Methods of graphically viewing and editing business logic, data structure

Shahzad Gul
Abstract

Methods of graphically viewing and editing business logic, data structure

Shahzad Gul

For financial institutions such as, who desire to lend money to the borrower or an invoice factoring companies who are looking to buy invoices, credit policy is a primary concern and vital measure for the risk or marketability associated with the particular issue. The goal of credit risk management is to maximize the profitability while reducing the risk of credit loss within the acceptable parameters.

In the wake of high profile bankruptcies, there has been recent speculation about the correctness and reliability of credit risk prediction.

Based on these issues, there is sufficient motivation to develop more gainful policy analysis system to predicting credit worthiness of the customers.

The system is investigated; designed and implemented that accomplishes the requirements of Risk department at Klarna. The result of the thesis work is KLAPAS, which enables the users to create custom policies and apply them on selective set of transactions.
Acknowledgments

First of all, praise is due to almighty ALLAH with His compassion and mercifulness to allow me finalizing this Master thesis project. I would like to thank Klarna AB for offering me this opportunity.

My thanks go to my family. They always worried about creating the required circumstances to carry out my studies.

I would like to express my gratitude to my supervisor, Martin Engström for his guidance and efforts. Without his help and kindness, this achievement would have been far away. My thanks also go to Justin Pearson for his support to guide me through the whole process especially in report writing.

During my stay at Klarna I found it a warm working atmosphere. I also appreciate the help and support from all persons who were directly or indirectly involved in my project. I would like to acknowledge the help of Tobbe, Peter, Richard, Uwe, Haseeb and Jordi.
# Table of contents

Introduction ......................................................................................................................... 8
  1.1 Invoice Factoring ........................................................................................................... 8
  1.3 Goals ............................................................................................................................. 9
  1.4 Thesis structure ............................................................................................................. 9

Related work ......................................................................................................................... 10

Klarna environment ............................................................................................................... 12
  3.1 Workflow of Klarna system ........................................................................................... 12
  3.2 Business Risk ............................................................................................................... 13
  3.3 Erlang ............................................................................................................................ 14
  3.4 Erlang’s characteristics ................................................................................................. 14
  3.6 Erlang web development .............................................................................................. 16
  3.6 XML-RPC .................................................................................................................... 16
  3.7 QLC ............................................................................................................................... 17
  3.8 The Klarna system ....................................................................................................... 17
  3.9 Klarna system architecture overview ........................................................................... 18

Problem analysis .................................................................................................................... 19
  4.1 Background ................................................................................................................... 19
  4.2 Klarna credit policy ...................................................................................................... 19
  4.3 Problem description ..................................................................................................... 21
  4.4 Requirement analysis ................................................................................................. 21
  4.5 Functional requirements .............................................................................................. 21
  4.6 Non-functional requirements ...................................................................................... 22

System architecture .............................................................................................................. 23
  5.1 Structure of the application ......................................................................................... 23
  5.2 Nitrogen ....................................................................................................................... 26
  5.3 Mochiweb .................................................................................................................... 28
  5.4 EryDTL ........................................................................................................................ 28
Table of figures

Figure 1.1 a simple factoring transaction with Klarna.................................................................8
Figure 2.1 the impact of downtime on business ...........................................................................13
Figure 2.2 Process creation times ...............................................................................................15
Figure 2.4 Klarna system architecture........................................................................................18
Figure 4.1 Klarna policy system....................................................................................................20
Figure 5.1 Erlang web framework comparison chart ..................................................................26
Figure 5.2 Nitrogen with Mochiweb .........................................................................................27
Figure 5.3 Nitrogen webpage .......................................................................................................28
Figure 5.4 Mochiweb web server configuration .........................................................................28
Figure 6.1 RPC inter communication .........................................................................................31
Figure 6.2 KLAPAS ......................................................................................................................32
Figure 6.3 Policymaker ................................................................................................................33
Figure 6.4 Add new policy ...........................................................................................................33
Figure 6.5 Date parameters ..........................................................................................................35
Figure 6.6 Table selection ...........................................................................................................35
Figure 6.7 Selection/Return variables .........................................................................................36
Figure 6.8 Filter/Condition variables ........................................................................................37
Figure 6.9 Session data ...............................................................................................................38
Figure 6.10 Acceptance ratio ......................................................................................................38
Figure 6.11 Invoice wise purchases ............................................................................................39
Figure 6.11 shows the purchase invoice comparison between two custom policies. ...............39
Figure 6.12 Rejection reasons ....................................................................................................39
List of Acronyms

QLC: Query List Comprehension
KLAPAS: Klarna Policy Analysis System
RPC: Remote Procedure Call
XML-RPC: Extensible Markup Language – Remote Procedure Call
OTP: Open Telecom Platform
DSL: Domain Specific Language
NN: Neural Network
HTTP: Hyper Text Transfer Protocol
Chapter 1

Introduction
This chapter starts with the history of the organization where this thesis was carried out. Moreover, it describes the problems and goals in general terms.

1.1 Invoice Factoring
In this modern era, any business can increase its cash flow by using a business tool called factoring, where the invoices for sold goods and services are still not paid. Factoring is quick, easy and a risk-free way to improve the cash flow of the company. It is a type of financial transaction in which three parties are involved: the one who sells the product, the buyer, and the third party (factor) which buys the invoices. One of the major benefits of using factoring company is to reduce the credit risk.

![Simple factoring transaction with Klarna](image)

*Figure 1.1 a simple factoring transaction with Klarna*

Klarna AB, where this thesis was carried out, is a Sweden-based factoring company which buys the invoices from different companies. The company transfers ownership of all receivable invoices to Klarna along with all rights and risks related to receivable invoices. Hence, Klarna obtains the right to collect the payments made by the buyer for the invoice amount and have to bear the loss if buyer does not pay invoice amount. This factor increases the importance of credit risk prediction in Klarna.
1.3 Goals

As of today, all the modern financial companies have a policy system to evaluate and minimize the credit risk. In Klarna policy system, all policies are defined in the Domain Specific Language (DSL), called wiki language. The policy rules are written separately and not in the source code, which makes it easier to modify rules. In order to change the policy, one must change that specific file and then view the result. Furthermore, it was not feasible to analyze the live policy. The investigation is carried out to check the solution for above mentioned issues. The investigation has the following goals:

- Understand the current policy system.
- Get the requirement from relevant departments.
- Investigate the tools for development.
- Develop the policy analysis system.
- Implementation of the developed system.

1.4 Thesis structure

Following is the outline of the thesis:

Chapter 1 gives the introduction of the company, Klarna. It also explains the motivation behind this thesis and goals in general terms.

Chapter 2 describes what research has addressed for similar problems and solutions and how does my work differ from it.

Chapter 3 presents a brief explanation of the Klarna system workflow and main technologies used in Klarna.

Chapter 4 explains the problem description and objectives of the thesis in more detail.
Chapter 5 describes the methods and tools used to accomplish the task.
Chapter 6 starts explaining the solution itself.
Chapter 7 describes what went wrong and what went good during the development.
Chapter 8 dedicated for conclusion and future work.
Chapter 2

Related work

There are few efforts has been made in the past to develop an efficient system to analyze the business rules in the financial institutes. Credit scoring [4] is a most common and used method. The Credit scoring is way in which the factor company pools together a large number of factors for its credit decisions. While calculating the score, a variety of data sources may be used, including data from an application form, credit references agencies or from previous purchase history of the customer.

In recent years, Neural networks (NN) have also been proposed to forecast credit risk because of their advantages of treating non-linear data with self-learning capability. The majority of the NN approaches to default prediction use multilayer networks. ‘Feed-forward’ NNS are might be the most popular network architecture in use today. They are sometimes referring to as ‘back-propagation’ NNS or ‘multi-layer perceptrons’ [18]. However, due to “black box” syndrome it has significant inadequacy [17].

Klarna has its own policy system that verifies the credit worth of the buyer through statistical analysis. Likewise credit scoring, it also uses the information of the buyer credit lending however it does not rank them on the bases of credit score. The work done in this thesis is also based on previous data of customers stored in the database. This solution is different from the credit scoring in different ways. Some of them are:

- It provides an interface to the user to fetch the buyer’s information.
- It enables the user to edit the policy rules and create new policies.
- It is possible to apply multiple policies on buyer’s data.
- The results can be visualized through Charts.

The work done in this thesis is specific and related to Klarna. The starting point of the thesis is based on another application that is developed initially to fulfill the requirements of the concerned department. It was a quick solution, developed only for testing purposes but it lacked in certain areas. Following are the major limitations in that application:

- Only one policy at a time can be applied on test person.
- The data can be fetched only for one test person at a time.
- It was difficult for the end user to run the DSL (Domain Specific Language) queries.
- No graphical analysis can be done.
- It was not possible to apply policy on the fetched test persons.
- It does not have a convenient procedure to edit the policy system.

After identifying the limitations of the system, I started collecting requirements from the Risk department. The first question I asked myself was whether I should continue with the current application or start from the scratch.

The testing application was developed by focusing the testing; therefore it contains irrelevant functionality which is not needed. Further, it was developed in old version of Nitrogen therefore it is required to do lot of changes according to new API. Lastly, the new requirements are mainly focused on getting graphical analysis of the rules, which is not done in it.

These reflections led me to the conclusion that I should start the application from scratch. However, I used some backend modules of data fetching.
Chapter 3

Klarna environment

This chapter is intended to disseminate the reader with a set of fundamental concepts relevant to this thesis work. It starts with briefing details of Klarna workflow and main technologies used by the system. Next, it explains the Erlang language, its characteristics and reason for being used in Klarna. Lastly, the system architecture of the company is explained in more detail.

3.1 Workflow of Klarna system

Klarna AB, previously Kreditor, is Sweden-based invoice factoring company founded in the 2005. Since then Klarna not only became one of Sweden’s rapidly growing company while keeping high profitability. It increases the sales of ecommerce stores by offering simple and safe payment solutions to the end-customer.

This is reachable through fast fail over to a warm standby and live code changes (using Erlang hot code swapping). The database is native Erlang (Mnesia) running in non-distributed mode, with a replication layer on top. Klarna is using an extensive user interface for customers (~4000 Ecommerce sites). It directly interacts with ecommerce store, banks and persons.

On the user’s site Klarna is available as payment method among other methods such as Visa card, MasterCard, Paypal etc. The Klarna services integrate with customer’s website through an API, which provides integration APIs for different programming languages. In order to use Klarna services, the end-user chooses Klarna as the payment method. Following is a step by step work-flow describing how system works in Klarna:

1. A customer of an e-commerce store adds the products in cart and at the checkout chooses the Klarna as payment option.
2. It registers the end-user through unique identity, such as social security number.
3. Verify the credit risk, checks the financial history of the customer, i.e. the customer is not black listed, age factor or not defaulter etc and create the reservation.
4. When e-store packs the products for delivery the reservation is activated. The Klarna Online system generates an invoice number, which can be used to track the purchase order.
5. Klarna pays the invoice amount to the e-store and presents invoice to the end-customer.

The end-customer receives the product along with the invoice from Klarna. The end-customer is then expected to execute the payment to Klarna in some specified time.
Klarna is dealing more than 20,000,000 transactions every year where each transaction have an average value of 1,000 SEK. The transactions are performed in real-time with the latency of 3 seconds by means of XML-RPC. Further, the availability goal is zero planned downtime, and less than a minute downtime in case of hardware failure.

3.2 Business Risk

The study shows that each year the average medium-sized company experiences 16-20 hours of system downtime [8]. This may happen due to the network, system or application downtime. This is very critical when it comes to financial institutions such as banks and factoring companies. Outages happening throughout business hours result in income loss due to loss of orders that in turn lead to problems such as unsatisfied customers and employees unable to access the system. The study also shows that with adapting best practices, these problems can be minimized to the following extent as depicted numerically in Figure 2.1.

![Figure 2.1 the impact of downtime on business](image)

This can be achieved by improving the reliability of the system. Erlang is well-known for developing reliable and highly available applications. Klarna is one of the leading companies that are enjoying
the benefits of Erlang used in their systems. Erlang was the first working version and was up and running within few months of time.

### 3.3 Erlang

In early 1980’s, Joe Armstrong, Claes Wikström, Robert Virding and Mike Williams in Ericsson were assigned a task to investigate that what programming language can overcome the problems mainly related to fault tolerance and highly availability of telecommunication products.

In 1986, Joe Armstrong came up with Erlang which was the first version ever developed by him. Initially it was property of Ericsson but after 10 years, in 1998, it was released as open source. Nowadays, many companies are using Erlang not only in telephony sector but every kind of business sites such as Amazon, Yahoo and Facebook. The whole chat system of Facebook is also developed in Erlang.

In short, one can ask: “why Erlang” Simply because the problem domain perfectly fits the solution provided by Erlang. Erlang is a functional concurrency-oriented language with extremely low-weight user-space "processes", share-nothing message-passing semantics, built-in distribution, and a "crash and recover" philosophy proven by two decades of deployment on large soft real-time production systems.”[7].

### 3.4 Erlang’s characteristics

Erlang is a functional language to develop scalable, fault-tolerant and highly available applications. The feature of concurrency also makes it one of the best choice for large applications. The processes do not interfere with each other as they have their own memory space and communication between them is done thorough message passing.

It has very light weight processes as comparable to threads and processes of operating system and other languages. Its VM [9] does not create an operating system thread for each process and they are created and scheduled in the VM without interference of the underlying operating system.
The variables can be bound only once in the program that makes it much easier to recognize and predict the behavior. In Erlang, pattern matching is used to bind values to the variables, extract the values from compound data types and control the flow of execution of programs.

It has ability of hot code swapping, which means that code can be changed during the execution of the application without having any planned outage [10]. This is of great use in zero-downtime systems [11] such as air traffic controller system, telecommunication and banking sector etc., where the system cannot halt for an update.

Compared to other languages, Erlang has simple but powerful error-handling mechanisms. Its efficient error-handling mechanism and exception monitoring constructs, which are used to build general library modules with robustness designed into their core. With programming for the right case and letting these libraries handle the error, makes the program shorter, easier to understand and having less bugs.

Erlang has distribution built-in into the language’s syntax and semantics, allowing systems to be built with platform independence in mind. This makes it easier to develop distributed systems and port applications written in single core to multi-core processor-based network without any complexity. The default distribution is based on TCP/IP, which allows a node or Erlang runtime system on a heterogeneous network to connect to any other node running on any operating system. If the nodes

---

**Figure 2.2 Process creation times**

The variables can be bound only once in the program that makes it much easier to recognize and predict the behavior. In Erlang, pattern matching is used to bind values to the variables, extract the values from compound data types and control the flow of execution of programs.

It has ability of hot code swapping, which means that code can be changed during the execution of the application without having any planned outage [10]. This is of great use in zero-downtime systems [11] such as air traffic controller system, telecommunication and banking sector etc., where the system cannot halt for an update.

Compared to other languages, Erlang has simple but powerful error-handling mechanisms. Its efficient error-handling mechanism and exception monitoring constructs, which are used to build general library modules with robustness designed into their core. With programming for the right case and letting these libraries handle the error, makes the program shorter, easier to understand and having less bugs.

Erlang has distribution built-in into the language’s syntax and semantics, allowing systems to be built with platform independence in mind. This makes it easier to develop distributed systems and port applications written in single core to multi-core processor-based network without any complexity. The default distribution is based on TCP/IP, which allows a node or Erlang runtime system on a heterogeneous network to connect to any other node running on any operating system. If the nodes
are connected through TCP/IP and firewall is properly configured, the product is a fully meshed network of nodes, where all nodes can communicate with each other.

Erlang also allows us to integrate legacy code, which is no-longer supported in different languages. It can communicate with nodes running Java or C and appear as Erlang nodes.

### 3.6 Erlang web development

Erlang was primarily developed focusing on the telecom industry. It had to satisfy some of the requirements of the telecom applications, such as distributed, massively concurrent, highly available and real-time applications. After the invention, Erlang was soon used for development of web-based applications. By using the OTP library, Erlang has reached a high level of maturity in the telecom marketplace but when it comes to languages for web development, Erlang is still considered to be in the stage of early adopters.

The powerful concurrency model makes Erlang special for web applications development. It allows a constant throughput of dynamic pages under extremely heavy loads [13]. In one of Joe Armstrong’s presentation [14] he showed, the amount of parallel connections needed to crash the Erlang web server was about 20 times as many as an Apache web server running on the same hardware.

The first tries of Erlang to overpower the web development community was back at 1997, when Ericsson developed its first web server [12]. The first web server was called INETS and it still comes with Open Telecom Platform OTP. The code for INETS was very small and it is only based on 10,000 lines of code and achieved 80% functionality of the Apache server which consisted of about 100,000 lines of code.

![Figure 2.3 Web development architecture](image)

### 3.6 XML-RPC

A remote procedure call is a method to call function of any module on a remote node and collect the information. It is mainly used to collect information on remote node, or for running a function with some specific side effects on the remote node.
XML-RPC [15] is a remote procedure call protocol that uses XML to encode its calls and then use HTTP protocol for transmission of data. It uses the structure which was originally developed for communication between humans to support communications between programs and computers. The client XML-RPC works by sending a HTTP request to the server implementing the protocol.

The client is mainly uses a software to call a specific method of the remote system. One or more parameters can be passed to the remote method call, one return value is returned. It allows sending nested parameters in the form of maps and lists and therefore, larger structure can be transported.

3.7 QLC

Klarna uses Mnesia database which is distributed, soft real-time database management system and also written in Erlang language. The Query List comprehension QLC[16] offers a query interface to Mnesia, ETS, Dets and to other data structures. It also provides support for user defined tables. A query is declared through Query List Comprehensions (QLCs) and the answer to a query is determined by a data in QLC tables that perform that constraints conveyed by the QLCs of the query.

It is similar to ordinary list comprehensions except the variables introduced in patterns cannot be used in list expressions. In ordinary list comprehension the lists are evaluated by calling qlc:q/1,2 which returns a Query Handle. In order to obtain all the answers to a query, qlc:eval/1,2 will be called with the query handle as first argument.

Using QLC might be expensive than Mnesia functions but It comes with nice syntax. As ordinary list comprehensions, QLC also have the same syntax:

```
[Expression || Qualifier1, Qualifier2 …]
```

Where Expression is an Erlang expression and Qualifiers are either generators or filters. Generators are Erlang expressions having form Pattern <~ListExpression, where ListExpression is an expression that evaluates to a query handle or list and Filters are also Erlang expressions which return the Boolean result as bool().

3.8 The Klarna system

The Erlang is the main language used in Klarna. This is because Erlang provides most of the features that a financial institute should have such as:

1. As Klarna, handles thousands of invoices every day and as its growth is gradually increasing therefore they wanted to have language which allows multi-core processing which leads to high performance.

2. In case of system failure the language should recover the system itself automatically. Since Klarna handles financial transactions which turned to be direct loss when the system is down. Erlang provides a mechanism where it recovers itself after the crash without causing the system to stop.

3. Hot code swapping that allows the upgrade of the system without stopping it.
4. Due to distributed property of the Erlang, it is easier to handle huge amount of transactions.

3.9 Klarna system architecture overview

Erlang fits into the above mentioned requirements. Along with Erlang, Klarna uses Mnesia database as a main database. Mnesia is a distributed and highly available key-value based database, which means tables can be configured within a schema and relocated between nodes. Klarna online system is roughly four tier application. From down, the first layer is Data layer which take care of the data that is stored in Mnesia. Next layer is the database abstraction layer on top of Mnesia.

![Figure 2.4 Klarna system architecture](image)

Business layer contains the main part of the business logic of the system. There are two different APIs inside Application layer and is used depending on who is using the system. System to system communication is done through XML-RPC API where as human to system communication is done through GUI API.
Chapter 4

Problem analysis

This chapter gives the brief detail of the problems and requirements. It starts with describing the problem in detail. Next, it explains how credit system works in Klarna. Finally the requirements are defined in detail.

4.1 Background

Klarna is the invoice factoring company. It buys the invoices and pays the amount immediately to the sellers. Consequently after selling the products, sellers get payment straight away rather than having to wait for buyer to pay the invoice. Klarna buys the invoices from clients that mean it owns the ownership rights of risk and receivables [4]. Klarna cannot obtain payment from the seller incase buyer do not pay to it, hence the importance of credit risk prediction is remarkable.

Like most of the financial companies Klarna also offers web-based system which is integrated with e-commerce shops. In order to pay through Klarna, users select the Klarna as payment option and Klarna system checks for the financial health of the client. The normal latency time for the transaction is 3 second therefore, decision to check whether to allow or reject the purchase must be made instantly at the time of purchase attempt and for this, the system for predicting the credit risk has to be present. This means, it immediately checks the request according to already written rules to make sure that all invoices fulfill the criteria defined in the rules.

To deal with this problem, Klarna uses large number of resources to study and monitor the credit risk. It runs a risk department whose job is to verify the worthiness of the buyers through tuning the policy rules. These rules can be tuned offline and cannot be overridden by manual intervention at the time of the purchase as we do not have time at the time of transaction.

These policy rules are often called credit policy. A "credit policy" is predetermined plan of action that is established to guide and monitor toward allowed business goals and strategies. Credit policy is also defined as the strategic link between day-to-day operations and the vision of the company.

Considerable resources and complicated program are required to analyze risk. Klarna utilizes its own credit policy system to grade potential and existing customers according to risk. The idea is walking the fine line of maximizing the acceptance rate, while minimizing the credit losses. It is very difficult to optimize the policy rules because every decision based on this policy turns into either revenue or loss.

4.2 Klarna credit policy

In the beginning, all logic for credit policy was written inside the source code. Since Klarna is growing very rapidly, it became a problem to understand and enhance the hard coded logic.
With the passage of time, following problems become more noticeable:

1. The policies could not be generalized. That is due to the specific requirements of larger customers or marketing departments.

2. As the rules are written in Erlang language, it was not possible to edit a person having no knowledge of Erlang.

3. Since the rules are mixed with source code, it was difficult to modify the rules.

Developers at Klarna decided to develop a new system. After months of effort, the new credit policy system was developed. The policy system is separated from the source code and written in DSL [6]. This new system resolves the following major issues:

1. Since the rules were separated from the source code therefore, it was easier for any person having no knowledge of Erlang to understand it.

2. It was now possible to customize the policy to the specific country or e-commerce store.

3. The policy engine returns the JSON type result in form of Grant/Reject with reason.

With the invention of new policy system, few milestones had been achieved. It accomplishes the customized requirements of clients and from development point of view; it is lot easier to modify the rules.

![Figure 4.1 Klarna policy system](image)

The policy engine mainly is responsible for running credit policy. It runs the parser and interprets the policy (Media Wiki file) and returns the reply based on the values provided in the call. If the credit information is not available, policy engine runs the policy data engine get it from the external lookup companies. This information would then be stored in cache for further use.
4.3 Problem description

Although new policy system made system very efficient but in order to visualize the instant effects of the transaction was still not possible. In order to have any change in the policy, Risk department must need to communicate with IT development department which commit the change and usually this back-en-forth communication take a lot of time. Moreover, it also takes time for development department to modify the policy system since they need to reschedule the tasks and their priorities during the sprint.

Further, when doing tuning of the rules, the effect is delayed. It will not be visible whether a rule change is profitable or not until several weeks after the change is made, because it is not until then that you can start to suspect that a purchase is a loss. Furthermore, when the effect is visible, it is hidden among effects of other variables, marketing campaigns of Klarna or the clients of the Klarna.

4.4 Requirement analysis

The Risk department wanted to have a controlled environment where it is possible to modify the rules on the test bases and gets the instant results. Moreover, they also wanted to visualize those results in graphical format so that it can be presented to higher management who has no knowledge of specific rules.

4.5 Functional requirements

1. To save the live credit policy with different names and modify it.
2. An editor to modify policy rules.
3. It should be possible to add, modify or remove the policy rules.
4. To fetch the test persons from database by setting different parameters.
5. To get all test persons of purchases and purchase attempts from database.
6. Compare the effect of the new structure of rules to live policy.
7. Get some key performance indices from the system to measure the effect. A graphical representation of the following results.

- Number of purchases (#)
- Number of purchases per e-store (#)
- Purchased volume (Specific country currency)
- Acceptance rate (%)
- Rejection reasons
- Average ticket size (Specific country currency)

8. When the new rules have been tested, it should be possible to inform IT department so that they can be entered in live policy.
4.6 Non-functional requirements

1. Ensure that the end user interface is easy to use and manage.
   a. Easy to navigate
   b. End user interfaces are of a high quality design and consistent throughout the whole application.

2. Proper client/server side validation of all modules.

3. Make a use of open source software for development to keep the cost low.
Chapter 5

System architecture

5.1 Structure of the application

In order to develop the web-based application in Erlang, it is needed to use an Erlang web framework that builds the HTML pages over the Erlang structure. Since the architecture of the application is developed in Erlang therefore, I searched for applications that are suitable for this kind of application.

Few years ago, there were only few Erlang frameworks available but as Erlang is getting popular, there are several frameworks available for Erlang. Following is a comparison of different frameworks [2].

**Comparison of different Erlang web frameworks**

**FEATURES**

<table>
<thead>
<tr>
<th>FEATURES</th>
<th>BeepBeep</th>
<th>Chicago Boss</th>
<th>Erlang Web</th>
<th>ErlyWeb</th>
<th>Nitrogen</th>
<th>Zotonic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARCHITECTURE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event-driven</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Data Model</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>View/Controller</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(web)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View/Controller</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>(email)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DEVELOPMENT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Admin interface</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Automatic recompile</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Framework for</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>functional tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BeepBeep</td>
<td>Chicago Boss</td>
<td>Erlang Web</td>
<td>ErlyWeb</td>
<td>Nitrogen</td>
<td>Zotonic</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>--------------</td>
<td>------------</td>
<td>---------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>PRODUCTION</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error logging</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>BeepBeep</th>
<th>Chicago Boss</th>
<th>Erlang Web</th>
<th>ErlyWeb</th>
<th>Nitrogen</th>
<th>Zotonic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MORE FEATURES</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA MODEL</td>
<td>n/a</td>
<td></td>
<td></td>
<td></td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Code generation</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Data validation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>EDoc generation</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>BeepBeep</th>
<th>Chicago Boss</th>
<th>Erlang Web</th>
<th>ErlyWeb</th>
<th>Nitrogen</th>
<th>Zotonic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I18N/L10N</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic I18n facilities</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Automatic translation w/ Google Translate</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>BeepBeep</th>
<th>Chicago Boss</th>
<th>Erlang Web</th>
<th>ErlyWeb</th>
<th>Nitrogen</th>
<th>Zotonic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JAVASCRIPT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSON generation</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Built-in Comet</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>JS form validation</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DATABASES</td>
<td>BeepBeep</td>
<td>Chicago Boss</td>
<td>Erlang Web</td>
<td>ErlyWeb</td>
<td>Nitrogen</td>
<td>Zotonic</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>--------------</td>
<td>------------</td>
<td>---------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>CouchDB</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Mnesia</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MySQL</td>
<td>✓</td>
<td></td>
<td>-</td>
<td>✓</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>✓</td>
<td></td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tokyo Tyrant</td>
<td>✓</td>
<td></td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TEMPLATES</th>
<th>BeepBeep</th>
<th>Chicago Boss</th>
<th>Erlang Web</th>
<th>ErlyWeb</th>
<th>Nitrogen</th>
<th>Zotonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErlyDTL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>ErlTL</td>
<td></td>
<td></td>
<td>-</td>
<td>✓</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>WPart</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Nitrogen records</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONTAINER</th>
<th>BeepBeep</th>
<th>Chicago Boss</th>
<th>Erlang Web</th>
<th>ErlyWeb</th>
<th>Nitrogen</th>
<th>Zotonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>EWGI</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SimpleBridge</td>
<td></td>
<td></td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Webmachine</td>
<td></td>
<td></td>
<td>-</td>
<td></td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SERVERS</th>
<th>BeepBeep</th>
<th>Chicago Boss</th>
<th>Erlang Web</th>
<th>ErlyWeb</th>
<th>Nitrogen</th>
<th>Zotonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inets</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Misultin</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Mochiweb</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Yaws</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
</tbody>
</table>
I found Nitrogen to be the most suitable framework for this purpose and one of main reason is its facility of rapid development with event-driven approach. Nitrogen also allows us to use a variety of web servers with it. I used Mochiweb with Nitrogen for my thesis work.

5.2 Nitrogen

Nitrogen comes with all the power of Erlang to web development, including stability, hot code swapping and scalability. There is couple of characteristics which makes Nitrogen the first choice for web development in Erlang. The Nitrogen framework is a light weight Erlang framework. It can site on top of different web servers such as Inets, Yaws and Mochiweb. It provides a simple, fast and event-driven platform for developing dynamic AJAX-based applications. Following are some of the main characteristics:

1. It uses event-driven approach built on top of the pattern-matching ability of Erlang. On server-side, you can tag an element with any Erlang term and then act on the tag for different events.

2. In Nitrogen, with one line of code, you can use AJAX that allows updating or even replacing the content of the section.

3. By using the power of JQuery-UI, Nitrogen allows you to develop complex interfaces such as dragable, droppable and sortable features with very short code.
4. Nitrogen also allows one-way powerful data binding.

**Figure 5.2 Nitrogen with Mochiweb**

5.2.1 Nitrogen page

Nitrogen calls the main() function in every page that returns an instance of the record. In template record, the field “file” points out the HTML based template page. It is the file that contains the HTML code mixed with Erlang.

```erlang
%%...%template reference
main() ->
    #template { file: "templates/kj-template.html" }.
```
5.3 Mochiweb

Mochiweb server is open source, event-driven and high-performance HTTP web server. Since Nitrogen supports Mochiweb, it is very easy to configure it with Nitrogen. All you need is to set variables in Mochiweb configuration file according to your application.

```
%%% -*- mode: erlang -*-
[{mochiweb, [
    {bind_address, "0.0.0.0"},
    {port, 8007},
    {server_name, nitrogen},
    {document_root, "/lib/k2/static"}
]}].
```

Figure 3.1 Mochiweb configuration

5.4 ErlyDTL

ErlyDTL is an Erlang implementation of the Django Template Language. The ErlyDTL module compiles Django Template source code into Erlang bytecode. The compiled template has a "render" function that takes a list of variables and returns a fully rendered document.[2].

Nitrogen is perfect for random development of web applications but it lacks a good template engine. Nitrogen does not cover the all HTML elements therefore; ErlyDTL can be used for complete solutions.

Although Nitrogen 2.0 is updated with some more element attributes, still it does not provide full support for HTML elements. For some elements, it is not even supporting attributes of the HTML tags. In Nitrogen, every HTML tag used as records i.e., <Div> will represent #panel{id=ID, title=Title, body=[]}.

For example, In Nitrogen record #link represent the HTML tag <a>. To trigger the <a> tag, we need “rel” attributes which is missing in the #link record therefore, we need to use some template engine that renders the Erlang code in HTML.

In order to do this we need to create the “.tpl” file.

```
<a rel="{{trigger}}" href="#"><img src="{{src}}"></a>
```
We can then use it in our Nitrogen page that calls the “render” function of the templates module of the ErlyDTL to render it in HTML code.

```
Tpl = [{trigger, "#mies1"}, {src, "/images/help.png"}],
templates:render_template("a", Tpl),
```
Chapter 6

Solution description

This chapter describes the solution that I came up with regarding the policy system analysis. The chapter starts with explaining the architecture of the new system and how it works. Finally it illustrates the reports that are generated from the system.

6.1 Overview of the system

All the requirements that I got from Risk department belongs to JSON type object. The main theme of this project is to run different policies on the previous transactions that are saved in database, and visualize the results with graphical representation so anyone can instantly analyze them that occur due to change in the policy.

For further reference in the report, I will refer the name of my application with KLAPAS (Klarna Policy Analysis System). Following is a step by step description of the system in general.

1. The KDB is the main database that contains the data related to transactions. It contains the information of purchases and purchase attempts (rejections). We fetch and save these particulars through our application.

2. The user selects the policies from the database that will be applied on the data.

3. Through RPC calls, it executes the policy engine on the kred application.

4. The KLAPAS filters the information which is needed for future use in the reports and saves it in session variables.

5. The reports, which are using Google chart API, represents the saved data for visualization.

6.2 System on sandsik

The synchronized copy of the Klarna online system is running on the machine, called Sandsik. On sandsik, a replica of Klarna online is running with the name of kred. On the kred, policy rules, a wiki file, is updated everyday while the database is scheduled to sync every week since it is growing very rapidly and takes couple of hours to synchronize with Klarna’s online system.

The policy engine is always running on kred. KLAPAS runs beside kred and communicates with it to execute the policies on the policy engine. It communicates with kred through RPC calls. Each time, when KLAPAS runs the query, the call goes to kred on sandsik which fetches the required information that includes values of PD and DB variables. After this, it sends these values along with
the policy to the main policy system, kula, for JSON type result. The return object from kula is then stored in session variables and used later for reporting purposes.

6.3 Communication through RPC calls

Since the kred and KLAPAS have different virtual machines there is need of a channel for communication between them. Erlang provides a very good way of RPC calls for two programs executing on different address space. Figure 5.1 shows the inter communication between two applications.

![Figure 5.1](image)

The RPC call has the following structure

\[
\text{Rpc:call(Node, Module, Function, Args)} \rightarrow \text{Res | {badrpc, Reason}}
\]

Node = kred@sandsik

Module = Function = atom()

Args = [term()]

Res = term()

Reason = term()

6.4 KLAPAS

In KLAPAS, web pages are developed using Nitrogen web framework. There are four web pages. In Nitrogen application, every page has the (.erl) file and each time when application runs, it compiles the pages and creates binary files into the ebin directory.
When the page renders in HTML, Nitrogen calls the main/0 function of the page that returns record (#template) which points to the html for that page.

6.5 Application modules

In Nitrogen framework, every webpage contains an Erlang file (.erl) with the same name as the webpage. In every page, there are some modules that work at the backend and HTML web pages that are used for the interface of the application. Following are the details of the backend modules and pages:

![Figure 6.2 KLPAS](image)

6.6 Web pages

There are three pages: Policymaker, Query Analyzer and Reports. Following is the detail of these pages:

1. Policymaker module: where one can add, edit and delete the policy. By default, live policy is used as the template.

2. In Query module, user enters the data, sets the parameters and fetches the results. The fetched data is then stored in the session variables.

3. Reports represent the data on the bases of stored data.

6.7 Policymaker

The policymaker module is responsible for creating, editing and deleting the policies. There are two sections of the page, one is the control panel that contains list of the policies and operations that can be done on them, i.e., Add, Edit and Delete. The other section contains the textarea field which is used to edit the policy.
Following are the steps to make a new policy and edit it.

1. Click on the Add button that makes a copy of the policy which is taken from the kred system on sandsik machine.

2. Enter the policy name, and save it. A new policy will be added.

3. Select the policy and click on Edit button. This will show another panel below it where we can view the policy rules and edit them.

4. To delete a policy, select the policy and press Delete button. The policy will be removed from the system.

The RPC calls are responsible for fetching the copy of the policy and the list of the saved policies. In all Nitrogen pages, validation is implemented through custom validators.

6.8 Query analyzer

The Query analyzer is used for getting data from database and uses it to fetch reply from kula. In Query analyzer, user can get test persons from the database and then stores them in the session variables, which would then be used for reporting purposes.
Nitrogen uses the AJAX-UI accordion. In this accordion, there are different sections which deal with different parameters of the query. The result of the page is QLC query, Query List comprehension, which is then executed on the Sandsik machine by using RPC calls to fetch the data from database.

6.8.1 Example

Suppose, Risk department wants to fetch all invoices and amount of the goods, where amount is less than 1000 SEK, and compare it with two different policies. In relational database the query would be:

```
“Select invno, goods_amount from invoice where goods_amount<1000 and currency =sek  AND date>= ‘2010-11-10’ and date<=’2010-11-15’
```

This will be converted to QLC as shown under:

```
[{V#invoice.invno, GoodsAmount(V)} || V <- kdb:table(invoice), GoodsAmount(I) <GoodsAmount(10000)]
```

Following are the sections that create the query:

- Policies
- Date Parameters
- Table selection
- Projection (Selection variables)
- Selection(Filter variables)

6.8.2 Policies

This is the list of all policies along with the live policy running on the sandsik system. All policies which are created by user would then be shown in the list.

6.8.3 Date Parameters

Here we set the basic parameters for the query. The date parameter, for instance can be set through this option to generate the query.
6.8.4 Table Selection

There are two tables available for making queries. We can choose either purchase or reject for this purpose. The purchase table will be used if we want test persons for purchase reporting and similarly, reject table would be used for reporting related to rejection.

After selection of the tables the autotext field of Selection variables and Filter variables will be filled with all the elements of the table.

6.8.5 Selection variables

The selection variables are the variables which we want in our query as a result. The result is presented in the form of HTML and these variables are not for further use in reports. In order to
apply these variables, we need to select the checkbox of variable. If not selected, the variable is not considered in the query.

The Nitrogen framework provides \texttt{wf:qs(ElementID)} function that gets the values of all the elements which have the id similar to ElementID. This function will return the values of all selected checkboxes as the ID is same for all. If not selected, it will not be considered in the query.

\begin{figure}[h]
\centering
\includegraphics[width=0.6\textwidth]{figure6.7.png}
\caption{Selection/Return variables}
\end{figure}

\subsection*{6.8.6 Filter variables}

The filter variables are used to filter out the query. It is similar to the WHERE clause in relational databases where we put conditions for the query. Following are the steps to choose elements and set their arithmetic operations and values:

1. Choose the variable name from auto fill text box, a text box that provides the intelligence and helps out to find the variable from variable list.
2. Choose the arithmetic operator (\textgreater, \textless, =, /=) for the variable.
3. Enter the value for the variable.

By using AJAX, Nitrogen provides the \texttt{wf:replace [X]} function to replace the textbox of value to the particular element type of the table. There are various types of elements and they are described in the backend module. For example, if the element type is Boolean the textbox will be replaced by combo box with true/false options and if the variable type is datetime the textbox will be replaced by datetime picker. Similar to selection variables, filter variables are also considered when their checkboxes are checked.
6.8.7 Run the query

After the selection of all parameters, the query is ready to be executed. There are two different tasks that are performed at the backend. In next section I will explain the technical details that how it works at the backend.

1. QLC fetches the data from database.
2. It shows the Selection variables in HTML format.
3. The set of persons that we get after the query would be used to run policy engine to get the JSON based output.
4. The value is then stored in the session variables for further usage in reports.

6.8.8 Data for reports

Since the data of each test person will go through the system. The query result, which is stored in session variable, is used to fetch the detail of every test person by using `fun` function. Following is the detail that shows how function is executed to get data from kula.

1. Get the PD, DB variable values from kula.
2. Call policy engine in Kula to get JSON based result of the query.

We can get the value of PD, and DB variables as now we have the personal number and StoreID. In order to get data about the person, the system sends every person his personal number, PD, DB variables, e-store_id and the policy to the policy engine.
6.9 Reports

There are number of reports that can be generated through this application. The reports are based on the stored data that has been fetched during in Queryanalyzer page. Google chart API is used for reports. Following are some of main reports:

![Figure 6.10 Acceptance ratio](image)

Figure 6.10 shows the comparison of live policy (which is currently running in the Klarna online system) and another custom policy. In live policy the acceptance rate is 30% whereas in custom policy it increases to 70%. This is due to change in rules in `cred.policy.lessstrictpolicy`. 
Figure 6.11 shows the purchase invoice comparison between two custom policies.

Figure 6.12 present the rejection reasons of two custom policies. Due to change of age in the policy rules the `cred.policy.age` got 90% invoices rejected.
Chapter 7

Analysis/Discussion

7.1 Shift of focus

While the original time was remain same, the actual focus of the work shifted a bit from what it was initially intended. It was anticipated that most of the work would be spent on developing an intelligent policy editor, where the value of policy rules should appear in the form of components such as Boolean option or dropdown box. Since this task is not important because of time constraints and moreover it is really easy to change the rules specified in wiki language therefore main focus then shifted to develop a system that graphically present the analysis of policies.

7.2 The design process

Furthermore the overall thesis, mainly writing a report, took longer than the expected time; this is because I went back to my home country for two weeks and started very late to start writing. It took some time to get momentum. Moreover, during the time I wrote the report, I was also applying for different jobs which led to many days going away since I had to go to interviews with different companies.

7.3 Development environment

The development environment is very good in Klarna, which enables me to concentrate on work without disturbance. The Master thesis students have their own offices for development. The system was IBM n-series with operating system Linux 10.4. The tools used for development are Erlang, Nitrogen and ErlyDTL.

7.4 Problems

Throughout the thesis the development was very smooth and I did not face any numerous problems except some tiny issues. It took me more time then what I expected to collect requirements from the Risk department to start development. Every Sunday night the system synchronizes with Klarna Online System and as this test server is a virtual machine it takes very long time to copy the database which most of the time made impossible to work on Monday. Particularly in the last month it becomes very slow as the test server become out of memory.
Chapter 8

Conclusion and Future work

The system investigated, designed and developed in this thesis was aimed to evaluate the policy system in Klarna AB, where the thesis is carried out. The result of the work is a system named as, KLAPAS, which is using majorly by the Risk department at Klarna.

KLAPAS makes it possible to collect different type of data and apply it on custom policies. KLAPAS uses the Nitrogen web framework for visualization of the policies. It is hosted on the machine inside the Klarna AB and uses Mochiweb server to serve Nitrogen web pages.

The Google charts API is used through Nitrogen for policy analysis purposes. The application fulfills most of the requirements that were mentioned earlier in chapter 3. However, it can be extended to enhance the functionality of the KLAPAS. At the moment the policy can be edited in rich textbox but it would be nice if it is possible to develop an intelligent editor to edit the only variables in the policy. It should be possible to run these queries on different estores.
References


[18] Hamadi Matoussi,