNode();

```

Nodes above are created by invoking the method:

GraphDatabaseService.createNode()

The relationships can be defined between Nodes as follows:

```
Relationship relationship=kalle.createRelationshipTo(hasse, CLASS_NAME.KNOWS);
```

Relationship object defines a relationship between two nodes in the graph. The meaning of the code above is that "kalle knows hasse". "KNOWS" is the name of the Relationship instance. You can set properties to both objects (Nodes and Relationships) with methods:

```
Node.setProperty(...)
```

```
Relationship.setProperty(...)
```

A complete example is illustrated below. Descriptions are available with a number related to each part of the code.

- 1 Creating Nodes "user_name" and "project_name".
- 2 Creating Relationship "own_relationship".
- 3 Setting properties "name" and "group" with the values "userName" and "userGroup" to the node "user_name". Setting properties "pname" and "pgroup" with values "projectName" and "project" to the node "project_name".
- 4 Setting property "name" with value "own_relationship" to the relationship own_relationship.
- 5 Printing out the value of the property "name" of the node "user_name". Printing out the value of the property "pname" of the node "project_name".

```

private void saveDbData(String userName, String userGroup,
    String projectName, String investment, String knows, String servicing) {
    GraphDatabaseService graphDb = new EmbeddedGraphDatabase("var/graphdb");
    IndexService index = new LuceneIndexService(graphDb);
    Transaction tx = graphDb.beginTx();

    try {
        if ("intrepreneur".equals(userGroup)) {
            Node user_name = graphDb.createNode();
            Node project_name = graphDb.createNode();

            Relationship own_relationship = project_name.createRelationshipTo(user_name,
                OwningRelationshipType.OWNING);

            user_name.setProperty("name", userName);
            user_name.setProperty("group", userGroup);
            project_name.setProperty("pname", projectName);
            project_name.setProperty("pgroup", "project");

            own_relationship.setProperty("name", "own_relationship");

            System.out.print(user_name.getProperty("name"));
            System.out.print(project_name.getProperty("pname"));

            ...
        }
    }
}

```

11.6.4 Data Model Implementation In Cassandra

The important objects in Cassandra are Keyspace , Column Family, Column and finally key-value pairs of data as the minimal data structure in Cassandra. The highest level of Cassandra data model is the Keyspace object which holds Column Family objects inside itself. Column Family Objects hold all Column objects and Column Objects hold the key-value pair of data. Keyspace is retrieved as follows:

```
Keyspace keyspace = client.getKeyspace("Value4All");
```

Here, you get a Keyspace object from your CassandraClient Object with the name "Value4All" which can be defined in Cassandra storage configuration file (Cassandra config file). With a keyspace, you can read and write to your instance of Cassandra.

```
ColumnPath columnPath = new ColumnPath("investor");
columnPath.setColumn(bytes("column-name"));
keyspace.insert("key", columnPath, bytes("value"));
```

First of all, you create a Columnpath object which indicates where the column family "investor" is located. The column family "investor" must be defined in your Cassandra

storage (config) file. When you successfully have your column family, you can then create an arbitrary number of columns in the column family. You can also write an arbitrary number of key-value pairs of data to each column. For instance, you can first create a column with the name "HASSE" as follows.

```
columnPath.setColumn(bytes("HASSE"));
```

You can then insert an arbitrary number of key-value pairs of data in your column that is created in the specified column family ("investor"). For instance, a key-value pair ("name" : "hasse") can be created as follows.

```
keyspace.insert("name", columnPath, bytes("hasse"));
```

The example above can be illustrated as follows.

```
HASSE {  
"name" = "hasse"  
}
```

A complete example is illustrated below. Descriptions are available with a number related to each part of the code.

- 1 Retrieving the keyspace "Value4All" from Cassandra Client defined in the Cassandra config file.
- 2 Instantiating a reference to the column family "entrepreneur". Setting column name "userName" and then insert the key-value pair (name/userName) on it.
- 3 Instantiating a reference to the column family "project". Setting column name "projectName" and inserting the key-value pair (name/projectName) on it.
- 4 Instantiating a reference to the column family "owning".Setting column name "userName" and inserting the key-value pair (name/projectName).
- 5 Shows how to retrieve the columns with a key named "name" within different column families "entrepreneur", "project" and "owning".
- 6 Printing out the values of the key "name" in each columns.

```

try{
    Keyspace keyspace = client.getKeySpace("value4all"); 1
    if("intrepreneur".equals(userGroup)){
        ColumnPath entrepreneur = new ColumnPath("entrepreneur");
        entrepreneur.setColumn(bytes(userName));
        keyspace.insert("name", entrepreneur, bytes(userName)); 2

        ColumnPath project = new ColumnPath("project");
        project.setColumn(bytes(projectName));
        keyspace.insert("name", project, bytes(projectName)); 3

        ColumnPath owning = new ColumnPath("owning");
        owning.setColumn(bytes(userName));
        keyspace.insert("name", owning, bytes(projectName)); 4

        // read
        Column col1 = keyspace.getColumn("name", entrepreneur);
        Column col4 = keyspace.getColumn("name", project);
        Column col8 = keyspace.getColumn("name", owning); 5

        System.out.println("entrepreneur name: " + string(col1.getValue()));
        System.out.println("project name: " + string(col4.getValue()));
        System.out.println(string(col8.getName()) + " owns: " + string(col8.getValue())); 6

        ...
    }
}

```

11.7 Comparison Test Cases

The test cases were based on some properties, which were compared the Cassandra as a key-value store and Neo4J as a graph database:

- Changeability.
- Easy-to-use.
- Performance.
- Scalability.

Easy-to-use property covered all development stages and were compared based on how easy it was to write code and understand the data model throughout all levels of the development.

11.7.1 Changeability

For example, if we have an object with some attributes, and then we realize that we need an additional attribute to add to our object. How difficult it is to change the data model and how much you need to change in your implementation. As mentioned in

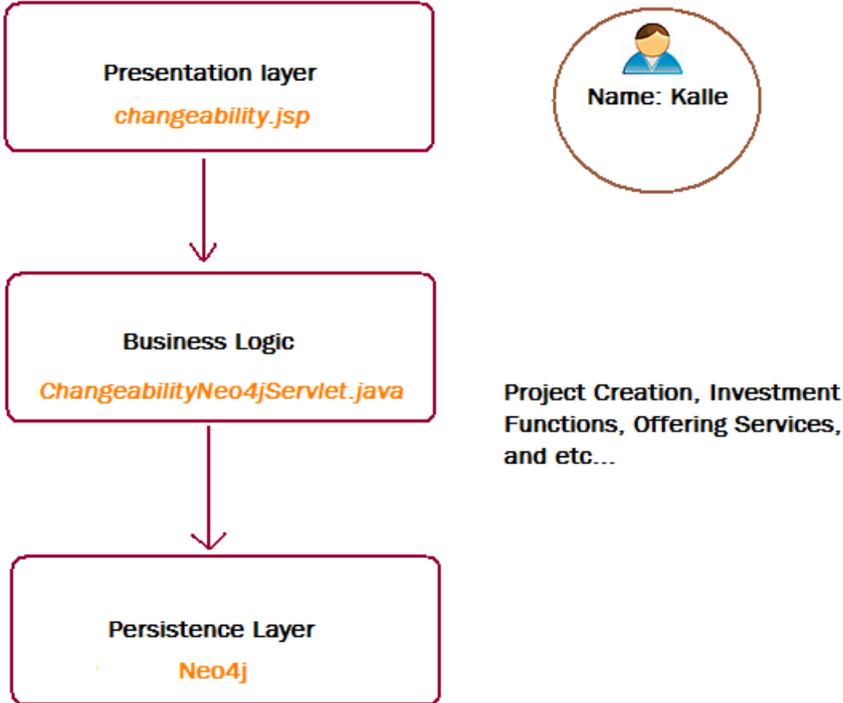
the chapter 4.5 (Value4All Three Tier Design), Value4All is a three tier architecture with separated layers:

- Presentation Layer
- Business Logic
- Persistence Layer

This separation makes the changeability of the implementation easier since you need to change your code, based on your needs in different layer.

Neo4J - Changeability

The prototype was developed for Neo4J based on a three tier architecture, that are illustrated below.



"Changeability.jsp" is the main page for the presentation layer for Neo4J database. Here, you can create a Value4All user and decide if he is an entrepreneur, an investor or a service provider. The user can also decide what to do, for instance, if they want to create a project, invest in a project or create a friendship relation with somebody within different groups existing in Value4All. The data are sent to the business layer for manipulation, for instance, in "ChangeabilityNeo4jServlet.java", you can create a project, investment in a project and, etc. The last layer is the persistence layer, and here the

data is stored in Neo4J. Checking the *changeability* power of Neo4J, I added another property to objects, for instance, a "group" property which indicates which user group (Investor, Entrepreneur, Service provider) a user belongs to.



A database index is used to access the data stored in a database which also improves the speed of data access. Neo4J core database has no built-in indexing system, but it provides a pluggable mechanism to support it. The indexing is done per Node's property. For instance, if you want to perform the query "*find all users who have name = hasse*", then you have to create an index for the property "*name*" for each Node you create. First you have to import all needed libraries and install them to make indexing to work, see the appendix F (Application Installations). Using indexing features is very easy in Neo4J. You have to declare an indexing service (*IndexService*) one time at the beginning of your code:

```
IndexService indexService = new LuceneIndexService( graphDb );
```

Now you can create an index for each property of your Nodes. First we create a Node "user_name" with a "*name*" property for our user "hasse" and then we create an index for the Node "user_name" and property "*name*":

```
Node user_name = graphDb.createNode();  
user_name.setProperty("name", "hasse");  
indexService.index(user_name, "name", user_name.getProperty("name"));
```

The problem is, when you design a new property for your Nodes (for instance, the user group that each user belongs to), then you have to create a script to modify all existing nodes for the new property. The best choice is to think about all properties from the beginning and create so many properties as possible to skip this kind of modifications. Here were the changes that, I made to the prototype when I added an additional property "*user_group*" (Investor, Entrepreneur, Service provider):

- Code modification. Try to have a *single-point-of-changes* in your application. In my case, I had only the class "ChangeabilityNeo4JServlet.java" to modify.

```
Node user_name = graphDb.createNode();
user_name.setProperty("name", "hasse");
user_name.setProperty("group", "Investor");
index.index(user_name, "name", user_name.getProperty("name"));
index.index(user_name, "group", user_name.getProperty("group"));
```

Now you can do more advanced queries to your database, for example, *"find all users with name hasse which belongs to user group investor."*

- Script. I had to create a script to made changes to my already existing Nodes. All Nodes before the creation of the additional property have to be changed and also a property index had to be defined for each node to make nodes accessible through the new query functionality. But scripting was easy because you don't need to go to your database, modify tables or something like that. For lookup, you need to loop through your Nodes based on some properties. See the chapter 10.4 (Neo4J) for more information about traversing in Neo4J.

There are other features available for Neo4J lookup, for instance, TimeLine object. You can create a TimeLine object and add nodes to it, each with a timestamp, and you can then ask Neo4J to return, for instance, all nodes within a specific period of time.

Cassandra - Changeability

Similar to Neo4J a three layer application was created to measure the changeability of the Cassandra database. An additional property (age) was added to Cassandra objects.



As mentioned in the chapter 11.5.2 (The Data Model In Cassandra), the definition of

the data model is stored in the storage-config of the Cassandra database. This file is created with the needed configurations and are loaded when the Cassandra database starts before you start the application server. It is very important to think about the application's data model from the beginning to avoid changes to this file since this file can become corrupt if you don't know what you are doing. Here you can see the configuration of Column Family for Value4All in the storage-config:

```
<Keyspace Name="value4all">
<ColumnFamily CompareWith="UTF8Type" Name="entrepreneur"/>
<ColumnFamily CompareWith="UTF8Type" Name="investor"/>
<ColumnFamily CompareWith="UTF8Type" Name="serviceProvider"/>
<ColumnFamily CompareWith="UTF8Type" Name="project"/>
<ColumnFamily CompareWith="UTF8Type" Name="owning"/>
<ColumnFamily CompareWith="UTF8Type" Name="investing"/>
<ColumnFamily CompareWith="UTF8Type" Name="knowing"/>
...
</Keyspace>
```

The keyspace ("*value4all*") is the highest level of the Cassandra data model. Within the keyspace "value4all", there are three Column Families (entrepreneur, investor, service-Provider) for the three group of users (Entrepreneur, Investor and Service Provider). There is a Column Family (project) defined for all projects created within the application keyspace "value4all". There are four another Column Families (owning, investing, knowing, developing) for the respective relationships. Within each Column Family, you can then create your (key-value) pairs of data in different columns. As mentioned in the previous chapter 11.6.4 (Data Model Implementation In Cassandra), the creation of a key-value pair of data is as follows:

```
ColumnPath entrepreneur = new ColumnPath("entrepreneur");
entrepreneur.setColumn(bytes(HASSE));
keyspace.insert("name", entrepreneur, bytes("hasse"));
```

Here, you first define a path to your Column Family "entrepreneur" and then you create a property with (key-value) pair = (name, hasse) to add to the column "HASSE". The example above, can be illustrated as follows.

```
HASSE {
```

```
"name": "hasse"  
}
```

Adding another property to a Column Family required the following changes:

- **Code Modifications.** Adding a new property to an existing column within a Column Family is easy:

```
ColumnPath entrepreneur = new ColumnPath("entrepreneur");  
entrepreneur.setColumn(bytes(HASSE));  
keyspace.insert("name", entrepreneur, bytes("hasse"));  
keyspace.insert("age", entrepreneur, bytes(30));
```

Here we define an age property to the column "HASSE" within the column family "entrepreneur" with the name "age" and value "30". The example above, can be illustrated as follows.

```
HASSE {  
  "name": "hasse"  
  "age": "30"  
}
```

- **Scripts.** The age changes are only applied to the new users and therefore, we need a script to look up all users, and then we can set the age for all of them. These changes are done, if we want to be able to look for our objects based on the "age" property. Lookup in Cassandra is as follows:

```
Keyspace keyspace = client.getKeySpace("value4all"); (1)  
ColumnParent cp = new ColumnParent("entrepreneur"); (2)  
SlicePredicate sp = new SlicePredicate(); (3)  
SliceRange columnRange = new SliceRange(); (4)  
columnRange.setStart(new byte[0]); (5)  
columnRange.setFinish(new byte[0]); (6)  
columnRange.setReversed(false); (7)  
sp.setSlice_range(columnRange); (8)  
KeyRange keyRange = new KeyRange(500); (9)  
Map<String, List<Column> map = keyspace.getRangeSlices(cp, sp,  
keyRange); (10)  
for (String key : map.keySet()) { (11)
```

```

List<Column> columns = map.get(key); (12)
for (Column column : columns) { (13)
Now you can do something with your columns
}
}

```

Description is as follow:

- (1) As I mentioned earlier the keyspace is the highest level of the data model of the Cassandra database. With keyspace, you can handle all read/write operations to Cassandra.
- (2) For searching, you can use ColumnParent, which means that, you can select groups of columns from the ColumnFamily ("entrepreneur"). With other words, we get details from the column family "entrepreneur".
- (3) A SlicePredicate is described as a property that the elements of a set have in common. In this case, all users within "entrepreneur" column family has a "name" property which is common between them.
- (4) With a SliceRange object, you can stores basic range, ordering and limit information for a query that will return multiple columns. It could be thought of as Cassandra's version of LIMIT (limit your search) and ORDER BY in relational databases.
- (5) The column to start the slice with. No parameter is defined here.
- (6) The column to stop the slice at. No parameter is defined here.
- (4-6) These settings will get all Keys in the Keyspace for the column family "entrepreneur".
- (7) It describes the order of the retrieved data. No reversed ordering of data.
- (8) Apply the SliceRange setting to the SlicePredicate Object. In this case, we tell the SlicePredicate to get all nodes in the column family "entrepreneur".
- (9) Set the max number of retrieving values that are 500 here.
- (10) Retrieve the column result in a Map.
- (11-13) Looping through all columns do something with them. For instance, adding the new property "age". For more information, see the previous point.

- Adding a new column to an existing column family "entrepreneur" is also easy. You can add an arbitrary number of columns to your column family. For instance, adding the column "KALLE" to your objects.

```
ColumnPath entrepreneur = new ColumnPath("entrepreneur");
entrepreneur.setColumn(bytes(HASSE));
keyspace.insert("name", entrepreneur, bytes("hasse"));
keyspace.insert("age", entrepreneur, bytes("30"));
entrepreneur.setColumn(bytes(KALLE));
keyspace.insert("name", entrepreneur, bytes("kalle"));
keyspace.insert("age", entrepreneur, bytes("45"));
```

The example above can be illustrated as follows.

```
HASSE {
  "name" : "hasse"
  "age" : "30"
}
KALLE {
  "name" : "kalle"
  "age" : "45"
}
```

Another test was to add another Column Family (developing) to the keyspace. The changes were done first in the storage-config file of the Cassandra database (*this is highly recommended that you back up the storage-config of the Cassandra database first*). For that purpose, I *stopped* the Cassandra database, added the following row to the storage-config file of the Cassandra database:

```
<Keyspace Name="value4all">
<ColumnFamily CompareWith="UTF8Type" Name="entrepreneur"/>
<ColumnFamily CompareWith="UTF8Type" Name="investor"/>
<ColumnFamily CompareWith="UTF8Type" Name="serviceProvider"/>
<ColumnFamily CompareWith="UTF8Type" Name="project"/>
<ColumnFamily CompareWith="UTF8Type" Name="owning"/>
<ColumnFamily CompareWith="UTF8Type" Name="investing"/>
<ColumnFamily CompareWith="UTF8Type" Name="knowing"/>
<ColumnFamily CompareWith="UTF8Type" Name="developing"/>
```

...

</Keyspace>

After that you can start the Cassandra database and perform your code modification as you want for new and old nodes.

Cassandra VS Neo4J- Changeability

I think the changeability of Neo4J is more flexible and faster than Cassandra for both writing and reading parts. Indexing is separated from Neo4J core and because of that, if you want to add a new property to your nodes, you have to define a new index for the new property, so you can search for nodes based on the new property. I think this is not a big problem to have been indexing separated from Neo4J core. The head problem here is to integrate indexing to the application which is difficult.

The structure of a social network's data is more easier to be designed in Neo4J. You have people or groups as your data with different properties, which are connected to each other by different relations. This is not obvious in Cassandra. Changing Cassandra's storage-config file, when you add a new Column Family was another reason, to skip by using Neo4J. Changing in Cassandra config file reminds a little about relational databases schema and tables. Reading from Cassandra is harder to understand than Neo4J because of the Cassandra's data model complexity.

11.7.2 Easy-To-Use

This property is an important factor of choosing a database. For example, how much work we have to do if we want to change something, how much code we need to write for a functionality in database layer, and, etc. The categories of comparing the easy-to-use property of Neo4J and Cassandra were as follows.

- Data model.
- Reading/Writing, from/to databases.
- Start/Shutdown of databases.
- Installation.

Choosing one of these databases for each category were done as follows. Within each category, the best database got an a plus (+) sign. The database with the highest number of plus sign was the winner database.

Data Model

The data model implementation of Neo4J is based on *Nodes* and *Relationship* with an arbitrary number of properties (key-value pairs) for both. The Data model implementation of Cassandra is separated in different levels (Keyspace, Column Family, Column, Key-Value pair). The data model of Cassandra is very powerful, and it helps to write very powerful queries. During the implementation phase, I had to concentrate on what these different objects were. On the other hand, the way of thinking in the Neo4J data model was new to me but very easy to understand, which helped me to concentrate on coding features instead of to think about what I am doing.

Criteria	Neo4J	Cassandra
Easy to understand the data model	+	
Easy to code the data model	+	
Result	2	0

Reading And Writing

Reading and Writing from/to Neo4J and Cassandra were discussed in previous chapter (*Changeability*). I think the writing to both is easy but reading from Cassandra is harder. Documentation of Neo4J was better, which helped me to understand what I am doing during the implementation phase.

Result: A plus sign for both reading and writing from/to Neo4J. A plus sign for writing to Cassandra. A plus sign for Neo4J documentation.

Database Start/Shutdown

Neo4J is an embedded persistence engine, which means that it's a small Java library that is easy to include in your development environment. Because Neo4J is embedded there is no need to start and shutdown Neo4J separately from our application. Cassandra had to be installed separately first and be integrated to the application (Value4All) by a client (Hector in my case), and then it had to be started (and shutdown) separately.

Criteria	Neo4J	Cassandra
Easy to Start and Shutdown	+	
Stand alone		+
Result	1	1

The positive thing with a stand alone database like Cassandra is, if your application goes down, your database is not affected. On the other hand, Neo4J database is embedded, and it is up and running when your application is up and running. It doesn't matter, in this case, to have the Neo4J down since it starts up when the application starts up.

Installation

The installation process of Neo4J and Cassandra are described in the appendix F (Application Installations). Installation of both Neo4J and Cassandra were easy. If you have worked with your development environment before, then it is easy to integrate Cassandra and Neo4J in your environment. Java API code functionalities follow with Neo4J core, and you can code directly when you had successfully integrated Neo4J in your environment, but it is harder to integrate *hector* (Cassandra client) in your environment. Hector is a fully developed *jar-module* and you need to know about module integration in your environment, for instance, by maven in my case, which is a hard task. Indexing is not supported in Neo4J core and because of that you have to integrate the API dependencies in your environment. I think Neo4J had very good documentation and Wiki pages related to it. You can find the documentation here:

<http://neo4j.org/>

Cassandra database had good documentation too.

<http://cassandra.apache.org/>

But I think Cassandra client "*Hector*" had not enough documentation but hector is one of the best choices of a high level Cassandra client. You can read about it here:

<http://prettyprint.me/2010/02/23/hector-a-java-cassandra-client/>

Criteria	Neo4J	Cassandra
Database Installation	+	+
Indexing feature		+
API installation		+
Documentation	+	
Result	2	3

Result

The comparison Result in this chapter is based on the result of the winner of each part of the above comparisons. For the start/shutdown, I think, there is no problem if Neo4J goes down when the application goes down (in my case). On the other hand, Cassandra is up and running even though the application is down.

Criteria	Neo4J	Cassandra
Data model	Winner	
Reading/Writing from/to	Winner	
Start/Shutdown		
Installation		Winner
Result	Winner	

11.7.3 Performance

Both Cassandra and Neo4J offer good performance. Comparisons in this chapter were done both by studies from the internet and also analysis done by a simple test by the prototype. I didn't have the best environment (hardware) to compare the power of reading and writing of these two powerful databases and because of that I had done some researches, and the results are also applied here. My two phase tests were about to add 1000 nodes to Cassandra and Neo4J (measure the writing time) and then reading from them (measure the reading time). During the phase two test, adding 10.000 nodes and perform the same actions as for 1000 nodes (measuring the read and write times). The results of these two phases were compared to each other for both Cassandra and Neo4J.

Neo4J

Neo4J is a high-performance graph engine with all the features of a robust database. While Neo4J is a relatively new open source project, it has been used in production applications with over 100 million nodes, relationships and properties, satisfying enterprise robustness and performance requirements. The properties that increase the performance of Neo4J are among other things:

- *Smart caching.* The first thing a client thinks about while visiting a site is, how much time it takes to see a page *irrespective* of what, they are looking for. In a three tier architecture, while the presentation layer shows information imported from persistence layer, you can cache the information (for example, such caching can be based on the popularity of the information and all newly information) for fast accessing. This technique gives high availability and high performance to the application.
- *Powerful traversals capability.* Traverse queries are done across several hundreds of thousands of relationships in depth higher than 100 000 in a matter of seconds, on fairly modest hardware.

In my case, the test of reading and writing time of Neo4J in phase one, by adding 1000 nodes to Neo4J and checking the time were as follows.

- Writing time in ms: 248 (ms).
- Reading time in ms: 932 (ms).

Phase two, I added 10.000 nodes to Neo4J and checked the time for reading and writing.

- Writing time in ms: 650 (ms).
- Reading time in ms: 1989 (ms).

```
Run Neo4jPerformanceTest
"C:\Program Files\Java\jdk1.6.0_12\bin\java" -Didea.launcher.port=7
*****
Start Neo4jPerformanceTest.
Writing 1000 nodes to Neo4J took: 248 ms.
Node name: 0
Node name: 1
...
Node name: 997
Node name: 998
Node name: 999
Reading 1000 nodes to Neo4J took: 932 ms.
Writing 10.000 nodes to Neo4J took: 650 ms.
Node name: 0
Node name: 1
...
Node name: 9997
Node name: 9998
Node name: 9999
Reading 10.000 nodes to Neo4J took: 1989 ms.
Ending Neo4jPerformanceTest.
*****
```

Neo4J has high performance in case of deep reading. Write speed is much dependent on the seek time of the file system and hardware. The Ext3 file system and SSD disks are a good combination and result in transactional write speeds of approximately 100,000 operations per second. Here you can read a performance guide for Neo4J:

http://wiki.neo4j.org/content/Neo4j_Performance_Guide

Cassandra

Cassandra offers very high performance. Its performance has been growing nicely in each point release. Cassandra is much faster than relational databases but slower than memory-only systems or systems that don't sync each update to disk. It's designed to use disk for durability and to accommodate using large sets of data, letting the operative system use memory as a huge cache for all data is impossible yet. On the other hand, Cassandra provides smart caching and because of that, for instance, the newest data or the most popular data can be cached in the memory. Other properties which increase the power of Cassandra are:

- Range queries: unlike most key-value stores, you can query for ordered ranges of keys.

- Powerful data model
- Distributed writes: you can read and write any data to anywhere in the cluster at any time. There is never any single point of failure.

In my case, That was hard to find out the true performance of Cassandra since, Cassandra is designed for multi-nodes, but I only used my laptop as a Cassandra instance (node). In my tests, I didn't change anything in Cassandra config file, for instance shutdown the logger or something else. Here is the result of my tests. In phase one, I added 1000 objects to Cassandra and checked the time for reading and writing.

- Writing time in ms: 398 (ms).
- Reading time in ms: 67 (ms).

Phase two, I added 10.000 rows to Cassandra and checked the time for reading and writing.

- Writing time in ms: 1944 (ms).
- Reading time in ms: 125 (ms).

```

Run CassandraPerformanceTest
"C:\Program Files\Java\jdk1.6.0_12\bin\java" -Didea.launcher.port=7538 "-Didea.launcher.bin.path=C:\Program Files\JetBrai
*****
Start CassandraPerformanceTest.
[INFO] service.JmxMonitor Registering JMX me.prettyprint.cassandra.service:ServiceType=hector,MonitorType=hector
[INFO] service.CassandraClientPoolImpl Creating a CassandraClientPool
Writing 1000 nodes to Cassandra took: 398 ms.
objectName1
name: 0
name: 1
...
name: 997
name: 998
name: 999
Reading 1000 nodes to Cassandra took: 67 ms.
Writing 10.000 nodes to Cassandra took: 1944 ms.
objectName2
name: 0
name: 1
...
name: 9998
name: 9999
Reading 10.000 nodes to Cassandra took: 125 ms.
Ending CassandraPerformanceTest.
*****

```

Another analysis shows that Cassandra 0.5.0 has been benchmarked to insert about 10000 rows per second on a four core server with 2GB of RAM. Here you can read about it:

<http://spyced.blogspot.com/2010/01/cassandra-05.html>

Here you can read another performance study for version 0.5 made by yahoo.

<http://www.brianfrankcooper.net/pubs/yycsb-v4.pdf>

Result And Analysis

The comparison Result in this chapter is based on the results of the two phase performance tests that, were done for both Cassandra and Neo4J.

Criteria	Neo4J	Cassandra
Reading (1000 nodes)	932 ms	67 ms
Writing (1000 nodes)	248 ms	398 ms
Reading (10.000 nodes)	1989 ms	125 ms
Writing (10.000 nodes)	650 ms	1944 ms
Result		

From the comparison table above you can see that writing is faster in Neo4J than Cassandra but reading is much slower in Neo4J than Cassandra.

The question was why?

The problem maybe lies in my coding. For instance:

```
for(int i=0; i<1000;i++){
for (Node hit1 : index1.getNodes("objectName1",
                                new Integer(i).toString())) {

String res = hit1.getProperty("objectName1").toString();
System.out.println("Node name: " + res);
}
}
```

From the code above, you can see that: *for each node, get the node(s) by name*. The problem was in the first loop because I had to look through all nodes existing in Neo4J with property "objectName1", one time for each node value. On the other hand, an in-depth lookup is very fast, for instance "*get all nodes that are friends with hasse*". It was hard to say which of these databases wins the performance tests. I think, Neo4J is good enough and can be compared with Cassandra as a high performance database based on the results from the performance tests and also researches that show the high performance of Neo4J. For the performance tests, the both of these databases are winners.

11.7.4 Scalability

One of the most important properties of a social network system is the scalability of the application. Having huge investments in time and other resources into the application, nothing is better to see the application is growing but on the other hand, the bad thing is to see the application collapse because of the load. The ability to grow based on the needs for Value4All is critical. Value4All, in the future, has to provide the exponential growth of the volume of data generated by users and the application itself. Value4All may also increase in the number of servers in the features because of, for example, increasing of interdependency and complexity of data, increasing use of the internet, Web2.0, other competitors in the market and, etc.

Neo4J

From the beginning, Neo4J is designed for large network of data, 100+ millions of nodes, relationships and properties (*Key-Value pairs*) for both nodes and relations on a single JVM (*Vertical Scaling*). If you hit this limit you have to start partitioning your data. Increasing of the number of data creates a need for multiple machines (*Horizontal Scaling or Sharding*). Neo4J *can be sharded* to scale out across multiple machines by additional configurations. Here you can see a suggestion for sharding:

<http://lists.neo4j.org/pipermail/user/2009-January/000997.html>

Sharding is very important when the Value4All gets bigger and the need for to-be-up and running are critical. Since Neo4J provides only running on one JVM, there is a risk for single-point-of-failure who has to be avoided. Maybe, in the future, a sharding possibility will be provided by Neo4J and until that, Neo4J can be used as a good first step database solution for Value4All.

Cassandra

Most of the database systems in the market are based on transactions to guarantee the integrity of data, which also ensures the consistency of data. Recall from chapter 8.1 (Market's Database Solutions), these transactional properties are known as *ACID*. However, scaling and ACID are tradeoffs, which means that, it is hard to scaling out and having ACID properties at the same time. Recall from chapter 8.1 (Relational Databases - Scalability Problem), according to CAP theorem about the scalability, only two of the three different aspects of the theorem can be achieved fully at the same time (Consistency, Availability, Partition Tolerance). Cassandra is a scalable, eventually consistent (guarantee AP of CAP theorem), running on Java with no single point-of-failure. Cassandra provides both *vertical scalability* and *horizontal scalability*. Cassandra can store hundred millions of data and can scale out vertically very fast. Huge amount of data can be queried because of the power of the data model and increasing the data can be solved by scaling out horizontally with powerful load balancing across servers and avoiding bottlenecks. With other words, Cassandra offers:

- **True scalability:** it scales horizontally. To add more capacities to a cluster or turn on another machine, you don't have to restart any processes, change your application queries, or manually relocate any data.
- **Multi-datacenter awareness:** you can adapt your node layout to ensure that if one data center burns in a fire, an alternative data center will have at least one full copy of every record.

Result And Analysis

The comparison Result in this chapter is based on the results of the studies for vertical/horizontal scalability of Cassandra and Neo4J.

Criteria	Neo4J	Cassandra
Vertical Scalability	+	+
Horizontal Scalability		+
Result		Winner

Neo4J has no support for horizontal scaling, but on the other hand, you can run billions of nodes and relations with arbitrary number properties on them in a single JVM.

If you come up to this level of data, you can then partition your data to other storages (in Value4All case, this will not happen soon). There is still a risk for single-point-of-failure. By additional solutions, you can have a success with horizontal scaling of Neo4J. I think there are no differences between Neo4J horizontal scaling problem and relational databases scaling problem, On the other hand, Cassandra, provides horizontal scaling by its huge-cluster solution. Cassandra is very powerful and offer solutions with high performance and scalability opportunities.

11.7.5 Conclusion

It was hard to say which database was the best choice for Value4All. I choose Neo4J as the database solution:

- Neo4J offered more flexible results for the changeability property. This property is very resource consuming. This property was the most important part of the comparison test cases.
- Neo4J had the most natural choice of the data model implementation of Value4All's data as a social network system. Furthermore, for the Easy-To-Use property, the best choice was also Neo4J database.
- Scalability was an important part of test cases. Neo4J didn't offer horizontal scaling by default as Cassandra, but it is possible to success with some configurations. On the other hand, you can run hundred billions of nodes, relationships with the arbitrary number of key-value pairs of property on both of nodes and relations on only one single JVM. I think, even this part was satisfied by Neo4J, at least, as the first version database solution for Valu4All since the amount of data in Value4All will not pass this level of data quantity soon. In the future, there will be a "need" for migration from Neo4J if there is no solution for horizontal scalability by Neo4J. Cassandra is designed for a cluster of servers which is not the first configuration case for Value4All in the first stages. These configurations are also expensive, complex and require professional skills.
- Neo4J had the best writing time in performance test cases for both 1000 nodes and 10.000 nodes. Neo4J provides high-performance for reading in depth levels. In my test cases, I get very bad results for reading part (not in-depth reading, which is very fast) for Neo4J compared with Cassandra. I think, it can depend on my bad coding knowledge. Neo4J is still good enough and can be compared with Cassandra as a high performance database based on the performance test results and results from the researches.

Cassandra database is a very *powerful* database because of its data model, providing horizontal and vertical scalability by default and high performance. Cassandra is also a good choice for Value4All, but it is designed for a cluster of machines, and you cannot get the best power of Cassandra by running it on a single machine (probably what we need in the first version of Value4All). On the other hand, Neo4J is easy to use, provides good changeability property with high performance and good vertical scalability and because of that Neo4J is enough as a winner database for the first version of Value4All. In the future, it is possible to have a migration of data from Neo4J, in the case that there is no solution available as default for sharding in Neo4J.

Part V

Future Of Value4All & Final Words

Chapter 12

Which Social Networks Can be Integrated In Value4All?

There are many active social network systems on the internet. The most popular social network systems in Sweden are Facebook, LinkedIn and Twitter. The purpose of this chapter was to make some research to see if there were any possibilities to Integrate Value4All with these platforms. For that purpose, I looked for the possibilities of login functionality of Value4All to be integrated with these popular social network systems. I found out that there are good possibilities for this technique. Login to Value4All through Twitter, Facebook or LinkedIn gives you:

- High security sign-in (trusted user authentication).
- A Value4All user can access all its personal details from LinkedIn, Facebook or Twitter and the information details can be imported to Value4All directly.
- Speed up the login process (Value4All doesn't need to create own login process).
- It makes Value4All more attractive and valuable.

12.1 Facebook

A good possibility with Facebook is that, a site (like Value4All) doesn't need own login process. By Facebook's account, you can log in directly to your system, for example Value4All. With an account in Facebook, you can import all your information to Value4All and stay logged in as long you are logged in to Facebook. Facebook calls it for "*Single Sing-on or Facebook Connect*". Facebook authorization technique is "*OAuth 2.0 protocol*".



Here you can read about this technique:

<http://wiki.oauth.net/OAuth-WRAP>

Facebook has a great information page about how you can connect your site to Facebook.

<http://developers.facebook.com/docs/guides/web>

12.2 Twitter

Twitter uses the same technique (OAuth) as Facebook called Twitter Sign-On, for authentication through their sign-in process. Here you can read about it:

http://dev.twitter.com/pages/sign_in_with_twitter



12.3 LinkedIn

LinkedIn provides the functionality to login to Value4All through their Sign-on infrastructure. The process behind the LinkedIn authentication is quite similar to Facebook Connect and Twitter Sign-In.



Here you can read about how you can login to your site through LinkedIn:

<http://developer.linkedin.com/index.jspa>

12.4 Result

LinkedIn, Facebook and Twitter support login to other sites like Value4All through them. There are *no* big differences between the technique, they are using to solve this functionality. The only differences are how much personal information details Value4All needs to import from these social networks. To answer this question, we have to know *what is the purpose of Value4All?* I think LinkedIn will be a good choice because:

- Almost all personal information details can be imported to Value4All.
- CV and personal presentations can also be imported to Value4All.
- You can also see the users academic studies and other important information that can be valuable for Value4All to import.

Imagine that an investor in Value4All became interested in a project, and he wants to know who owns the project. The information makes it easier for the investor to decide a possible investment in the project. In that situation, the investor maybe wants to see all personal information of the entrepreneur, CV, other academic studies and, etc. This information exists in LinkedIn that can be imported to Value4All by LinkedIn Sign-In Solution.

12.5 OAuth Protocol

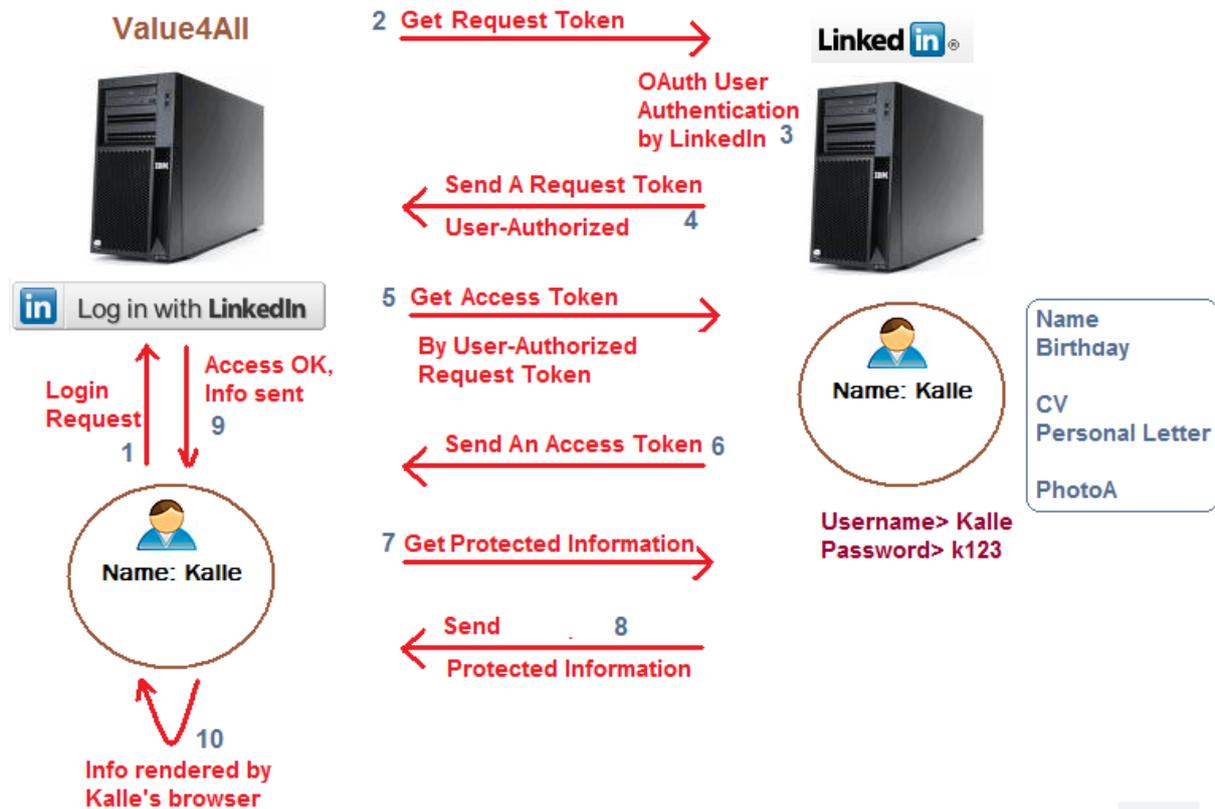
OAuth is a new technique (2006) that allows you to share the private information from one site to another site with shared username and password. With OAuth, you can share other resources like contact list, videos, photos, CV and, etc. The following example shows how OAuth can be applied between Value4All and LinkedIn.

Picture that, Kalle is an investor in Value4All. At the same time, Kalle is a member of LinkedIn with own username/password, photos, CV, Personal letter and, etc. Since Value4All is integrated with LinkedIn, all personal details can be imported from Kalle's LinkedIn personal page to Kalle's Value4All personal page, added with other Value4All specific information. Assume that, the information that Value4All imports from LinkedIn are:

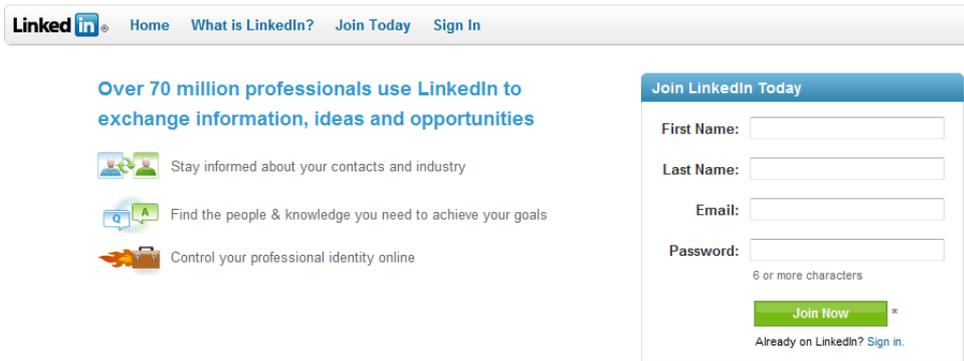
- Name
- Birthday
- CV

- Personal Letter
- Photo A

In the picture below, when Kalle wants to login to Value4All by clicking on "Log in with LinkedIn".



A Request Token is sent to LinkedIn server. A Request Token is not user-specific at this point and can be used to approve Kalle's login request. When Value4All server receives the Request Token, it sends Kalle to the LinkedIn OAuth User Authentication page and requests LinkedIn to redirect Kalle to Value4All when Kalle's login request was approved by LinkedIn. Now Kalle can access the Login page of LinkedIn. OAuth requires that LinkedIn first authenticates Kalle and then asks LinkedIn to grant access to Value4All.



Once Kalle can approve his identity to login, LinkedIn marks the Request Token as "User Authorized". Kalle is then redirected back to the Value4All start page with the Request Token as the user authorized. Value4All asks for an *Access Token* by Kalle's user authorized request token, to be able to retrieve Kalle's personal information.

The Request Token is used for user approval and Access Token is used to access protected resources like Kalle's Name, Birthday, CV, Personal Letter and Photo A.

Value4All receives all requested information for Kalle from LinkedIn. Kalle can now access his Value4All page by his browser.

Chapter 13

The Future Of Social Network Systems

Facebook, LinkedIn, Twitter and many other social networks growing very fast. Almost everybody knows what Facebook is. In the future, social networks can exist anywhere, and everywhere we need and want them to be. "*What's happening to our social life?*" is one question that is worth thinking about. I think that, the dependencies to social networks increase more and more. We spend more time on the internet, and we look for friends or maybe our life's big love on the internet. There are a more positive picture to such things now than before. As mentioned in the chapter 5 (A Social Network System), social networks are the forth most popular kinds of online activities with 67% of the world *online* population.

"SOCIAL NETWORKS WILL BE LIKE AIR!"

Internet access will be much better in the future, web 2.0 and other new techniques are coming all the time. I think all these techniques cause that our dependences on Internet increase more and more because, there are more important answers about our real life that we find out on the internet, for instance:

- Relationship: you search for your life's big love or you want to stay connected with your new internet friends.
- Activities: you maybe want to play games with other people on the internet.
- Doing business: you have to be updated to know about the price of your share portfolio or other businesses.

What's happening in the future is maybe, that all big social network companies and other big companies want to be corporate to each other to create something called (*Universal Identity Or Universal Sign-In*). It is very unpractical and difficult to have a

different profile on different sites, for instance, Facebook, LinkedIn, Twitter or having different accounts on MSN, Yahoo, Google. It will be a good idea to have a universal identity for every site you want to use. There will be, for instance, a few large centers for this purpose, saving our universal identities, in the future.

We are going from a translation of people's *real life* to a *digitalized life*. Personally, I don't like the idea, but the techniques are going to that direction. During the master thesis, I send a questionnaire to more than 100 private persons and a question was about "*What do you think that will happen in the future?*". The people who answered this question were few but almost all of them who answered the question said, that they will be able to use their social network sites everywhere (Mobile, Tv, and, etc.) and they like the technique very much.

How Can Value4All be developed in the future?

Picture this: during a conference, a managing director look for a new finance director. Within minutes, by Value4All connected to LinkedIn in his mobile, he identifies ten people with the right CV and two of them are looking for changing their jobs. His mobile tells him by GPS that, one of them is standing 10 meters from him in the conference. At the same time, a record of all people's profile that he found by his mobile during the conference, is automatically sent to his PC. It sounds more like a science fiction movie by James Bond, but this one can comes true in the future.

Chapter 14

Final Words

The master thesis had been very interesting and within a new and popular field of technique in the market. I started with little knowledge about techniques involved in social network platforms and challenged myself to learn some of them from scratch. An interesting observation was, how hard it can be to ask companies about their techniques and development environment. Probably, the information is secret company information.

A new experience was to work with bigger projects, learn to plan the work in good time and limit the scope of the project. That was also hard to filter the important information from the beginning but during the master thesis I became more effective to filter the relevant information.

Designing a good three tier application is very important because you have to think about, how to separate the presentation layer from the business logic and the database layer. Another new experience was, how the Java-library is structured and how the standardization of products looks like. It is very important to use standard solutions because it is easier for applications to integrate future techniques in the more flexible and easier way. Standard solutions are also proposed for Value4All's development environment.

Integrating Value4All to Facebook, Twitter or LinkedIn makes Value4All more valuable because the trustworthiness of the application will increase and there will be easier for people to become a member of Value4All. Developing a new login process is hard and requires good programming knowledge about the security issues and professional skills about design of the database for user information management in a secure way.

Value4All is a great idea because there are no competitors in the market. On the other hand, there are many *young* people with good and innovative ideas that don't know what to do, and at the end the idea is gone. Value4All can help them to introduce their ideas by writing a good business plan and find investors like social banks and private persons to make their ideas true.

Part VI

References & Appendices

Bibliography

- [1] E. F. Codd, *Proceedings of the 1980 workshop on Data abstraction, databases and conceptual modeling* .
ISBN:0-89791-031-1
- [2] Jason Hunter, William Crawford *Java Servlet Programming*,
ISBN: 0-596-00040-5,
<http://books.google.se/>
- [3] M. E. J. Newman, Juyong Park (PDF), *Why social networks are different from other types of networks*, 2010-04-05,
<http://arxiv.org/abs/cond-mat/0305612>
- [4] *Graph Database Comparison*, 2010-07-03,
<http://infogrid.org/blog/category/related-technologies/>
- [5] *Ken Thompson*, 2010-07-08,
<http://changethis.com/manifesto/show/19.BioteamingManifesto>
- [6] *Maturana and Francisco Varela*, 2010-05-03,
<http://www.users.globalnet.co.uk/~rxv/people/maturanavarela.htm>
- [7] *Social Network*, 2010-04-03,
http://en.wikipedia.org/wiki/Social_network
- [8] Tomas Björkholm & Hans Brattberg, *Prioritera, Fokusera, Leverera*,
ISBN-978: 9197863056.
- [9] *NoSQL Databases*, 2010-06-15,
<http://cstjanster.idg.se/sprakwebben/ord.aspordnosql>
- [10] *Proof-Of-Concept*, 2010-07-12,
http://en.wikipedia.org/wiki/Proof_of_concept
- [11] *Refactoring*, 2010-05-12,
http://en.wikipedia.org/wiki/Code_refactoring

- [12] *Introduction to Distributed System Design*, 2010-08-03,
<http://code.google.com/edu/parallel/dsd-tutorial.html>
- [13] *Sql Databases*, 2010-04-17,
<http://en.wikipedia.org/wiki/SQL>
- [14] Alexander Kolesnikov, *Tapestry, Building Web Application*,
ISBN-978: 184719307053999.
- [15] *Web Development*, 2010-04-08,
http://www.grassrootsdesign.com/intro/web/web_design.php
- [16] *Three Tier Architecture*, 2010-05-25,
<http://dotnetslackers.com/articles/net/introductionto3tierarchitecture.aspx>
- [17] Starr Hall and Chadd Rosenberg, *Social Network Design*, 2010-04-23,
<http://www.netsocializing.com/>
- [18] *A List Of Key-Value Stores*, 2010-06-28,
<http://www.metabrew.com/article/anti-rdbms-a-list-of-distributed-key-value-stores/>
- [19] *Get Connected: The Social Networking Toolkit For Business*,
ISBN-13: 9781599183589.
- [20] *Persistence Layer* (PDF), 2010-05-06,
<http://www.ambyssoft.com/downloads/persistenceLayer.pdf>
- [21] *Database Design*, 2010-06-25,
http://en.wikipedia.org/wiki/Database_design
- [22] *Database models*, 2010-06-25,
<http://www.djangobook.com/en/1.0/chapter05/>
- [23] Avinash Lakshman *Of Facebook*, 2010-07-13,
http://static.last.fm/johan/nosql-20090611/cassandra_nosql.pdf
- [24] *Key-Value Stores, A Practical Overview*, 2010-08-03,
http://blog.marc-seeger.de/assets/papers/Ultra_Large_Sites_SS09-Seeger_Key_Value_Stores.pdf
- [25] *CouchDB*, 2010-08-14,
<http://jchrisa.net/drl/ApacheCon-Talk-Slides/CouchDB-Intro-EU.pdf>

- [26] *A Database Perspective Of Social Network Analysis Data Processing*, 2010-06-05,
http://swp.dcc.uchile.cl/TR/2006/TR_DCC-2006-007.pdf
- [27] George Reese, *Java Database Best Practicies*,
<http://oreilly.com/catalog/9780596005221>,
ISBN:978-0-596-00522-1, ISBN 10:0-596-00522-9
- [28] *Higher Performance For A Real Time Workd*, 2010-08-21,
<http://www.odbms.org/download/029.01>
- [29] *A Bibliographic Exploration Tool Based On The DEX Graph Query Engine*, 2010-07-23,
<http://portal.acm.org/citation.cfm?id=1353439dl=GUIDEcoll=GUIDE-CFID=100412322CFTOKEN=24143391>
- [30] Marty Hall, Larry Brown *Core Web programming*,
ISBN: 0-13-089793-0,
<http://books.google.se/>
- [31] *NOSQL: Scaling To Size And Scaling To Complexity*, 2010-08-01,
<http://blogs.neotechnology.com/emil/2009/11/nosql-scaling-to-size-and-scaling-to-complexity.html>
- [32] *Object Database vs. Object-Relational Databases*, 2010-05-07,
<http://ce.sharif.ir/courses/84-85/1/ce384/resources/root/Object>
- [33] *Column-Oriented Database Systems*, 2010-07-25,
http://phdopen.mimuw.edu.pl/lato10/boncz_mimuw.pdf
- [34] Hans Bergsten *Java Server Pages*,
ISBN-0-596-00231-9,
<http://books.google.se/>
- [35] Frank Dabek, *A Distributed Hash Table*, 2010-07-28,
<http://pdos.csail.mit.edu/papers/fdabek-phd-thesis.pdf>
- [36] Frank Dabek, *Social Network - Living System*, 2010-05-28,
http://www.masternewmedia.org/social_network_design_the_network_is_a/
- [37] Frank Dabek, *Social Network - Design Considerations*, 2010-05-28,
<http://thenextweb.com/socialmedia/2010/09/27/6-design-considerations-for-your-enterprise-social-network/>

- [38] Frank Dabek, *CAP Theorem*, 2010-06-28,
http://en.wikipedia.org/wiki/CAP_theorem
- [39] *NoSQL Graph Databases Comparison*, 2010-08-01,
<http://java.dzone.com/news/nosql-graph-database-feature>
- [40] Thomas M. Connolly, Carolyn E. Begg *Database systems: A Practical Approach To Design, Implementation*,
ISBN-10: 0321210255, ISBN-13: 978-0321210258,
<http://books.google.se/>
- [41] *Ant And Maven Integration*, 2010-07-19,
http://www.jetbrains.com/idea/features/ant_maven.html
- [42] *Introduction to the Dependency Mechanism*, 2010-07-17,
<http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html>
- [43] *Servlets And JSP Tutorial*, 2010-07-18,
<http://www.apl.jhu.edu/hall/java/Servlet-Tutorial/>
- [44] Filip Zavoral, Jakub Yaghob, Pit Pichappan *Networked Digital Technologies, Part II: Second International*,
ISBN-10 3-642-14305-9,
<http://books.google.se/>
- [45] *NoSQL Databases Comparison*, 2010-07-28,
<http://nosql.mypopescu.com/post/978742866/document-databases-compared-couchdb-mongodb-ravendb>
- [46] *NoSQL Databases Comparison*, 2010-07-29,
<http://www.scribd.com/doc/26452377/Name-Cassandra-CouchDB-Dynomite-GlusterFS-HBase-Hypertable>
- [47] *XOOPS VS DROPAL*, 2010-05-11,
http://www.lastcombat.com/Xoops_vs_Drupal.html

Appendix A

Graph Databases - A Complete Comparison

	Neo4J	InfoGrid	DEX
Software			
Vendor	Neo Technoloies	NetMesh	DAMA-UPC
Version	1.0	29.3	3.0
Licence	AGPL V3	AGPL V3	Commercial -Restricted
Commercial	Y	Y	Y
Open Source	Y	Y	N
Schema			
Node Typing	N	Y	Y
Edge Typing	Y	Y	Y
Attributes (Nodes)	Y	Y	Y
Attributes (Edge)	Y	N	Y
Directed Edge	Y	Y	Y
Undirected Edge	N	Y	Y
Restricted Edge	N	N	Y
Loop Edge	N	Y	Y
Attribute Indexing	Partially (Only Nodes)	Partially (Only UUIDS)	Y
Starting Node	Y	Y	N
Schema	N	On Demand	N
Querying			
Language	Gremlin	N	N
Traversals	Y	Y	Y

Database			
Transactional	Y	Y	N
ACID	Y	Partially	
Full-indexed	On Demand	Store Dependant	Y
Distributed	Partially (RM API)	Y	N
Cache	Y	Y	Y
Embedded	Y	Y	Y
Store-engine	On Demand	Flexible	Customized
Migration framework	Y	Y	N
Object Mapping	N	Y	N
Utilities			
Shell	Y	HTTP	Y
Algorithm	Y	N	Y
Benchmark			
Protocols	Y	Y	N
RDF Store	Y	N	N
OLW Store	Partial	N	N
SPARQL Integration	Y	N	N
IDE Integration	Y (Eclipse)	Y (Netbeans)	N
Admin Tool	N	Y (Viewlets)	Y
Importer	Y	Y	N
Exporter	Y	Y	Y
Loader	Y	Y	Y
Scripting Language	N	N	Y
Language Support			
Java	Y	Y	Y
C#	N	N	N
Ruby	Y	N	N
Python	Y	N	N
C	N	N	N
C++	N	N	Y
PHP	Y	N	N

Operating System			
Windows	Y	Y	Y
Linux	Y	Y	Y
Mac OS X	Y	Y	N
Unix	Y	Y	N
Other	Patial (JVM Enabled)	Partial (JVM Enabled)	N

Appendix B

Social Network Creators - Comparison

ELGG

Category	ELGG
Cost	Free
Source Code	Freely available via stable releases and development SVN
Codebase	LAMP (Linux, Apache, MySql and Php)
Access Control	Yes, users and groups (extensible via plugins)
Wiki	Yes
Forum	Yes
Blog	Yes
Media Sharing	Image, Video, Audio, Documents any filetype, Automatic podcast support via Kaltura Elgg Plugin
Messaging	Yes
Event Calendar	Yes
Social Grouping	Yes
Connectivity	Yes (Facebook, YouTube, Twitter)
Contact Management	Yes
Customizable	Extensible via plugins with a flexible API, Skinnable, Available in many languages

XOOPS with Yogurt extension

Category	XOOPS with Yogurt extension
Cost	Free
Source Code	Some codes are available.
Codebase	LAMP (Linux, Apache, MySql and Php)
Access Control	Yes, Permissions by group
Wiki	Yes
Forum	Yes
Blog	Yes
Media Sharing	Personal album of pictures, Videos from YouTube, Mp3 files, Documents
Messaging	Yes
Event Calendar	Yes
Social Grouping	Yes
Connectivity	Yes (Facebook, LinkedIn, Youtube)
Contact Management	Yes
Customizable	Theme-based skinnable interface, Extensible via plugins, Available in many languages

Appendix C

NoSQL Databases - Comparison

Language And Fault-Tolerance

Name	Language	Fault-tolerance
Project Voldemort	Java	Partitioned, Replicated, Read-repair
Ringo	Erlang	Partitioned, Replicated, Immutable
Scalaris	Erlang	Partitioned, Replicated, Paxos
Kai	Erlang	Partitioned, Replicated
Dynomite	Erlang	Partitioned, Replicated
MemcacheDB	C	Replication
ThruDB	C++	Replication
CouchDB	Erlang	Replication, Partitioning
Cassandra	Java	Replication, Partitioning
Hbase	Java	Replication, Partitioning
Hypertable	C++	Replication, Partitioning

Persistence

Name	Persistence
Project Voldemort	Pluggable: BerkleyDB, Mysql
Ringo	Custom on-disk (append only log)
Scalaris	In-memory only
Kai	On-disk Dets file
Dynomite	Pluggable: couch, dets
MemcacheDB	BerkleyDB
ThruDB	Pluggable: BerkleyDB, Custom, Mysql, S3
CouchDB	Custom on-disk
Cassandra	Custom on-disk
Hbase	Custom on-disk
Hypertable	Custom on-disk

Client Protocol And Data Model

Name	Client Protocol	Data model
Project Voldemort	Java API	Structured / Blob / Text
Ringo	HTTP	blob
Scalaris	Erlang, Java, HTTP	Blob
Kai	Memcached	Blob
Dynomite	Custom ascii, Thrift	Blob
MemcacheDB	Memcached	Blob
ThruDB	Thrift	Document oriented
CouchDB	HTTP, Json	Document oriented (Json)
Cassandra	Thrift	Bigtable meets Dynamo
Hbase	Custom API, Thrift, Rest	Bigtable
Hypertable	Thrift, Other	Bigtable

Docs And Community

Name	Docs	Community
Project Voldemort	A	Linkedin, No
Ringo	B	Nokia, No
Scalaris	B	OnScale, No
Kai	C	No
Dynomite	D+	Powerset, No
MemcacheDB	B	Some
ThruDB	C+	Third rail, Unsure
CouchDB	A	Apache, Yes
Cassandra	F	Facebook, No
Hbase	A	Apache, Yes
Hypertable	A	Zvents, Baidu, Yes

Appendix D

The Popularity Of The Market's Social Platforms

A questionnaire was sent to more than 100 private persons.

1. Do you know what a social network (Facebook, Twitter, LinkedIn or something like that) is?
2. Are you using these sites? Which of them do you use?
3. Do you know anybody else who use these services?
4. Are you interested in using these sites?
5. Which group do you think is more interested in using these sites (gender and age)?
6. Are these service user friendly?
7. Can they improve their sites? How?
8. Do you want to use these services everywhere you want to?
9. What do you think will happen in the future?

Appendix E

Market's Development Environment Solutions

For each category that is mentioned below, I ask some companies by a questionnaire, if they could provide me some information about their development environment solution.

- Development language(s).
- Database solution(s).
- Version control system.
- Application server/Web server.
- Frameworks.
- Do you recommend your environment?
- What do you think will happen with social networks in the future?

Appendix F

Application Installations

This chapter is about installation, integration and configuration of all applications and tools needed for the prototype and Value4All. The development environment solution is a stable and flexible environment that can be used for development of any type of social network systems or similar three tier applications.

F.1 What we'll need?

Applications and the download pages are listed below.

- IntelliJ IDEA, download page: <http://www.jetbrains.com/idea/>.
- GIT, download page: <http://git-scm.com/>.
- Apache Maven, download page: <http://maven.apache.org/>.
- Subversion, download page: <http://subversion.tigris.org/>.
- Jetty Web Server, download page: <http://jetty.codehaus.org/jetty/>.
- JUnit, download page: <http://www.junit.org/>.
- Selenium Test Server, download page: <http://seleniumhq.org/>.
- Java, download page: <http://java.sun.com/javase/downloads/widget/jdk6.jsp>.
- Neo4J Database, download page: <http://neo4j.org/>.
- Cassandra database, download page: <http://cassandra.apache.org/>.

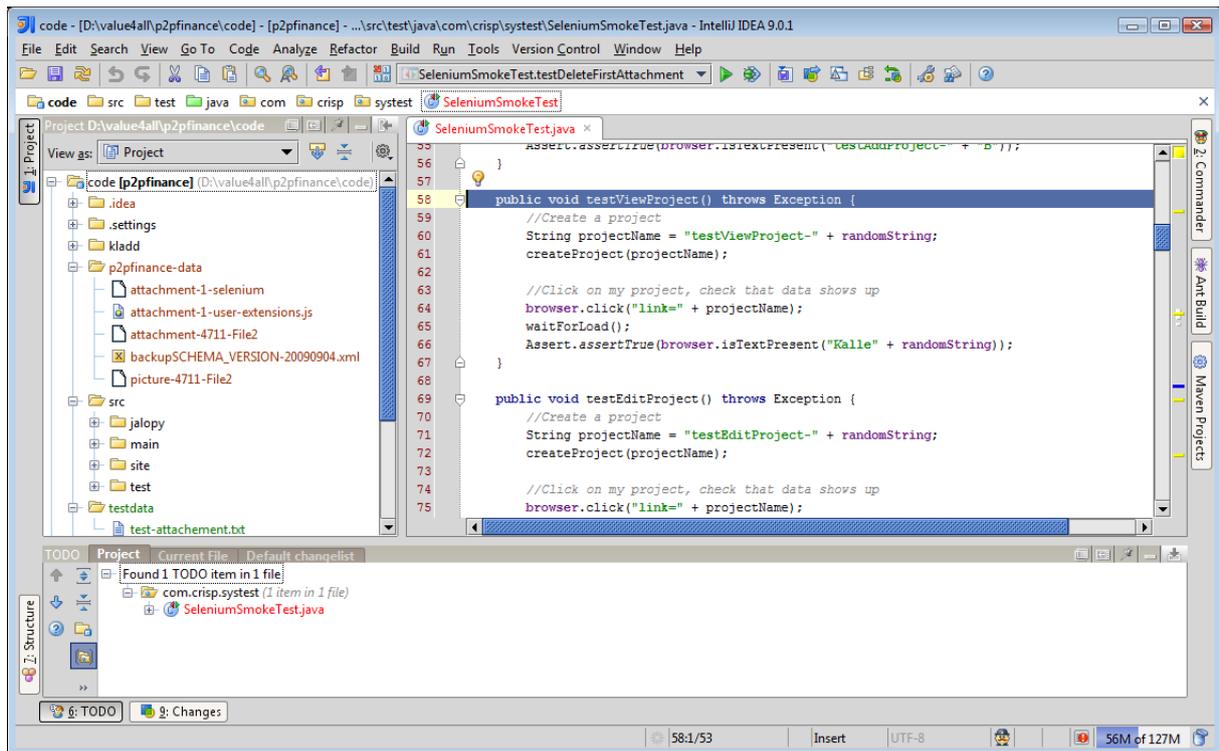
F.2 Applications And Their Versions

- IntelliJ IDEA: *ideaIC-9.0.2.exe*.
- GIT: *Git-1.7.0.2-preview20100309.exe*.
- Apache Maven: *apache-maven-2.2.1-bin.zip*.
- Subversion: *Setup-Subversion-1.6.6.exe*.
- Jetty Web Server: *Jetty-6*.
- Junit: *junit-4.8.1.jar*.
- Selenium Test Server: *selenium-core-1.0.1.zip*.
- Java: *jdk1.6.0₁₂.Neo4J Database : neo4j-apoc-1.0.tar*.
- Cassandra: *apache-cassandra-0.6.2-bin.tar.gz*.

Start off by downloading each application in a given folder, for instance, on your desktop.

F.3 IntelliJ IDEA Installation

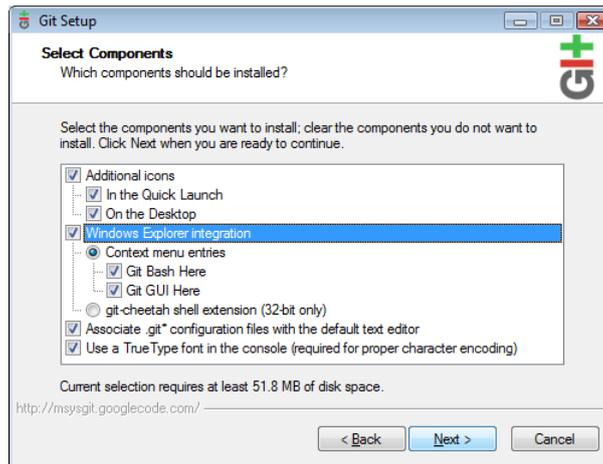
First of all, unzip Java JDK, in your system and set the system path to your Java JDK folder. Now you can install IntelliJ IDEA by double-clicking on the file "ideaIC-9.0.2.exe". Choose an installation folder and follow the installation instructions.



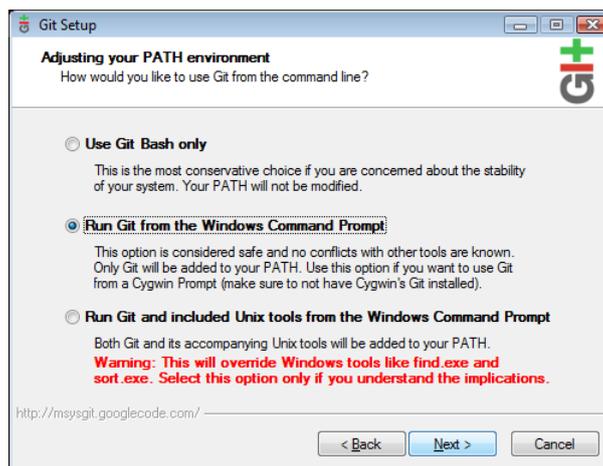
The next step is to create a new application, if you don't have an existing application. For creation of a new application writes the name of your application (for instance, "value4all"). If you have an existing application, you only need to locate the application from IntelliJ IDEA for loading.

F.4 Git Installation

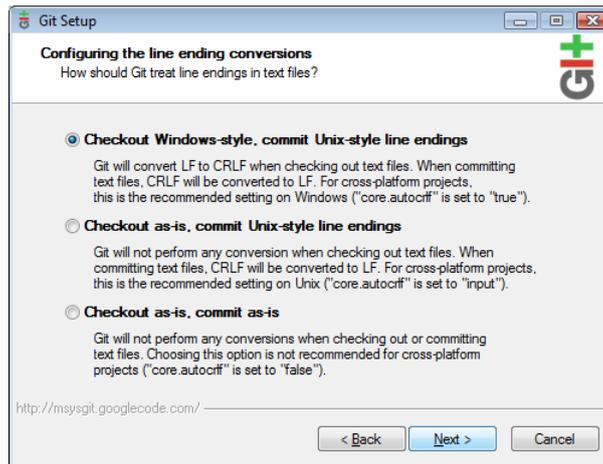
Double-click on the file "Git-1.7.0.2-preview20100309.exe", choose the installation folder and click next. Make sure that you have checked "Git Bash Here" and "Git GUI Here" options from "Windows Explorer Integration". Click next and next.



In the next step, check "Run Git from Windows Command Prompt", which is needed by "IntelliJ IDEA plugin" and then click next.

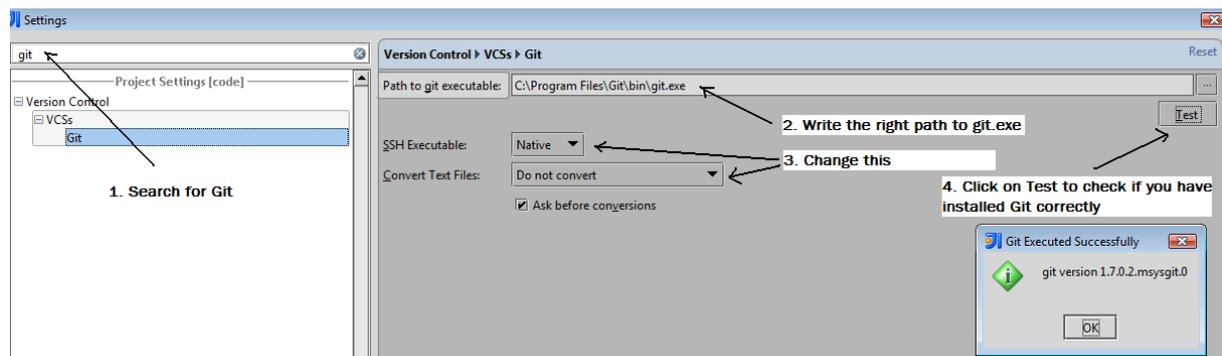


Now, check "Checkout Windows-style, commit Unix-style", which is the recommended setting on windows environment. Click next and finish the installation.



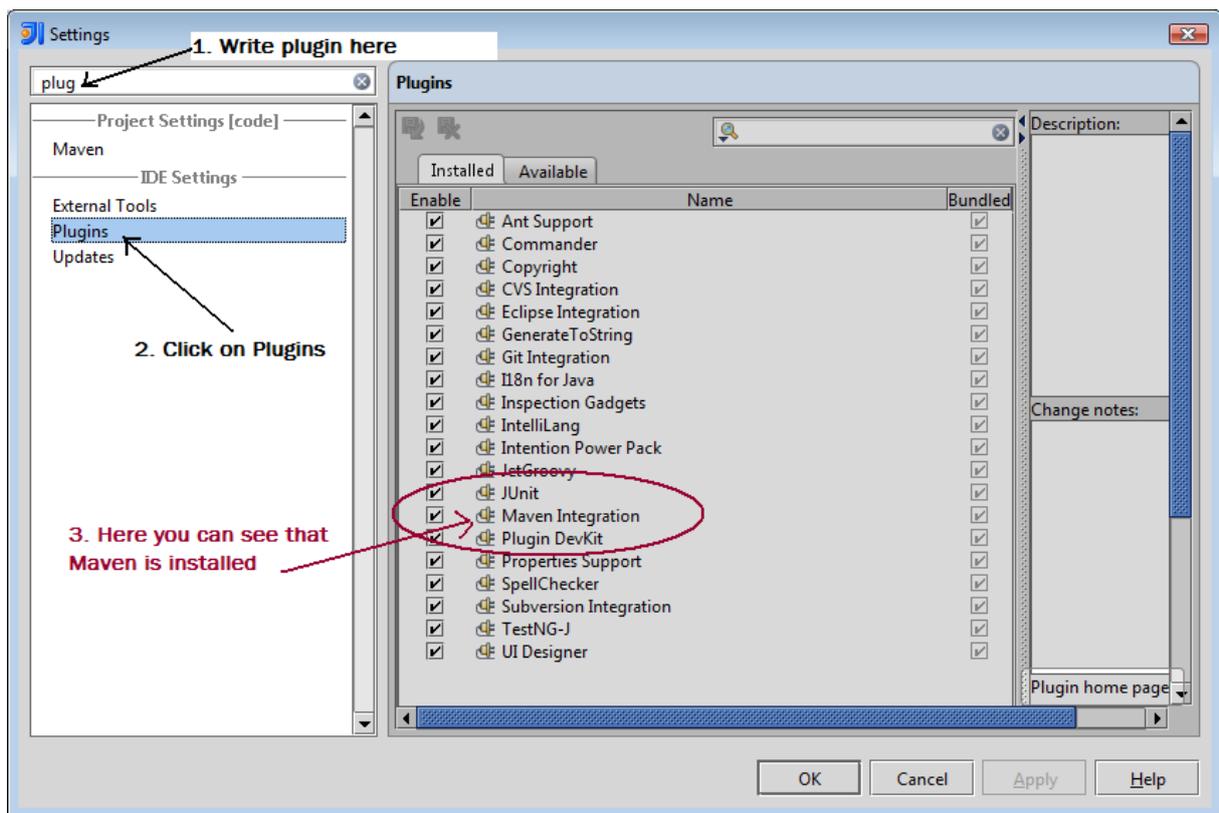
F.4.1 Git Installation in IntelliJ IDEA

Check if IntelliJ IDEA has the right path to "git.exe". Go to File and click on Settings. Do the changes according to the picture below. Click on "Apply" and then Ok.

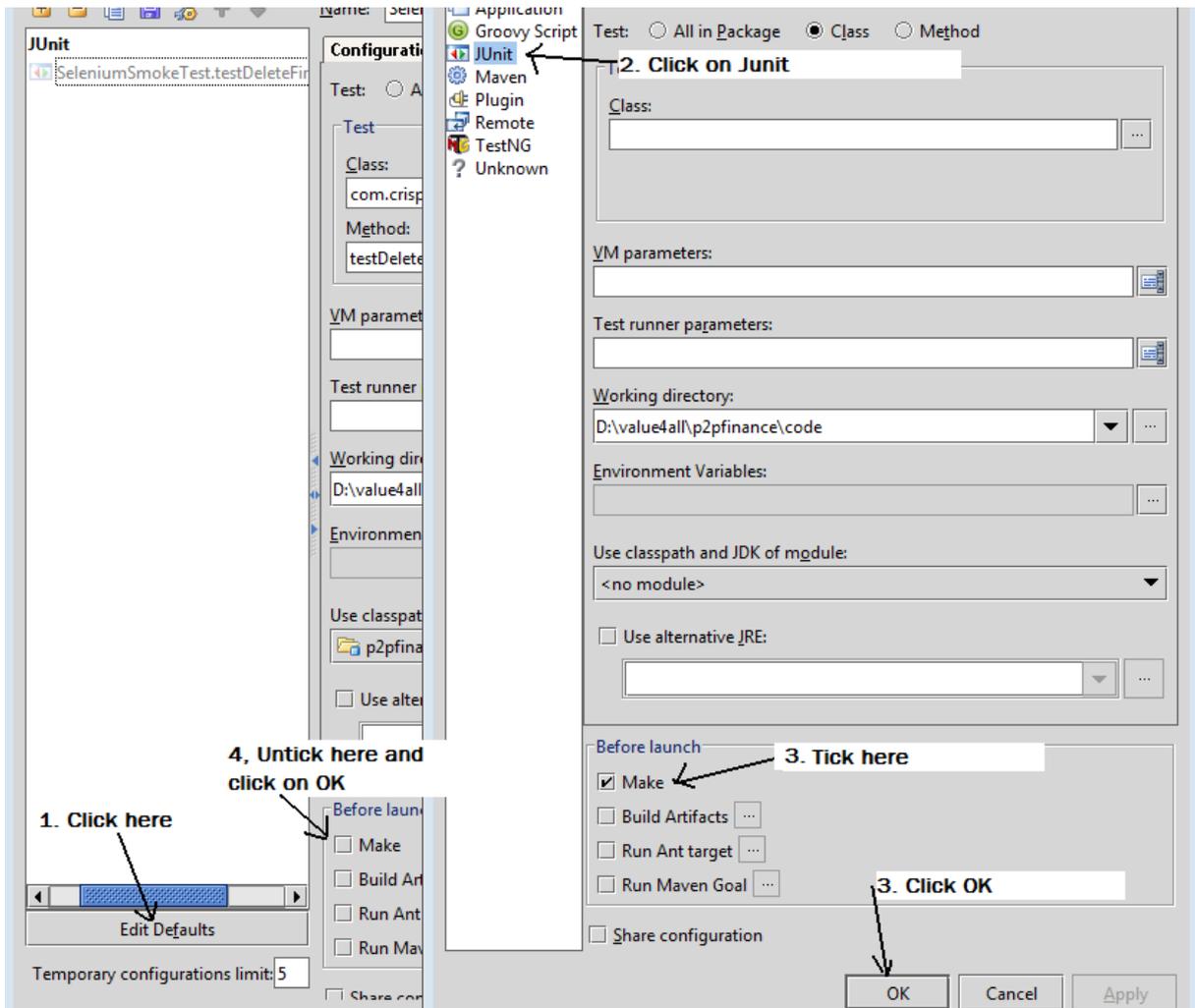


F.5 Maven and Junit Installation

Maven is installed by default in IntelliJ IDEA as a Plugin. Open IntelliJ IDEA. Go to File and choose Settings. Follow the instructions as follows.

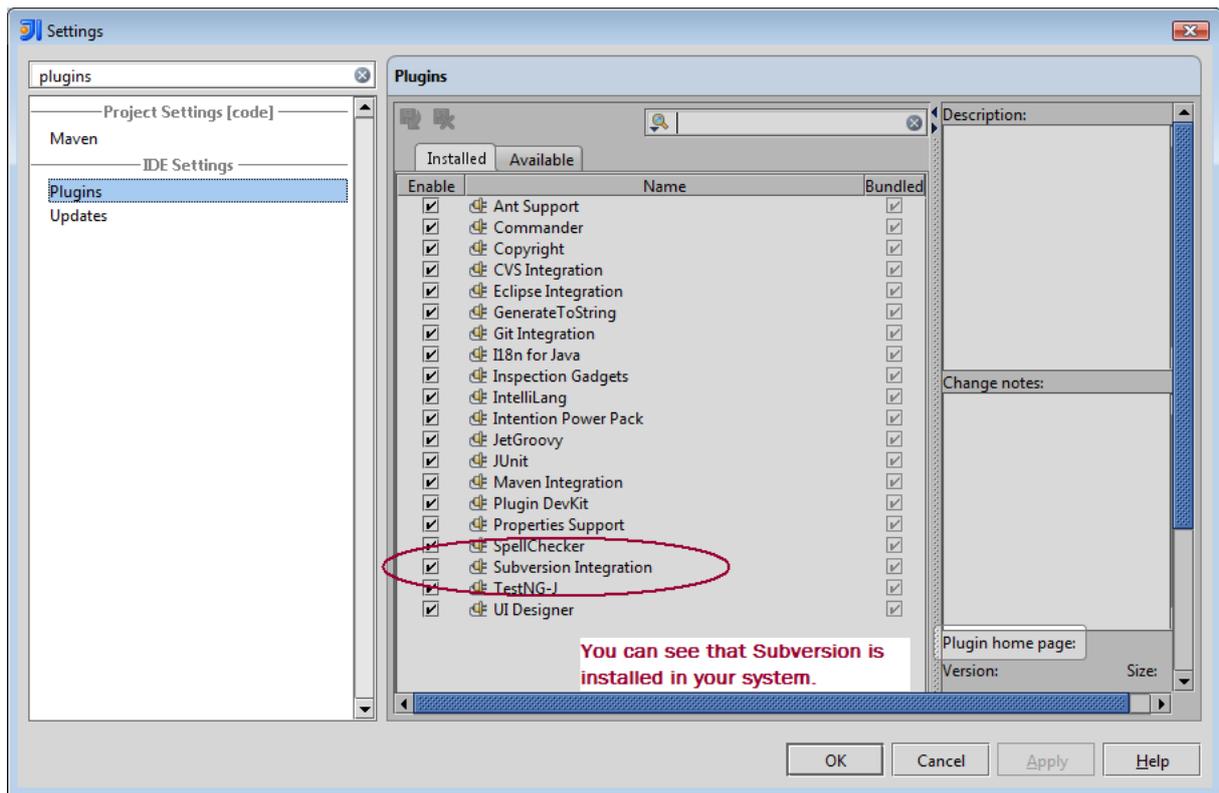


Otherwise, unzip "apache-maven-2.2.1-bin.zip" and connect IntelliJ IDEA by giving the right path to the unzipped maven folder. In the picture above you can also see that, Junit is installed in IntelliJ IDEA by default. Otherwise import the file "junit-4.8.1.jar" to IntelliJ IDEA. Click on Run, choose Edit Configurations and do the following changes.

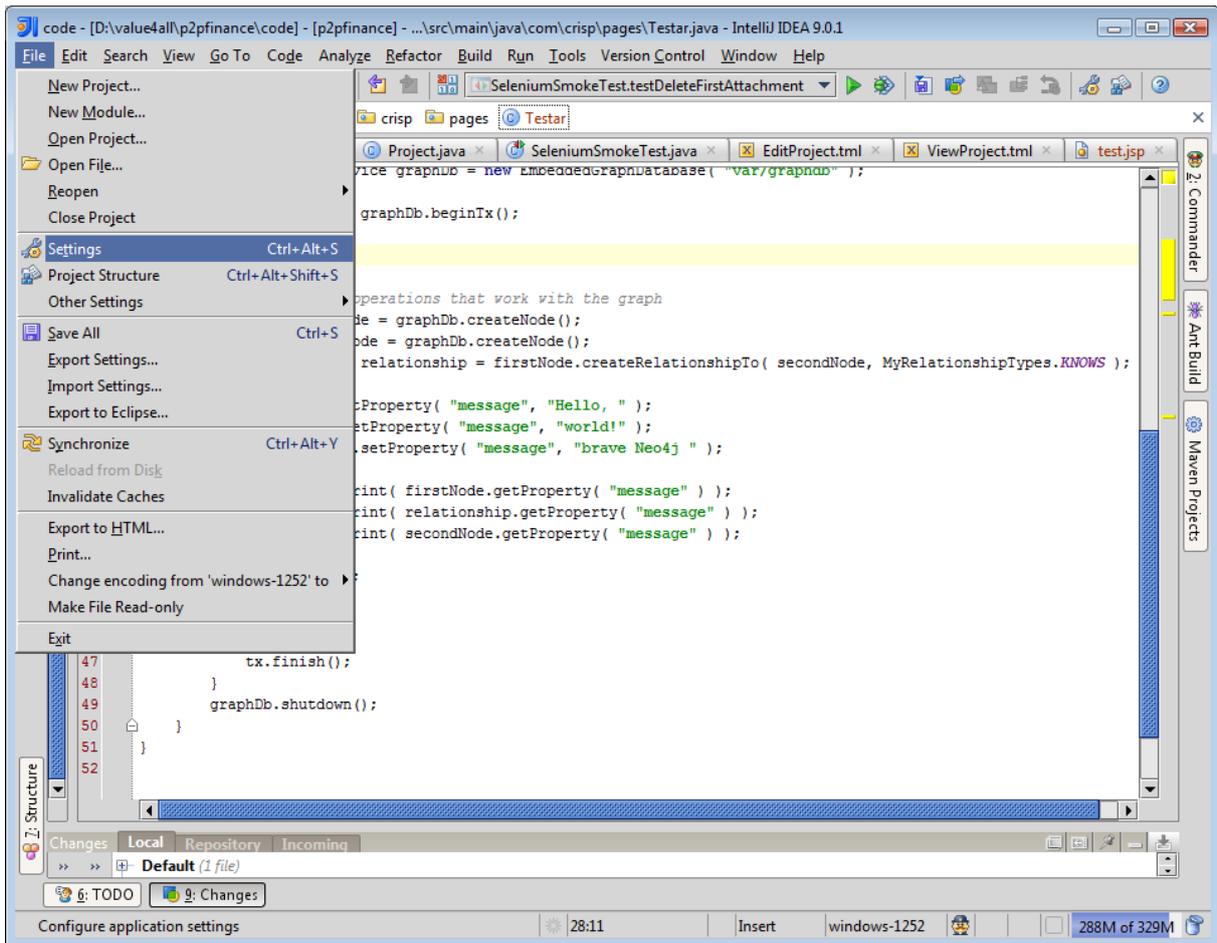


F.6 Subversion Installation

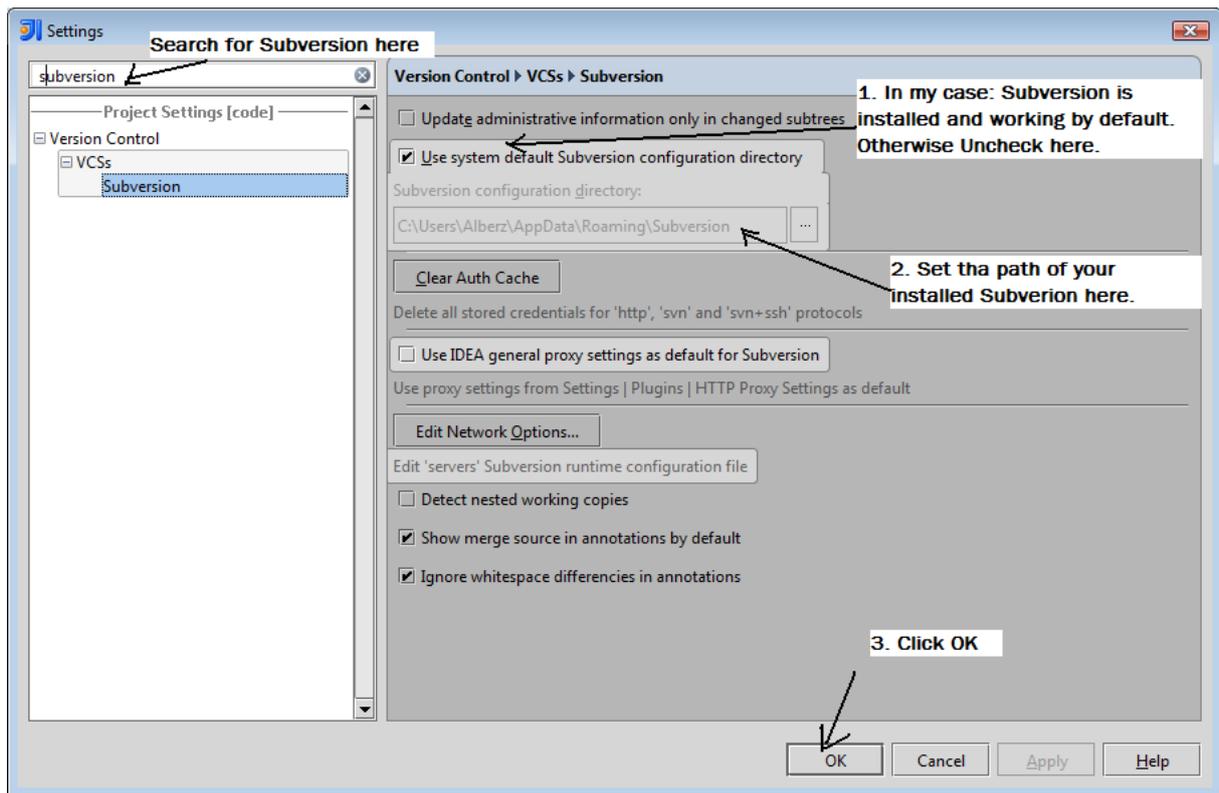
Subversion is integrated in IntelliJ IDEA by default. You can check if you have it as follows.



Otherwise install the file "Setup-Subversion-1.6.6.exe" and follow the installation instructions. Open IntelliJ IDEA. Click on *File* → *Settings*.



Search for subversion.



F.7 Selenium Installation

For installation of Selenium test server with maven in IntelliJ IDEA, open your maven configuration file "POM.xml" and add the following changes to it.

```

<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>selenium-maven-plugin</artifactId>
  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-server</artifactId>
      <version>2.0a1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <executions>
    <execution>
      <phase>pre-integration-test</phase>
      <configuration>
        <background>true</background>
      </configuration>
      <goals>
        <goal>start-server</goal>
      </goals>
    </execution>
  </executions>
</plugin>

```

Start the Selenium server as follows.

```
mvn selenium:start-server
```

F.8 Jetty Installation

For installation of Jetty web server with maven in IntelliJ IDEA, open your maven configuration file "POM.xml" and add the following changes to it.

```

<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>maven-jetty-plugin</artifactId>
  <version>6.1.10</version>
  <configuration>
    <scanIntervalSeconds>10</scanIntervalSeconds>
    <stopKey>foo</stopKey>
    <stopPort>9999</stopPort>
  </configuration>
  <executions>
    <execution>
      <id>start-jetty</id>
      <phase>pre-integration-test</phase>
      <goals>
        <goal>run</goal>
      </goals>
      <configuration>
        <scanIntervalSeconds>0</scanIntervalSeconds>
        <daemon>true</daemon>
      </configuration>
    </execution>
    <execution>
      <id>stop-jetty</id>
      <phase>post-integration-test</phase>
      <goals>
        <goal>stop</goal>
      </goals>
    </execution>
  </executions>
</plugin>

```

Start the Jetty web server as follows.

```
mvn jetty:run
```

F.9 Neo4J Installation

Neo4J can be downloaded automatically by Apache Maven, but you have to configure your Maven repository file (POM.xml) first. Do the following changes to it.

```

        <version>1.2.2</version>
    </dependency>

    <dependency>
        <groupId>org.neo4j</groupId>
        <artifactId>neo4j-kernel</artifactId>
        <version>1.0</version>
    </dependency>
</dependencies>

```

Add these code to your pom.xml

```

<build>
    <finalName>p2pfinance</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>

```

And the following changes.

```

        <url>http://www.chenillekit.org/mvnrepo/release</url>
        <snapshots>
            <enabled>>false</enabled>
        </snapshots>
    </repository>

    <repository>
        <id>neo4j-public-repository</id>
        <name>Publically available Maven 2 repository for Neo4j</name>
        <url>http://m2.neo4j.org</url>
        <snapshots>
            <enabled>>true</enabled>
        </snapshots>
    </repository>
</repositories>

```

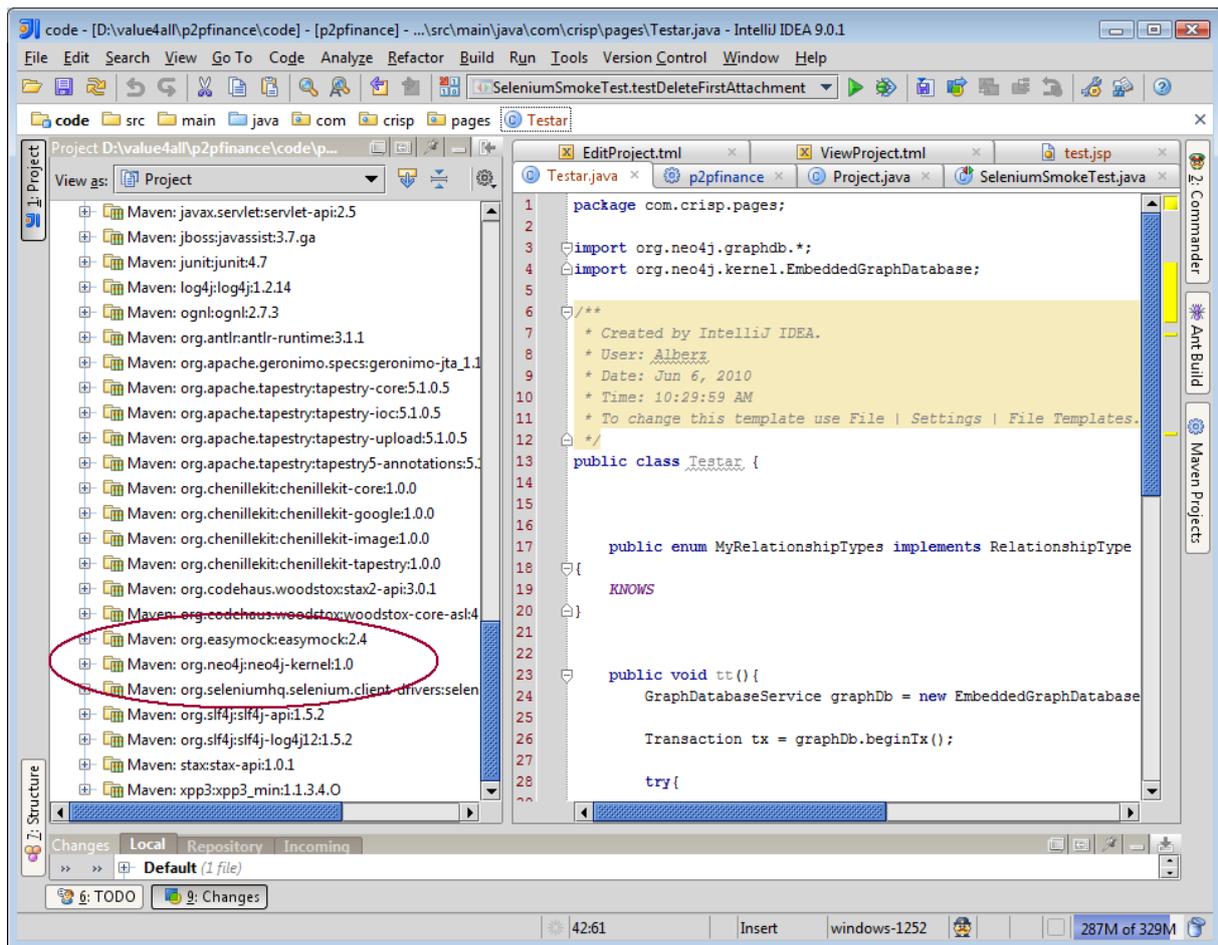
Add these lines to your pom.xml

```

</project>

```

Restart your application, you have now Neo4J kernel jar file installed in your system in IntelliJ IDEA. Check it under External Libraries.



E9.1 Testing Neo4J Installation By An Example

You can copy the following example into your application to test the Neo4J database installation.

```

import org.neo4j.graphdb.*;
import org.neo4j.kernel.EmbeddedGraphDatabase;

public class Test {

    public enum MyRelationshipTypes implements RelationshipType
    {
        KNOWS
    }

    public void firstTry(){

```

```

GraphDatabaseService graphDb = new EmbeddedGraphDatabase( "var/graphdb" );
Transaction tx = graphDb.beginTx();
try{
Node firstNode = graphDb.createNode();
Node secondNode = graphDb.createNode();
Relationship relationship = firstNode.createRelationshipTo( secondNode, MyRelation-
shipTypes.KNOWS );

firstNode.setProperty( "message", "Hello, " );
relationship.setProperty( "message", "Neo4J " );

System.out.print( firstNode.getProperty( "message" ) );
System.out.print( relationship.getProperty( "message" ) );
tx.success();
}
finally
{
tx.finish();
}
graphDb.shutdown();
}
}

```

F.9.2 Indexing In Neo4J

Indexing is not supported as a part of Neo4J core. The following is a list of dependencies you need to install.

- neo4j-graph-algo.
- neo4j-index.
- neo4j-kernel.
- neo4j-remote-graphdb.
- neo4j-shell.
- protobuf-java.
- jline.

- geronimo-jta_1.1_spec.
- lucene-core.
- neo4j-online-backup.

You can download these libraries from:

<http://components.neo4j.org/neo4j-apoc/dependencies.html>.

Now you have to install all of these libraries. Perform the "*mvn instal*" to all of these libraries.

Go to your maven catalogue:

```
D:/value4all/p2pfinance/code>mvn install:install-file -Dfile=C:4j-graph-algo-0.6-20100709.201559-78.jar -DgroupId=org.
```

Now open your application maven configuration file (pom.xml) and do the following changes for all of these libraries.

```
<dependency>
<groupId>org.neo4j</groupId>
<artifactId>neo4j-graph-algo</artifactId>
<version>0.6-SNAPSHOT</version>
</dependency>
```

F.10 Cassandra And Cassandra Client Installations

It is easy to install Cassandra database, but it is difficult to find a good Cassandra client. I found two clients.

- Thrift API. Very low level and it is very hard to work with it.
- Hector API. Higher lever than Thrift API and it is easier to work with it.

A complete installation process of Cassandra database and Hector are described below. I tried with Thrift API first, but it was quite difficult for coding. Therefor I used Hector client instead of Thrift.

F.10.1 Cassandra Installation

Extract the file "apache-cassandra-0.6.2-bin.tar.gz", for instance, in "D:/cassandra". Now open the file:

"D:/cassandra/conf/storage-conf.xml"

Do the following changes to make Cassandra to start on a single node. *Cassandra is designed to run on a cluster.*

```
<!--
~ Directories: Specify where Cassandra should store different data on
~ disk. Keep the data disks and the CommitLog disks separate for best
~ performance
-->
<!--<CommitLogDirectory>/var/lib/cassandra/commitlog</CommitLogDirectory>
<DataFileDirectories>
  <DataFileDirectory>/var/lib/cassandra/data</DataFileDirectory>
</DataFileDirectories-->
<CommitLogDirectory>D:/cassandra/data/commitlog</CommitLogDirectory>
<DataFileDirectories>
  <DataFileDirectory>D:/cassandra/data/data</DataFileDirectory>
</DataFileDirectories>
<CalloutLocation>D:/cassandra/data/callouts</CalloutLocation>
<BootstrapFileDirectory>D:/cassandra/data/bootstrap</BootstrapFileDirectory>
<StagingFileDirectory>D:/cassandra/data/staging</StagingFileDirectory>

<!--
~ Addresses of hosts that are deemed contact points. Cassandra nodes
~ use this list of hosts to find each other and learn the topology of
~ the ring. You must change this if you are running multiple nodes!
-->
<Seeds>
  <Seed>127.0.0.1</Seed>
</Seeds>
```

Open your windows command prompt and do the following steps.

```

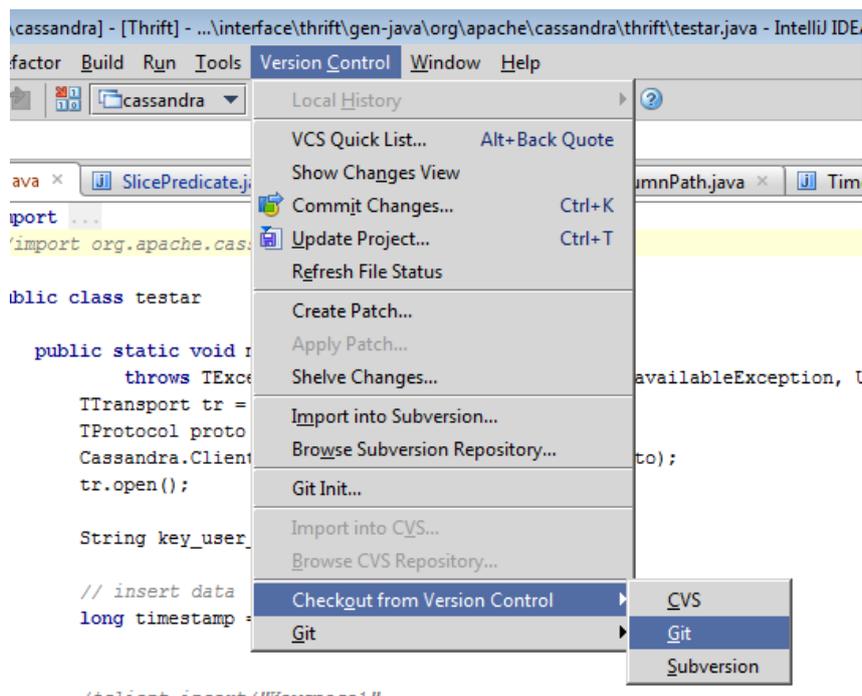
C:\Windows\system32\cmd.exe - bin\cassandra.bat
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

C:\Users\Alberz>d:
D:\>cd cassandra
D:\cassandra>bin\cassandra.bat
Starting Cassandra Server
Listening for transport dt_socket at address: 8888
INFO 20:05:40,407 Auto DiskAccessMode determined to be standard
INFO 20:05:41,110 Sampling index for D:\cassandra\data\data\system\LocationInfo-1-Data.db
INFO 20:05:41,161 Sampling index for D:\cassandra\data\data\system\LocationInfo-2-Data.db
INFO 20:05:41,173 Replaying D:\cassandra\data\commitlog\CommitLog-1276081412374.log
INFO 20:05:41,179 Log replay complete
INFO 20:05:41,337 Saved Token found: 15745698210689325775881206821363166806
INFO 20:05:41,341 Saved ClusterName found: Test Cluster
INFO 20:05:41,346 Creating new commitlog segment D:/cassandra/data/commitlog\CommitLog-1276106741346.log
INFO 20:05:41,500 Starting up server gossip
INFO 20:05:41,675 Binding thrift service to localhost/127.0.0.1:9160
INFO 20:05:41,710 Cassandra starting up...

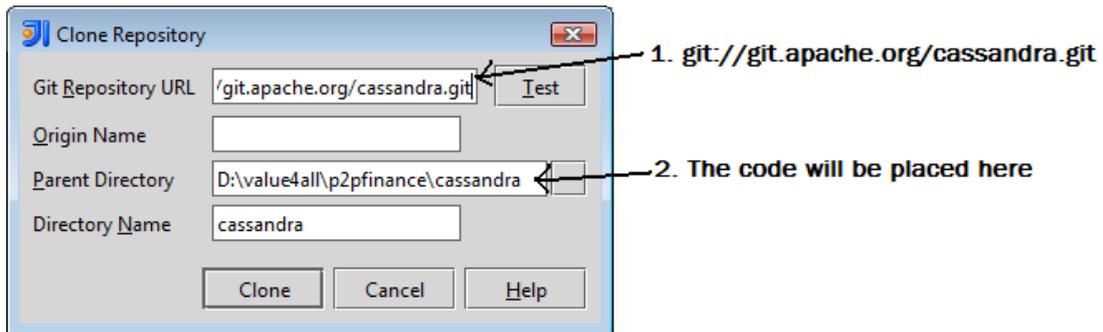
```

F.10.2 Cassandra-Thrift Installation

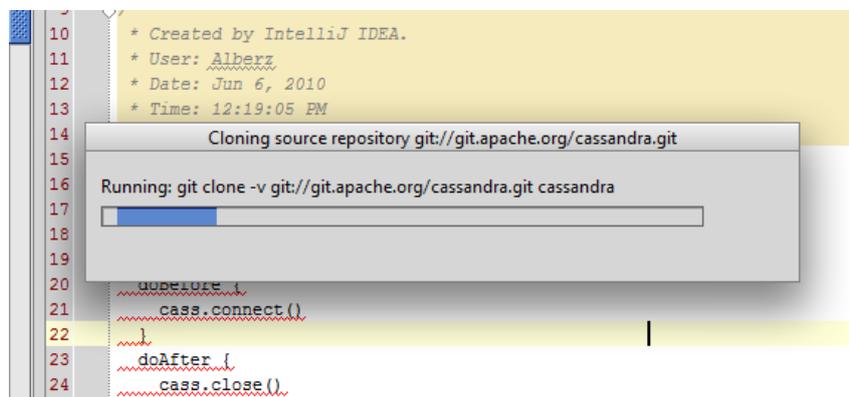
Open IntelliJ IDEA, select Version Control -> Checkout from Version Control and select Git.



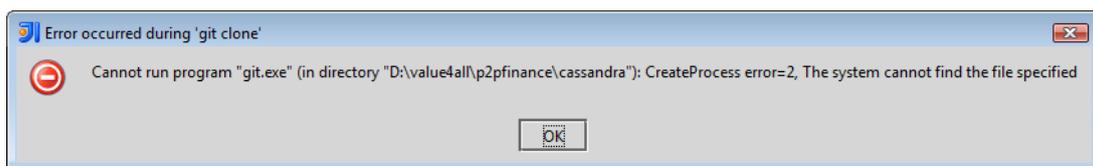
A. Now do the following settings and click Clone.



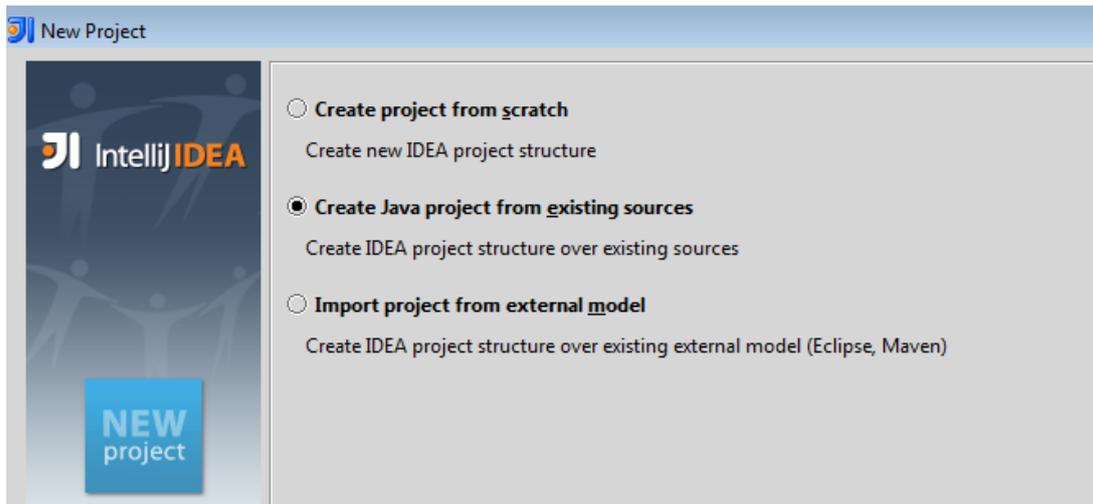
Wait until the checkout process is finished.



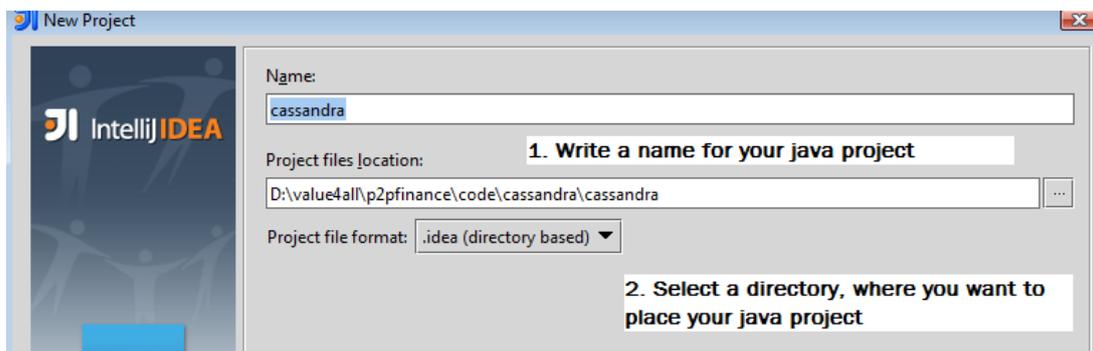
In case you get an error message as the picture below, check your firewall settings because Git uses a non standard port that is usually blocked.



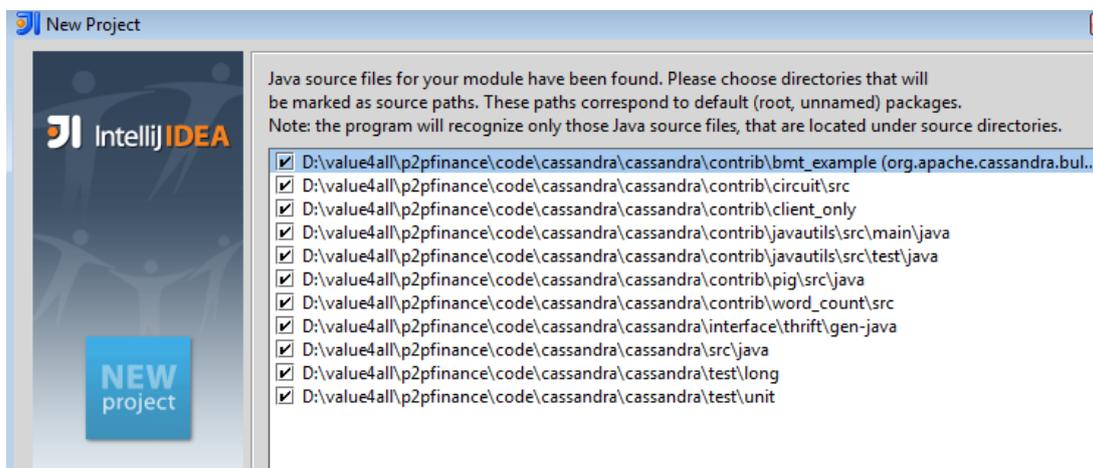
Otherwise, answer "yes" if IntelliJ IDEA asks you "Would you like to create an IntelliJ IDEA project for the sources you have checked out?" and then do the following changes and press next.



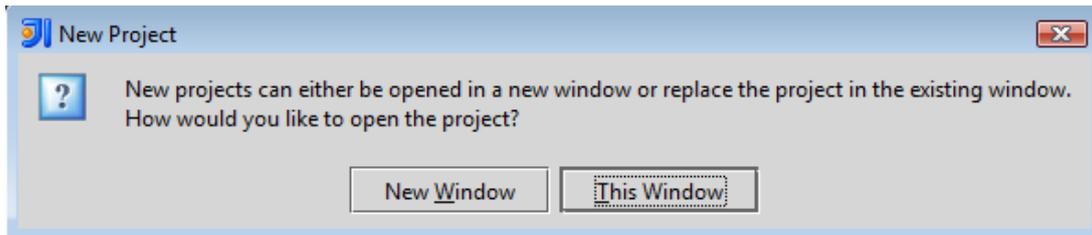
Do the following settings and press next.



Add the following Java package to your application and press next and next and finish.

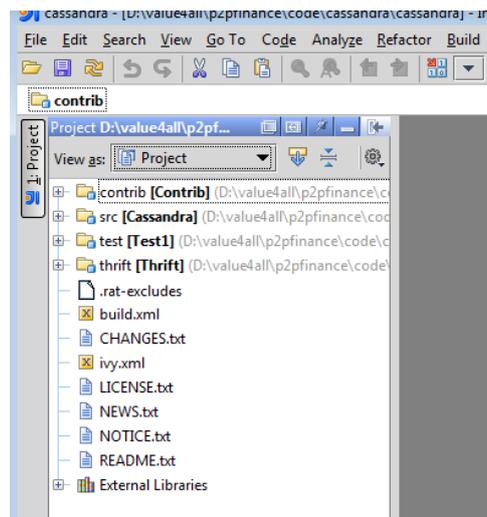


IntelliJ IDEA asks you the following question.

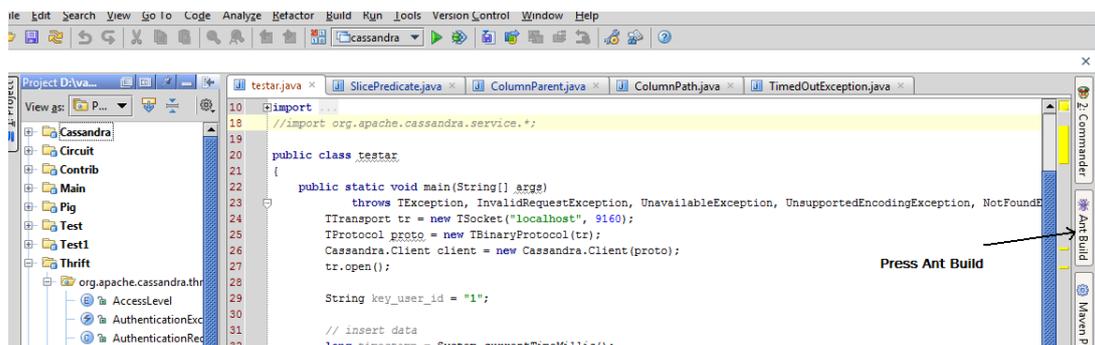


If you answer "This window" to the question, that means, your working area will change to the new application in the same window, and if you answer "New window", the new application will open in a new window. The import process is in progress now, wait until the process indexing is finished.

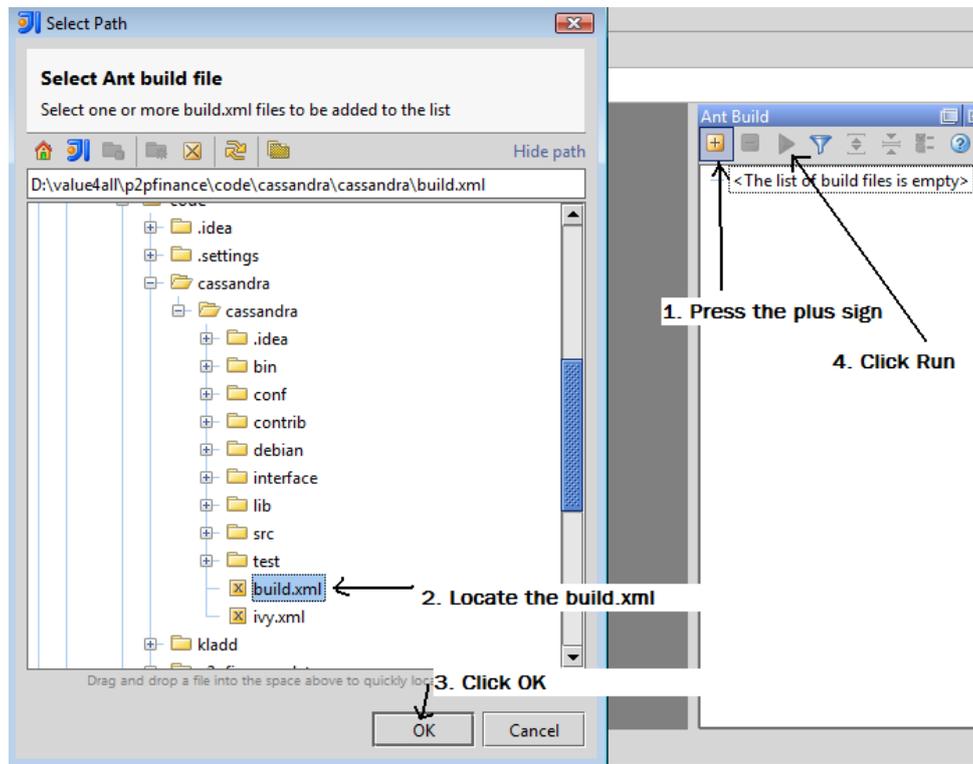
B. Now you have something like this:



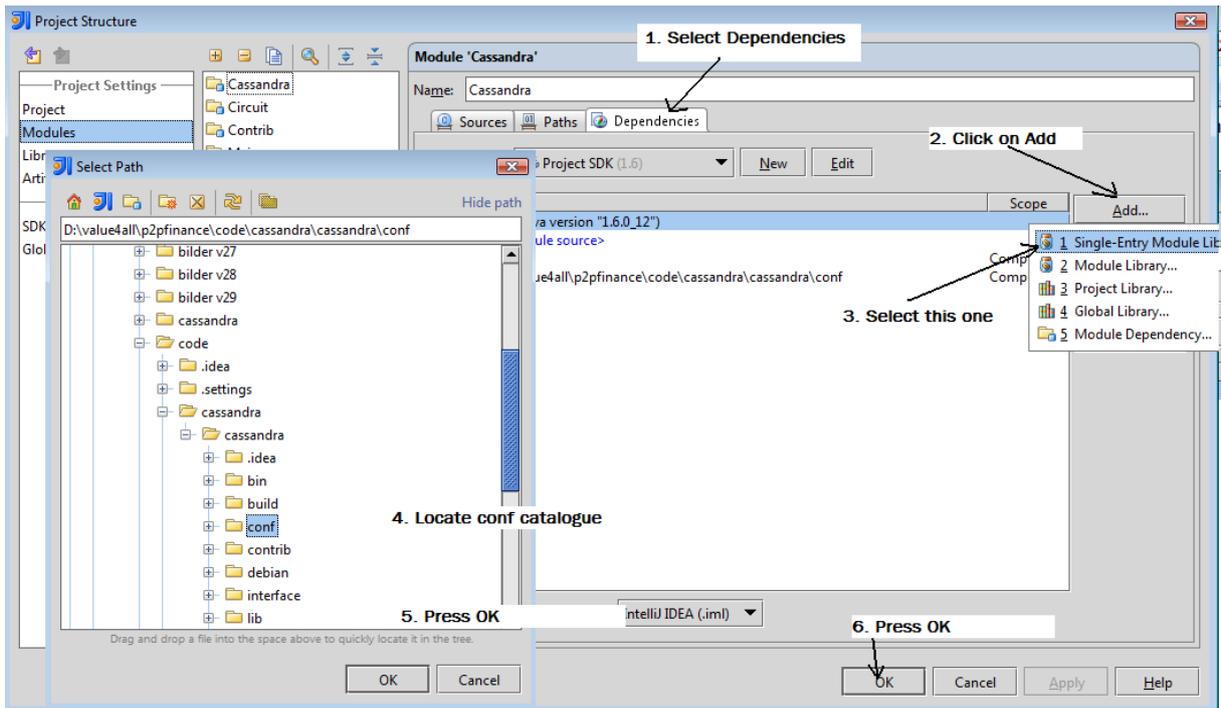
Press the Ant Build and do the following steps.



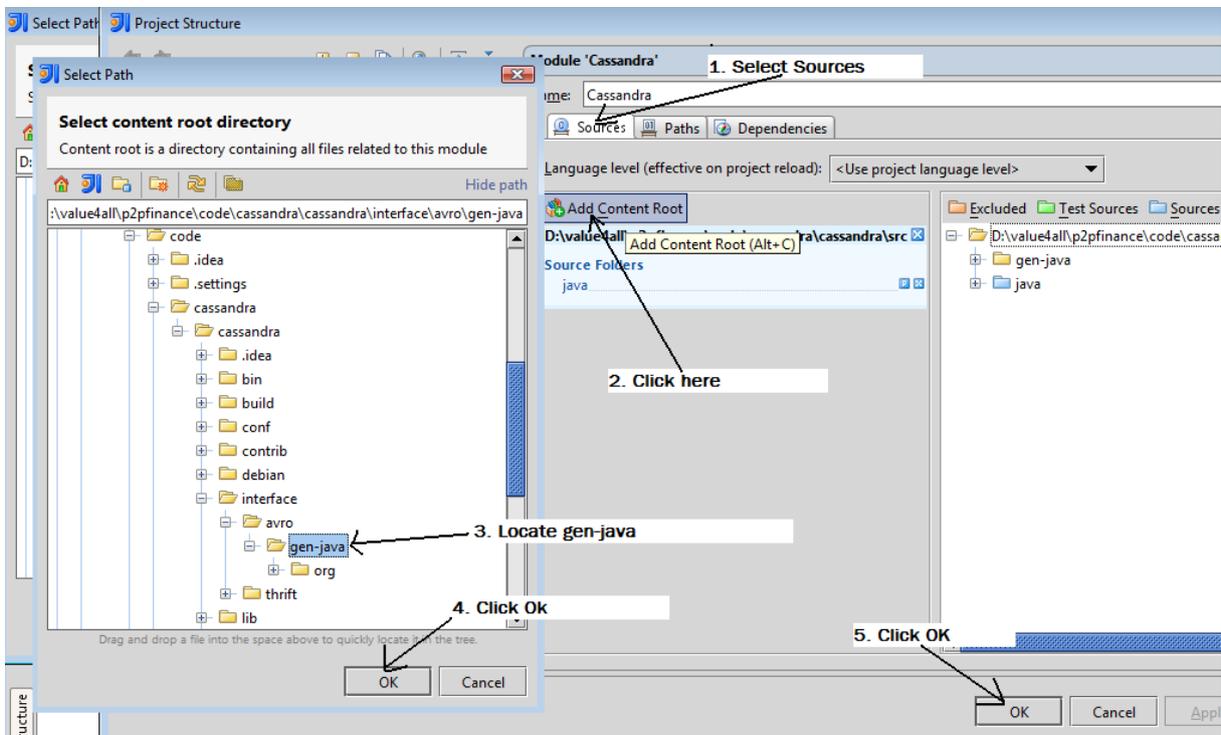
Do the following steps and click ok and wait until the Ant builds process is completed.



C. Change the view to Packages. Right click on your application (in my case, "cassandra") and select "Module settings". Select dependencies. Press "Add" and select "Single Entry Module Library..." and find the conf catalogue.

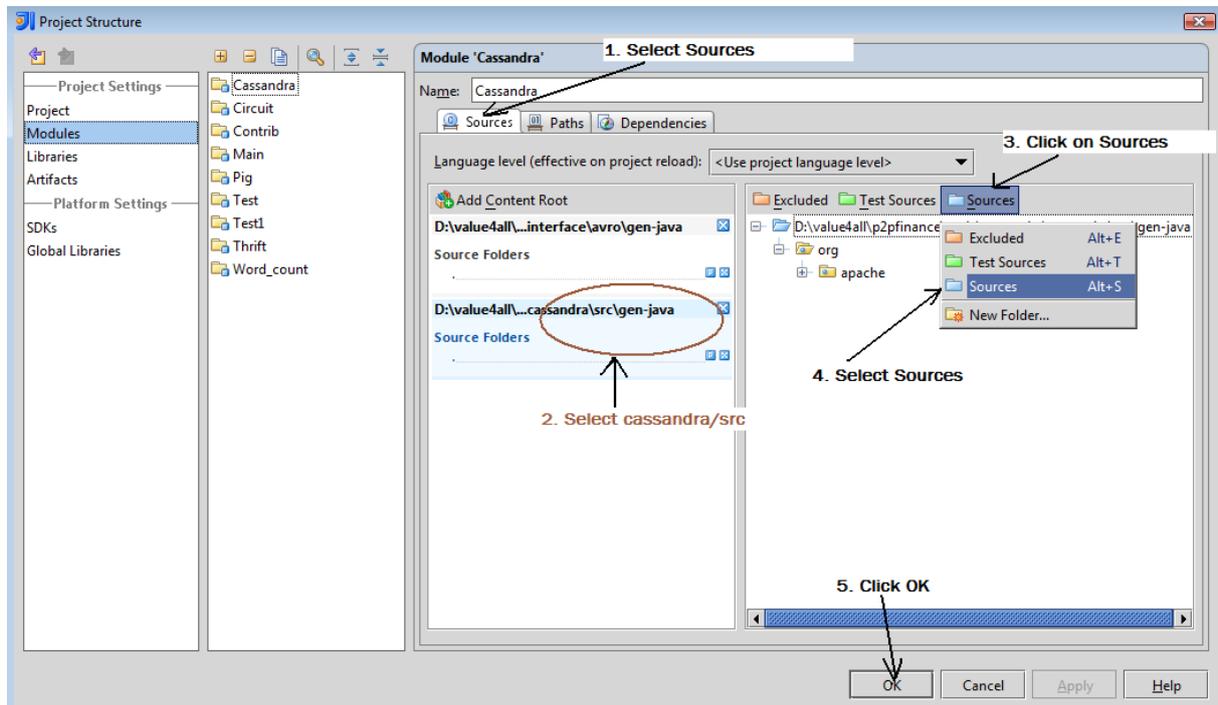


Right click on the application and select "Module settings". Press "Add Content Root" and find "gen-java" under "interface/avro/"



Now do the same configurations to locate "gen-java" under "interface/thrift/"

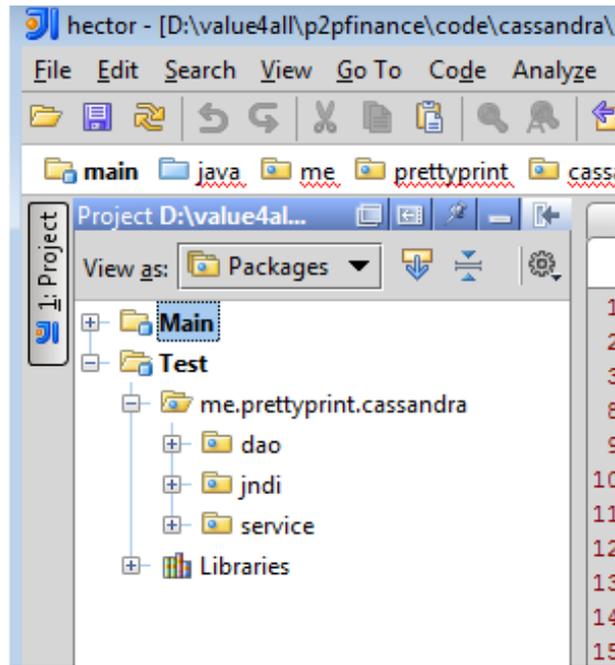
Finally, the last step is to choose "src" under "cassandra" as content root. Right click on the application and select "Module settings". Do the changes as follows.



F.10.3 Cassandra-Hector Installation

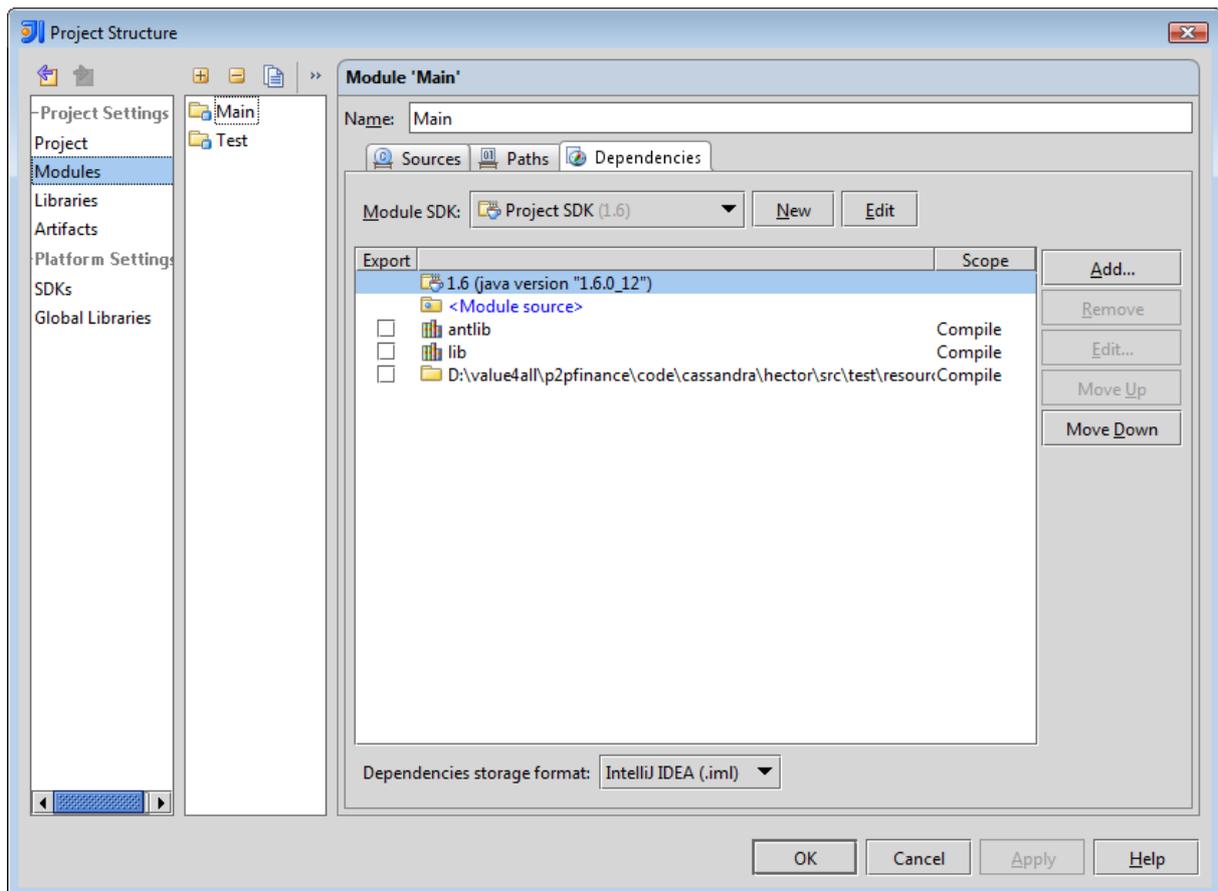
Almost all steps of Hector API installation and configurations are similar to Thrift installation and configuration. The differences are:

- Step A. GIT repository for hector is <http://github.com/rantav/hector.git>
- Step B. Remember to change the name of the application to "hector" and also the source folder to another than the folder for thrift.



- Step C. Add a "Single Entry Module Library..." for Hector here. Follow all steps as mentioned for the Thrift installation and choose your hector "*storage-config.xml*". The folder for that file is

"D:/value4all/p2pfinance/code/cassandra/hector/src/test/resources".



F.11 Master Thesis Hardware Environment

I used a flash memory (4GB) and a laptop, Acer Aspire 5530 with the following capacity.

- Processor, CPU: AMD Turion X2 Dual-Core Mobile RM-70 2.00 GHz.
- Memory (RAM): 4 GB.
- System type: 32-bit Operating System.
- Hard disk: 300 GB.

Softwares

- Windows Vista as operative system.
- Latex for writing the final paper.
- Firefox, Google chrome and Internet Explorer as a web browser.
- Windows Command Shell (CMD) and Cygwin as a command prompt.

- Skype as a way to communicate with my instructor from CRISP.

TRITA-CSC-E 2010:168
ISRN-KTH/CSC/E--10/168-SE
ISSN-1653-5715