IN3405 Bachelor project
Conversion Ratio

Mark Oostdam
1174681

Onno van Paridon
1100211

August 27, 2009

Faculty EEMCS, Delft University of Technology
Delft, the Netherlands

Adyen Enterprise Payment Services
Amsterdam, the Netherlands

Abstract

Before a shopper converts into a buyer there are a lot of steps to be taken. In each of these steps a portion of shoppers discontinues their purchase and the remaining part finalises the transaction. The conversion ratio is defined as the number of shoppers that finalise their transaction in comparison to the number of shoppers that start a transaction, expressed as a percentage. Because Adyen handles the payments, it is an important part of the process.

Chapter 1 states the main goal of the project as to gather information about the course of the payment sessions of consumers buying at on-line shops and explains why off the shelf solutions like Google Analytics are not feasible. In this chapter three sub-goals are defined.

1. Live data feed for merchants.
2. Data for marketing material.
3. Form a basis for an article.

Next technologies that are currently in place are analysed. First the infrastructure of the Hosted Payment Pages (HPP) is discussed in Chapter 2 and how Skins can be used to customise the appearance of the HPP. Finally the difference in the payment flow between the Multi Page Payment (MPP) and One Page Payment (OPP) is explained as well as the information that is currently included in the log files.

In Chapter 3 the problem is analysed and the main question for the project is defined as:

"How to measure and present the Conversion Ratio on the Hosted Payment Pages?"

To answer this question the process is decomposed in five distinct steps.

1. Log the required data.
2. Make the logs machine readable.
3. Centralise the log information.
4. Store the data.
5. Visualise the data.

This decomposition is applicable for a generic logging framework. As such, the individual steps, which are treated in subsequent chapters, start with a generic part and end with a project specific part.

In Chapter 4 the flow of events within a payment session is analysed. With this information additional data to be logged, like the IP address of the shopper, is determined to produce rich statistics. Finally the validation of payment sessions through state machines is discussed.

Chapter 5 discusses various logging formats ranging from CSV files to directly logging to a database. These formats are compared as how useful they are when read by a machine to detect interesting patterns as well as how readable they are for humans in case of an error. Before a format is chosen, the file based formats are tested on how well they perform when being parsed. An optimal solution is to simultaneously log to a database for pattern recognition and log to CSV in case an error occurs.

With the logs available in a suitable format the information has to be gathered on a central system. Chapter 6 considers two possible centralisation strategies that can be used. After the

two strategies, file transfer and database replication, have been analysed, they are evaluated with regard to load balance, network capacity and security. Database replication is considered to be the preferred strategy to be used because it spreads the load more evenly over time and the mechanism is built into the database system.

Chapter 7 talks about how the data is stored once it is available on a central location. First a generic approach is discussed that can be applied to logging in general where a distinct split is made between the log4j part and the application specific part. Then a conversion specific approach is taken which handles data extension, aggregation and validation. Finally an additional aggregation step is discussed for performance enhancement.

As the statistical information is now stored the data needs to be visualised in a intuitive way. First several use case are discussed in Chapter 8. To be able to handle with more use cases a template framework is developed which can contain multiple chart types. Next the statistic types are discussed that enable specific calculations to be performed for a specific statistic and hide the data processing from the templates and chart types. After that the user interface and the structure behind it are examined. Finally the extendibility of the system is outlined together with performance enhancements.

Chapter 9 covers software quality and testing. Initially unit tests were developed to test an important building block of the system, the state machines. With the unit tests written a code coverage analysis is performed to see how well code is covered by tests. Because of some imperfections in the logging, not all logged sessions adhered to the state machines while they were in fact valid. For these sessions to be taken into account some shortcuts were added to the state machines. Regression testing enables these sessions to be used while still preserving the quality of the software by explicitly testing for these sneak paths.

In Chapter 10 the conclusions of Chapters 4 through 8, regarding the five distinct steps in the process, are consolidated. After that, conclusions are drawn about the three sub-goals as outlined in Chapter 1. Regarding the live data feed for merchants the project has been successful. Merchants that got a glance of the user interface during development were very enthusiastic about the possibilities the system gives them. The goal of providing data for marketing material also has been met. On multiple occasions during the development, sales employees used the statistical data and its graphical representations as a Unique Selling Point to show Adyens expertise in the payment process. During the project a basis has been formed for a newspaper article, as well as a scientific article, thereby fulfilling the third goal. The newspaper article concerns the conversion ratio of PayPal payments. PayPal does not perform as well as other (localised) payment methods in the Netherlands, while introducing higher costs. A scientific article can be written on the basis of the generic logging framework and the preparations that have been made during this project for this framework.

During the process, a broad range to topics was touched but not all could be researched within the scope of the project. In Chapter 11 several recommendations are made for topics that should be investigated further. These topics include the creation of a generic logging framework that can be used throughout the platform as well as system that allows merchants to upload multiple Skins and compare the results. Finally more research can be done on pattern recognition to improve the rate of successfully reconstructed sessions as well as the creation of a generic framework to measure performance throughout Adyen's platform on multiple levels. In order to be feasible, all of these frameworks should be highly scalable to cope with large amounts of data. They should also be generic enough to be applied on multiple levels, ranging from the application level to a platform level.

Bachelor committee:

Chair:	Ir. B.R. Sodoyer, Faculty EEMCS, TU Delft
University supervisor:	Dr. P. Cimiano, Web Information Systems, Faculty EEMCS, TU Delft
Company supervisor:	M. Lobbezoo MSc, Adyen

Preface

Online shopping is a fast growing business. For this sector aspects like usability and Search Engine Optimisation are very important to attract more visitors. Tools like Google Analytics and Google Optimizer are standard parts of their toolkits. A very important metric for these merchants is the conversion ratio: the number of shoppers that finalise their purchase in comparison to the number of shoppers that start the shopping process. Adyens specific expertise is the payment process. The metric that Adyen is able to provide is therefore the conversion ratio of the payment process. This insight will give Adyen an additional tool to use their expertise in increasing the number of transactions. In this project the framework is developed to enable detailed insight in the payment process and conversion ratio, visualised in an intuitive way this results in a tool that is unique in the payment industry.

Conducting this project at Adyen has been a great experience. Not only the subject matter itself has been proven to be very interesting, also the corporate setting and learning new technologies have been an exiting adventure.

We would like to thank all of our colleagues at Adyen for helping us out whenever we had a question and giving us pointers on how to tackle some problems. We especially would like to thank Maikel Lobbezoo for guiding us and taking the time to review our work. Last but not least we would like to thank Philipp Cimiano for mentoring the project and taking the time to travel to Amsterdam to view our progress on multiple occasions.

Mark Oostdam
Onno van Paridon
Delft, the Netherlands
August 27, 2009

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Adyen is a Payment Service Provider (PSP), started in 2006 by industry specialists. Adyen focusses on providing one online payment interface for the European and North America online market. The role of Adyen in the online payment process is enabling on-line shops (merchants) only having to connect to Adyen in order to support multiple payment methods, like credit cards and local payment methods like iDeal, bank transfers, et cetera. In addition to local payment methods international payment methods can also be made available to the merchants audience. With offices in Amsterdam, London and Boston (MA, USA), Adyen takes care of all the communication and money flows for the merchants for all the payment method providers (acquirers) and offers her merchants one unified type of reporting and money flow with one contract.

1.1 Business model of Adyen

Adyen's business model is based on providing innovative features to its merchants at a minimum cost. Adyen is a collecting PSP, meaning that Adyen holds the contracts with multiple financial institutions. These institutions fund the Adyen Client Management Foundation, which then distributes these funds to Adyen's merchants. As a result Adyen is under supervision by the Dutch

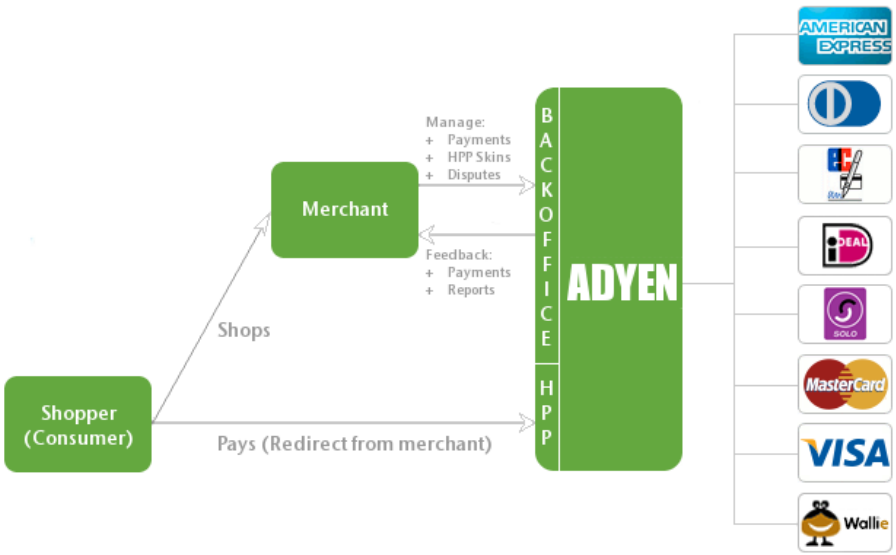


Figure 1.1: Adyen system overview

Banking Authorities. For the merchants the advantage of this model is that Adyen reconciles the payments (matching the incoming money with the outstanding authorised transactions) and provides one money stream and one type of reporting to the merchants.

The cost for the transaction are twofold for the merchants, a fee for the financial institution and the Adyen processing fee of a maximum of 10 cents per transaction. There are no monthly charges or setup fees for the merchants. The costs of the financial institution are transparent and charged directly to the merchant. Because Adyen has the contracts with the financial institutions it processes large volumes and has a stronger position in negotiating lower costs. Since Adyens margin is only the small processing fee, the only way for Adyen to be successful is to process large volumes of transactions. One of the benefits, besides providing a low price to the merchants, is that both the merchant and Adyen strive to increase their transaction volume. Giving insight in the conversion rates is not only an innovative feature, it also provides knowledge on how to increase the number of payments and thus generate higher revenues.

The shoppers are redirected from the merchant's website to the merchant's Hosted Payment Pages (HPP) hosted at Adyen's platform (Figure 1.1). Adyen is therefore determined in making the payment process as easy and intuitively as possible, reducing the level of abandonment to a minimum.

The payment system is part of the core of the merchant's business and as a result there are high demands on Adyen's payment platform. For this reason the platform consist of several stand alone applications which all run on their own redundant hardware. It can still process (and store) payments even if (external) components are unavailable. Leading to one of Adyen's feats, a platform that is scalable and has a 100% uptime guarantee.

However having a good architecture and robust applications are just the basis for offering a high Service Level Agreement (SLA) to an online payment platform. Logging and monitoring are of the utmost importance, since if problems occur the responsible person should be notified and fully informed instantly to handle the situation. The standard logging and monitoring tools have proven to not yet meet the requirements for these kind of situations. We will not take this problem up as a goal of our project, but we will take it into account when creating a solution for the conversion logging. As a result we will focus on a solution which meets all the requirements of the Adyen platform on the (application) log files.

1.2 Goals of the project

Converting website visitors to actual customers spending money at the merchant's site is the challenge of every merchant conducting business on the internet. The design of the site, ease of use of the shopping cart, data and payment entry pages should be no hurdles for the merchant's customers in finalising the sale. The more steps in the merchant's payment process, the less chance that customers will complete the payment successfully [22]. Payment pages that are difficult to understand, counterintuitive and hard to navigate are a major source for fallout which decreases the merchant's online revenue unnecessarily.

The goal of the Conversion Ratio project at Adyen is to gather information about the course of the payment sessions of consumers buying at online shops. As stated above not all of the initiated payment sessions will result in successful payments. Some consumers will abandon the payment pages prematurely and abort the payment session. Information about the completed intermediate steps, acquired by logging, can be used as source for statistical analysis. Results of the analyses can be used for several purposes. In this chapter we will discuss some of the possibilities.

Although there are off the shelf solutions like Google Analytics [11] to get insight into the behaviour of visitors and shoppers, these are not feasible to analyse the conversion ratio on the HPP. The problem with products like Google Analytics is that they send information back to servers of the producer. Because the HPP deal with highly sensitive information this practice is not allowed by Payment Card Industry (PCI) regulations. An additional problem is that the

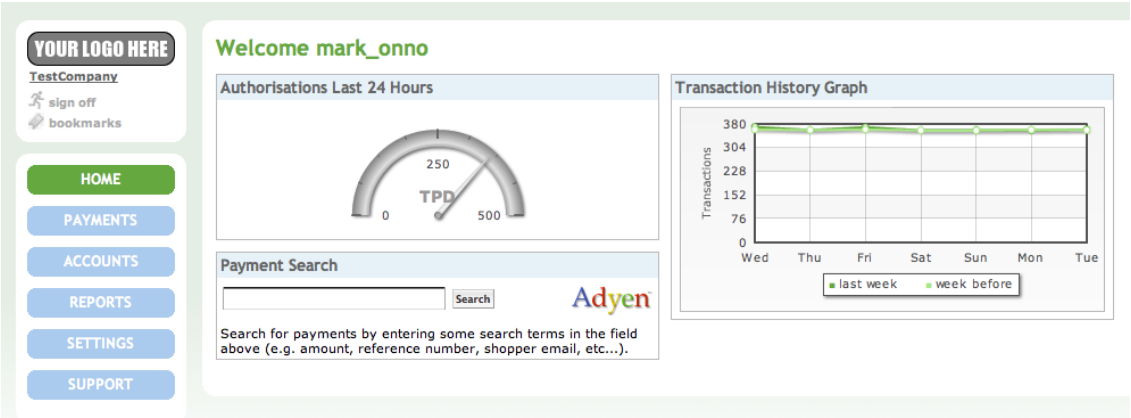


Figure 1.2: Dashboard interface of the Adyen merchant backoffice

information has to be available on both the PSP level as well as on a merchant level as explained in the following sections.

1.2.1 Live data feed for merchants

Adyen provides a merchant backoffice for their clients in which they can manage their account configuration and get live feedback as illustrated in Figure 1.1. Part of this feedback is a dashboard interface (Figure 1.2) where drag-able widgets enable the merchant to view live data about the payments that have been performed. One of the uses of the statistical information is to provide feedback to the merchant in the form of widgets. A possible widget would be a funnel diagram (Figure 1.3) showing the conversion ratio.

1.2.2 Data for marketing material

Marketing material should emphasise the unique selling points of a company. One of the selling points of Adyen is the partnership it forms with its merchants. Improvement in the levels of

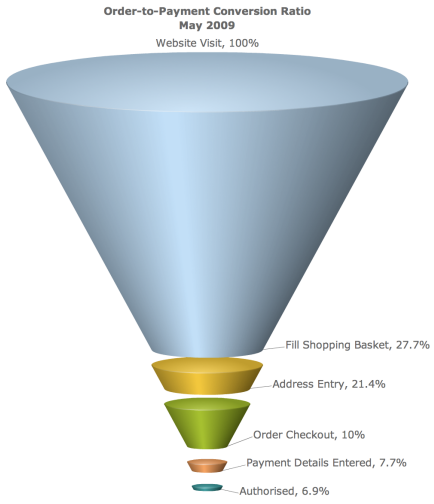


Figure 1.3: Order to payment conversion ratio

abandonment and key insights in consumer behaviour are of beneficial for both Adyen and its merchants. The information can therefore be used as a promotional tool.

1.2.3 Form a basis for an article

Adyen has an agreement with two Dutch newspapers to publish an article near the end of the project. Results of the analyses and the tests can be used to enrich the articles and promote Adyen. Furthermore a scientific paper can be made concerning the generic logging framework, for which the first steps are taken during this project. Such a paper would benefit the community while a wide range of companies could benefit from a logging framework that does monitoring, notification and reporting automatically.

Chapter 2

What's in place

When starting a new project it is always useful to look at what's already in place. In this chapter we will examine which technologies and systems are used in Adyen's system. The project needs to be integrated within Adyen's platform and its development process. Therefore technologies that are already used should be preferred. These technologies include Eclipse [32], Ant [14] and Subversion [7]. The following sections discuss the infrastructure, Skins, payment flow and log files.

2.1 Infrastructure

In order to achieve high availability the HPP run on a cluster of identically configured redundant systems as indicated by the simplified overview in Figure 2.1. Because of the statelessness of the HPP the system is highly scalable. These systems run the Apache Tomcat [4] web server to serve the pages to the shoppers. They hand off the requests to other machines in the system to do the risk analysis and the actual payment. These machines are represented by CS in Figure 2.1 and are compromised of multiple hardware systems running multiple applications.

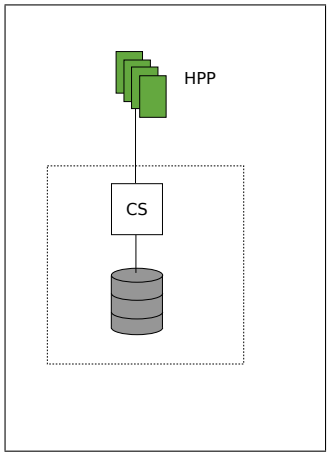


Figure 2.1: Simplified overview of infrastructure at Adyen

Adyens' whole system has been set up in a Service Oriented Architecture (SOA) [26]. This means that all the services that adyen provides are interchangeable and run independently (even on different hardware when needed). For example the HPP are services, but also the conversion analysis framework will be implemented into the system as a service.

2.2 Skins

The payment pages that are presented to the shoppers can be customised by the merchant through Skins (Figure 2.2). Skins are ZIP files that contain multiple image, HTML, CSS, Javascript and resource files.

Using the Skin technology merchants can not only adjust the look and feel of the pages to match their colouring scheme and have their logo appear, but they can also adjust which of the payment methods will appear and their order. Also the way of interaction can be altered by a Skin from a multi page payment, where several pages will be shown sequentially, to a one page payment where all the interaction will take place on a single page.

Because merchants can tailor these Skins entirely to their needs, the payment pages can completely mimic the layout of the online shop pixel by pixel.

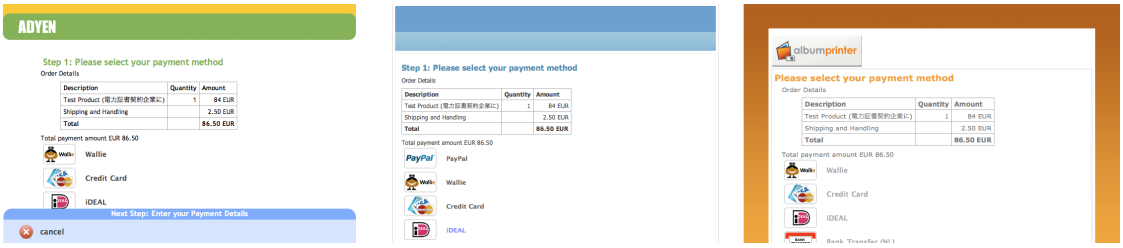
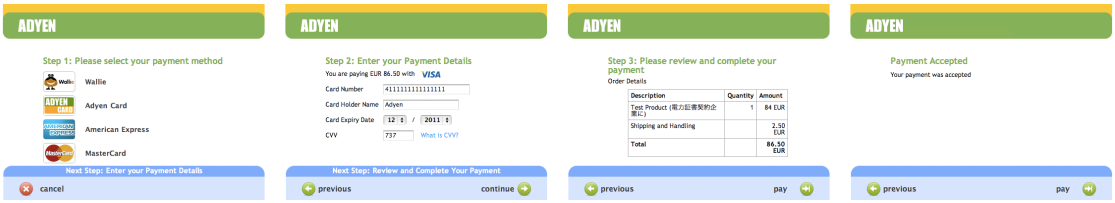


Figure 2.2: Examples of Skins for the HPP

2.3 Payment flow

As stated in Section 2.2 the Skins can define the way of interaction. When Multi Page Payment (MPP) is used, multiple pages are shown to the shopper in sequence to complete the transaction. The payment process consist of the following stages:



(a) using multi-page payment process

(b) using one-page payment process

Figure 2.3: Page flow of the payment process

- Payment method selection
- Enter payment details
- Review order details
- Result

Each of the pages in Figure 2.3a represents one of these stages.

When using One Page Payment (OPP) all the user interaction takes place on a single page. If the payment method support this, the first three stages are combined on one page as illustrated in Figure 2.3b. With this insight gained, the payment flow could be modelled more accurately. In Section 4.2.1 the result of this modelling can be found. The next Section 4.2.2 discusses the additional data needed for step {2} of the next implementation cycles.

2.4 Log files

In the current system logging is being performed by the log4j framework [3]. The main purpose for the application logging that is in place, is to detect errors and warnings in the application. Some additional information for the context is also logged to trace the errors and warnings. Besides this log the behaviour of the shoppers is recorded in a conversion log. All actions that result in a new page are currently logged except for error and warning pages. Adyen can therefore use these logs to distil the conversion ratio on the payment sessions that occur within the HPP.

Listing 2.1 shows the first 4 lines of a sample log file. The log file contains information from both the log4j framework and a message part defined by de application.

Listing 2.1: Sample existing logfile

```
1 2009-03-28 00:05:01,004 INFO [com.adyen.actions.hpp.conversion.
    ConversionLogger] {LogTimer} -- Something to roll the logs --
2 2009-03-28 01:18:46,786 INFO [com.adyen.actions.hpp.conversion.
    ConversionLogger] {TP-Processor2} "TUDelftWebShop","6seEWw90","dhqxPh+
    RqLGnzI3BCHMHAWfVrII=","1201408","2009-03-28T00:18:46.78Z",23499,"EUR","
    review","CompleteCardPayment","mc",,,
3 2009-03-28 01:18:46,786 INFO [com.adyen.actions.hpp.conversion.
    ConversionLogger] {TP-Processor2} "TUDelftWebShop","6seEWw90","dhqxPh+
    RqLGnzI3BCHMHAWfVrII=","1201408","2009-03-28T00:18:46.78Z",23499,"EUR","
    review","HandlePaymentAction","mc",,,
4 2009-03-28 01:18:54,039 INFO [com.adyen.actions.hpp.conversion.
    ConversionLogger] {TP-Processor2} "TUDelftWebShop","6seEWw90","8JRJbd0+
    QCZCzjrgl4M6ti7R+ww=","1201408","2009-03-28T00:18:54.03Z",23499,"EUR","
    details","PaymentDetails","mc",,,
```

Information supplied by the log4j framework is:

- The exact **DATE** in a unified format (a.k.a. timestamp).
- The severity **LEVEL** assigned to the message, for example INFO, WARNING, ERROR et cetera.
- The **CLASS** which is requested to perform the logging.
- The **THREAD** in which the logger is called.

Information provided by the application is:

- The **MERCHANT** requesting the payment.
- The **SKIN** used to customise the HPP. The code identifying the Skin is unique and generated at design time.
- The **SESSIONID** identifying the session is generated by a hash algorithm based on fields provided by the merchant.
- The **MERCHANT REFERENCE** is a reference provided by the merchant to identify the transaction.
- The **TIME** until the payment session is valid as specified by the merchant.
- The **AMOUNT** of the transaction.
- The **CURRENCY** used in the transaction.
- The current **STAGE** the transaction is in.
- The name of the **CLASS** currently handling the transaction.
- The **METHOD** used to perform the payment for example MasterCard, VISA, iDeal et cetera.
- The **RESULT CODE** of the transaction as issued by the acquirer, for example AUTHORISED, REJECTED et cetera.
- The **PSP REFERENCE** is a unique code identifying a transaction within Adyen's system.

The current log files can already be used to identify differences in conversion rate between payment methods. Analysing the conversion log can bring a greater insight into the behaviour of shoppers. For example, as stated in Section 2.2 Skins can influence the user experience. The results of the analyses might therefore indicate that some Skins have a greater risk of the shopper abandoning the payment prematurely. Particularly interesting to see for example is how the conversion ratio differs between MPP and OPP because less pages will be shown when using OPP as explained in Section 2.3.

Chapter 3

Problem analysis

Before a browser converts into a buyer there are a lot of steps to be taken. In each of these steps a portion of shoppers discontinues their purchase and the remaining part finalises the transaction. The conversion ratio is the number of shoppers that finalise their transaction in comparison to the number of shoppers that start a transaction, expressed as a percentage (Figure 3.1). Raising the conversion ratio would mean more shoppers finalise their transactions. Therefore the merchant's goal is to maximise the number of successful payments.

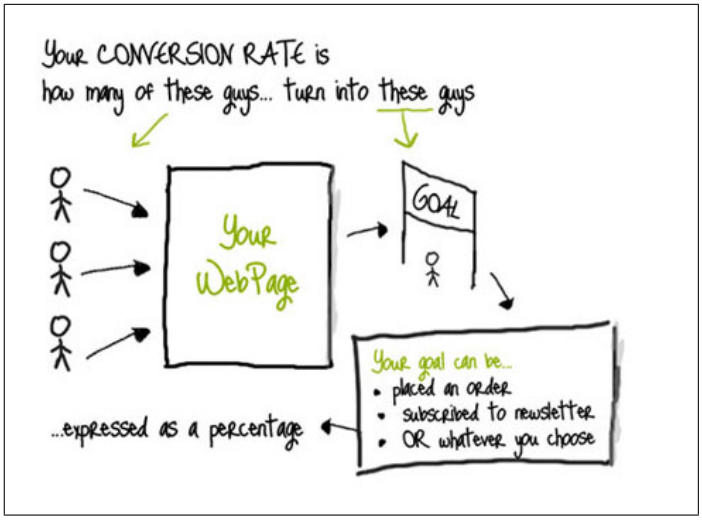


Figure 3.1: Conversion ratio explained [8]

When a PSP is used, the PSP takes over the payment process from the merchant. Adyen is interested in how the conversion ratio is affected by for example banners, menus, movement and animation, number of pages and difference in layout between the merchant's pages and the HPP. These are the main things that can be modified through the use of an Adyen Skin.

As stated in Chapter 1, Adyen's business model is based on charging the merchant a fixed amount per transaction. Greater insight in how the conversion rate is influenced by these factors and ultimately higher conversion rates mean that Adyen can provide a better service towards its merchants and a higher turnover.

The main question that will be answered in our report is:

"How to measure and present the Conversion Ratio on the Hosted Payment Pages?"

To answer this question the problem will be split in the five mayor aspects of this project. The following sections will digress on these aspects, starting with the analysis of the log files in order to determine what data needs to be logged, indicated by step {1} in Figure 3.2. Step {2} concerns the format in which the data can be stored. The gathering of the available information, step {3}, will be analysed in Section 3.4. After that we will look at how to store and archive the information on the central system, step {4}, and prepare analyses on the information. Finally we will examine how the results can be presented visually as indicated by step {5}.

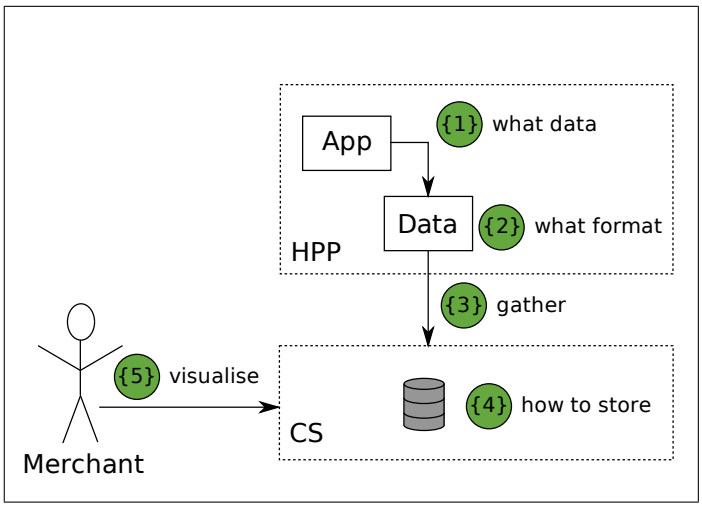


Figure 3.2: Overview problem analysis

3.1 Using the log files

As stated in Section 2.4 the log files contain various information. In order to see what information is present and how this can be used we built a simple parser to generate some statistics. A first problem we encountered was the occurrence of delimiters in fields as described in the paper by Reichle et al. [29]: *"A common mistake is to fall for delimiters within elements."*

The main question regarding the current log files is whether all necessary information is available. At the moment the IP addresses of the shoppers are not recorded in the log files. This information in combination with a DNS lookup as used in the system proposed by Reichle et al. (Figure 3.3) can enrich the logged information with the company name or the country name. This knowledge can give great insight into the demographic distribution of the shoppers. Other interesting external data which can enrich the conversion data will be researched.

3.2 Conversion logging as part of a generic logging framework

The current logging standard for Java applications is to use log4j to compose log files. While this provides basic functionality, Adyen still requires a developer on watch 24/7, to monitor the system and to extract the interesting data from the log files when a suspicious situation (warning) occurs. One part of the problem is that Adyen's platform consists out of several soft- and hardware components. Another part is that large sections of the log events, like sensitive payment details, are logged encrypted. If the person on watch notices that there is a technical problem, this person can resolve the situation, but often an infrastructure manager, sales representative, support employee or one of the suppliers (banks) needs to be contacted. Beside the cost of having a developer on

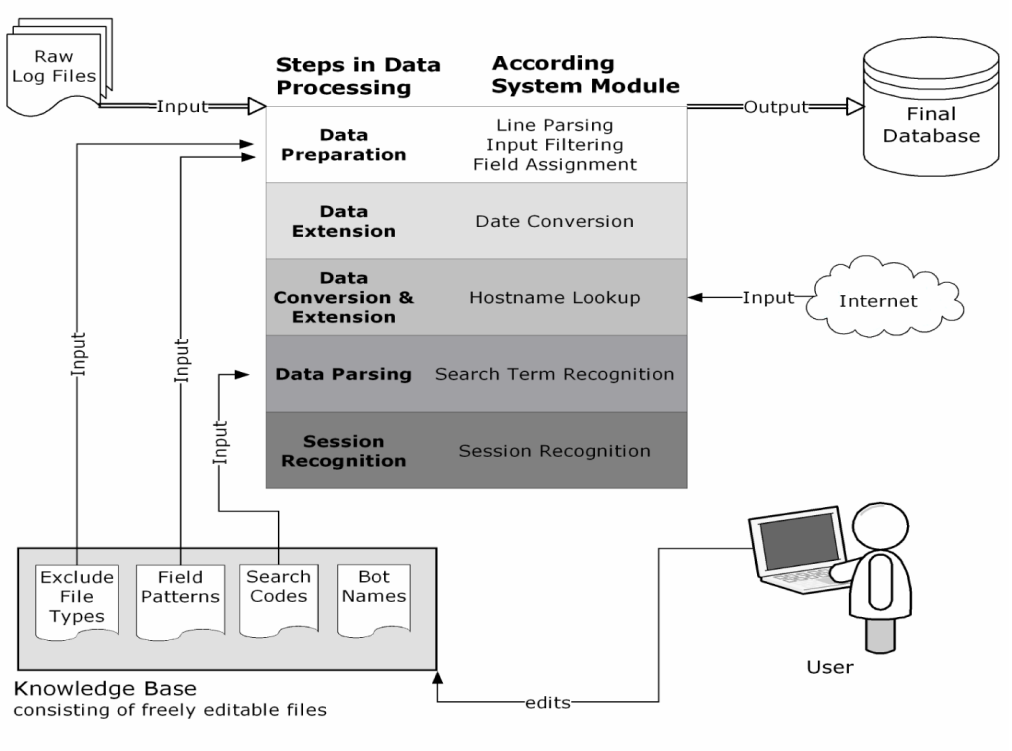


Figure 3.3: Overview of steps in data processing by Reichle et al.

watch, this process is time consuming and detecting an erroneous situation is still a labour intensive process. Solving a situation in the middle of the night based on incomplete or misinterpreted information may even lead to escalation of the problem.

Adyen is looking for a logging system that reports on the performance of the platform, monitors the applications and hardware components and notifies when suspicious or erroneous situations occur. There are many challenges for such a system. Reports should be available on a transaction, application, hardware and platform level. Third party events should also be included. The reporting features must of course be compliant with the security standards set by the financial industry. Reports should be easy and fast to access and interpret. The system should be able to monitor multiple applications and hardware instances in a platform that is able to handle thousands of concurrent payments 24/7. This automatic process should instantly detect suspicious and erroneous situations on both a high (platform) as well as in low (transaction) level. When a suspicious or erroneous situation is detected the responsible person should be notified immediately by email or sms with all required and available information. Ideally this notification would include indications of how similar situations were solved in the past and extra information gathered from automatically executed analytical jobs.

3.3 Make conversion log files machine readable

Currently the log files generated by the log4j framework are human readable. However they are not very comprehensible for human beings, there is simply to much data to get an idea of what is happening. Because of the high volume of data we need an automated way to parse and process the log information. Therefore we need to make the log data machine readable. In order to use the log data for statistical analysis we need to read it into a structured format. Although the

current conversion logs are parsable to some extent there are some difficulties associated with this approach. After writing a simple parser to see if we could parse the current log files and to try and get the first simple statistics we quickly noticed that the current log format isn't suitable for our needs. The possible occurrence of delimiters in fields and deviations from the standard format make the parsing hard. Thus our challenge is to put the logging data into a machine readable format so we are able to order and compare different payment transactions. A few formats we have already considered and will be investigating further are:

1. CSV,
2. XML,
3. directly to a database,
4. directly to a web service,
5. Java property files [30],
6. serialised Java objects.

3.4 Gather the conversion log data on a central system

The HPP applications run on a cluster of identically configured redundant hardware systems as explained in Section 2.1. Because of the statelessness of the HPP each of the intermediate actions (and their corresponding payment page) within the payment session may be handled by a different system. In order to perform useful analyses, the data of the individual systems has to be loaded onto a central system to get a complete overview of all the data.

Because of the high volume of log messages we need to take into account the available bandwidth between the systems and the processing power of particular systems. For instance we don't want the systems hosting the HPP to stall because they have to load the conversion data to a central system. We also need to consider the required disk space for the logs so the hard disks will not flood in these circumstances. The latter is a known issue since this has caused problems in the past.

Security also has to be taken into account due to the fact we are dealing with highly sensitive data. Adyens systems adhere to the PCI Data Security Standard (DSS) which is a set of comprehensive requirements for enhancing payment account data security. The logging framework has to adhere to this standard too. Furthermore the systems have to deal with situations where the central system is not available. All these considerations influence the message protocol used and the frequency of the messages.

3.5 Manage statistical information on the central system

The data gathered can be stored on the central system in several ways.

1. In a database system.
2. In files on the filesystem.
3. Create (visual) output immediately from data in memory.

Once the logging information is available on a central system in a structured way, aggregations can be made and analysis can be performed. Queries to the central system and data-sets for graphical representation have to be defined. Within the logging information a distinction has to be made between information provided by log4j and the information in the message part that is constructed by the application as outlined in Section 2.4.

Because of the high volume of logging information special attention has to be paid to optimising the queries so that realtime access is possible. The high volumes also necessitates designing a archiving strategy to manage all the data coming in and prevent the hard disk from flooding. A third aspect to check is whether all the data is available. If for example the information of one system is missing, the statistics can be off and can give the wrong impression.

A possible optimisation step is to aggregate the information based on common denominator for example session id.

3.6 Visualise statistical information

Once the shopper behaviour and transaction information is neatly organised, statistical analyses can be performed on the information. Subsequently a way to visualise these statistics is needed. These need to be visualised in a clear and consistent way such that the interesting conversion trends within the payment process can easily be recognised. Even more important is to identify the bottlenecks in the HPP user interface which cause shoppers to leave before the payment is finalised.

After the statistics have been analysed in some detail some of them can be provided to the merchants. For this an intuitive and aesthetically pleasing way of presentation is needed. The goal is to make the visualisations are self explanatory so the merchants can use them without training. Figure 3.4 shows the user interface of the first implementation cycle. It is still very basic and will be extended as discussed in Section 3.7, but it gives a good impression of what should be achieved.

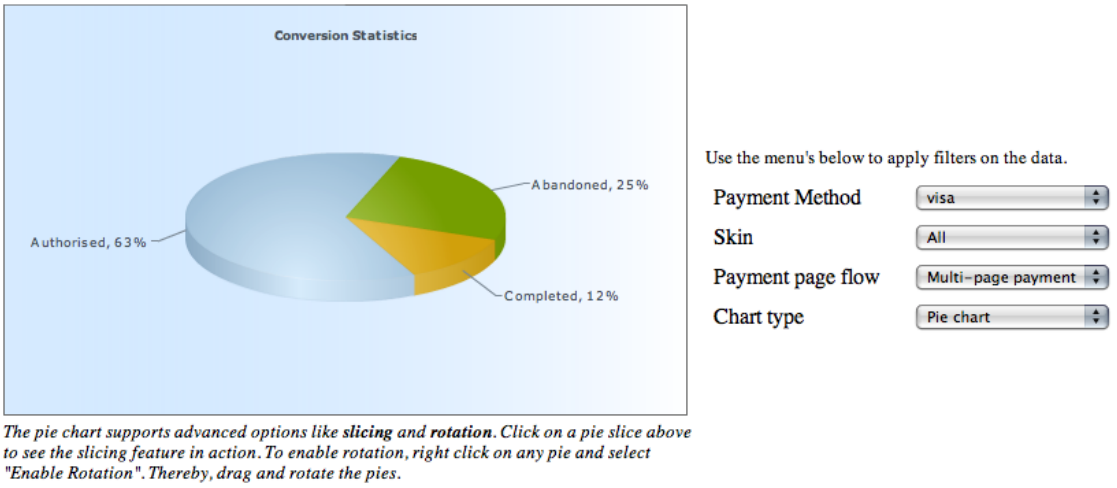


Figure 3.4: Web interface of the first implementation cycle

The payment result codes encountered in Figure 3.4 are listed below.

- **Authorised**, the transaction is authorised by the acquirer.
- **Completed**, the transaction is either rejected by the acquirer or pending final verdict.
- **Abandoned**, the transaction was abandoned prematurely.

3.7 Strategy

Because more insight is needed into the shopper behaviour in order to formulate proper hypotheses, an iterative approach will be used, with implementation cycles kept as short as possible. In this way

the things that were learned at the end of each cycle can be taken into account when updating our initial implementation and research. This is why a start has been made with parsing the conversion logs in their current format, doing statistical analysis in memory and providing basic visual representation on our development station before starting to make the log data machine readable.

Furthermore testing will initially be done with information supplied by the sales team before moving on to external testing with the merchants. During the design and development the system will be kept as generic as possible so that parts of the system can be reused.

Chapter 4

Log the required data

In this chapter first the design will be discussed with focus on additional information to be recorded by modelling the payment flow. The second part of this chapter concerns the implementation of this design and touches on the frameworks used and the validation of payment sessions through state machines.

4.1 Iterative development approach

As is noted in Section 3.7 an iterative approach is used in developing the conversion analysis framework. This approach allowed for early implementation of a basic parsing framework and simple user interface (see Section 3.6), according to the five step process as discussed in Chapter 3. In actuality only step {4} and {5} had to be performed in order to create a working system. This is because a good deal of conversion logging was already done on the HPP's as described in Section 2.4, making it possible to skip step {1} and {2}. Step {3} was circumvented by manually retrieving log files from the HPP's. This early implementation proved to be a good platform for analysing the logged data and understanding how the payment process was logged.

4.2 Design

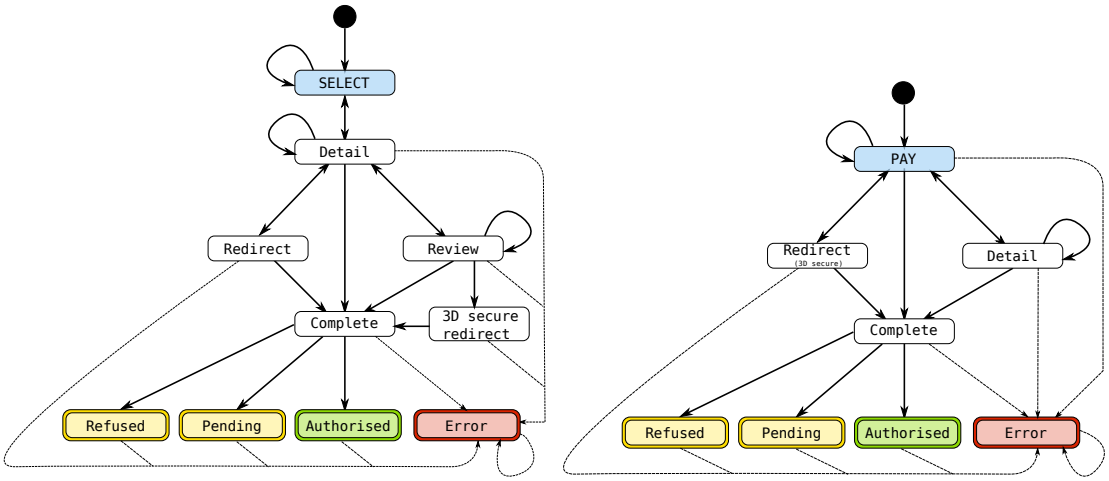
With insight gained from the first implementation, the payment flow can be modelled more accurately. In Section 4.2.1 the result of this modelling can be found. The next Section 4.2.2 discusses the additional data needed for step {2} of the next implementation cycles.

4.2.1 Model the payment flow

The logs as described in Section 2.4 contain one event per line. A payment session typically consists of multiple events. To gain greater insight in the flow of events within the payment process, flow diagrams were constructed to model the payment sessions.

As described in Section 2.3 there is a great difference in the flow of a payment session between the Multi Page Payment and the One Page Payment. But there are also differences between the flows of the various payment methods (e.g. not all payment methods use redirection). These two factors make the flow of a payment session fairly complex. For this reason two flow diagrams have been created, one for OPP and one for MPP sessions (as depicted in Figure 4.1).

A regular payment that is completed will pass the 'Complete' node and will end in one of the following three end nodes: 'Refused', 'Pending' or 'Authorised'. This partial flow of events holds for both the MPP and for the OPP. As is clear by looking at Figure 4.1 the 'Error' node can be reached by all other nodes in the diagram.



(a) Flow diagram modelling MPP sessions

(b) Flow diagram modelling OPP sessions

Figure 4.1: Flow diagrams modelling the payment sessions

4.2.2 Additional information to be logged

Besides the information that is already logged some additional information can be logged. Adding this extra information serves two different purposes. The first goal is to make the parsing of the logs as described in Section 3.3 easier and more robust. The second goal is to enrich the information as discussed in Section 3.1 to be able to create more valuable statistics.

- **PREVIOUS STAGE** Because of the statelessness of the HPP a transaction can be handled by different systems. The inclusion of the previous stage of the transaction in the conversion logs, in combination with the time stamp, makes it possible to reconstruct the exact path taken in the transaction.
- **IP ADDRESS** The IP address of the shopper can be matched against the GeoLite Country [24] database. The GeoLite Country database is a freely available database where an IP address can be used to retrieve the country where the IP address is based. Information about the shopper can be further enriched by performing a reverse DNS look-up of the IP address. The hostname that is retrieved by the reverse DNS look-up can also indicate the country of the shopper trough the top level domain of the ISP or company. Besides the country the name of the ISP or company is contained in the hostname.
- **USER AGENT** The user agent identifying the browser used by the shopper can give more information about the shopper. The browser and operating system and their respective versions are part of the user agent.
- **ERROR MESSAGES** Of considerable importance are the possible error messages that are displayed to the shoppers during their payment process. Decisions of the shoppers can be examined in more detail when we have a better view of what is going on (or wrong) in the payment process. Currently only a few errors are logged in the conversion log. Validation errors like wrong user input (e.g. invalid credit card number) are not logged at all, but are very valuable to abandonment statistics.

In the future the information present in the log files may change due to altered needs. In order to facilitate different versions of the log files a header can be included in the logs. Although this can alleviate some problems, the complete process of parsing, pattern recognition and visualisation needs to be adapted. Having a good testing framework in place can help find failures caused by log format changes in an early stage. Software testing and quality is discussed further in Chapter 9.

4.3 Implementation

Now that the data is defined and the process modelled, the various frameworks used to perform the logging within Adyen's platform are discussed. The last section covers how the payment sessions are validated through the use of state machines.

4.3.1 Java logging framework

Throughout Adyen's whole platform a Java logging framework is used to keep track of behaviour of the various applications. Within Adyen's platform the Apache Commons Logging API [10] is used. The Commons Logging package is an ultra-thin bridge between different logging implementations. When Commons Logging is used in a library this will not force the user to use a specific logging framework but allows the user of the library to make its own choice. Using Commons Logging allows Adyen to change to a different logging implementation without recompiling code; as discussed in Pro Apache Log4j [13].Another benefit of Commons Logging is that it is supported natively by Tomcat. In the current situation the log4j framework is used as a back-end for the Commons Logging API.

4.3.2 Validate the payment sessions

To perform valuable statistics the validity of the payment session as well as the 'route' (sequence of pages/actions) the shopper followed in this session needs to be determined. To determine this state machines modelled after the diagrams in Section 4.2.1 were created. This resulted in two distinct state machines for the MPP and OPP respectively.

A state machine operates on all events associated with a certain session and finds out things like payment result (authorised, pending, et cetera) and whether there were redirects or not.

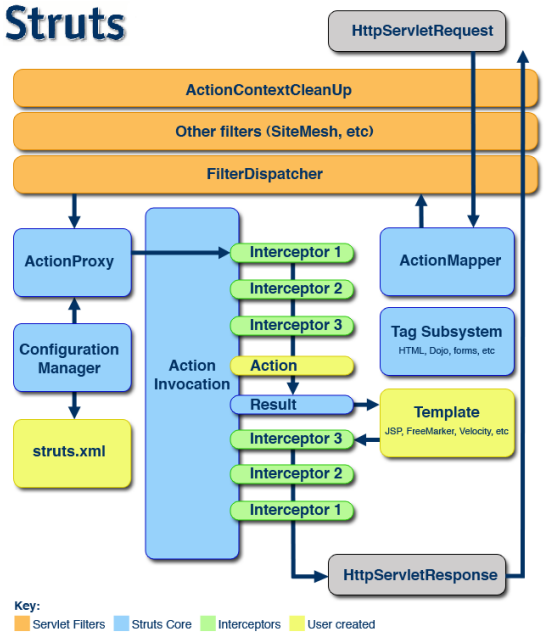


Figure 4.2: Struts 2 architecture Diagram [1]

4.3.3 Struts framework

Adyen uses the Struts 2 framework [1] to create the applications in its platform. The Struts framework is designed to help developers create web applications that utilise a Model View Controler (MVC) [21] architecture. Using the MVC model helps to separate concerns in a software application between database code, page design code, and control flow code. Within the Struts framework there are filters, called interceptors, which can modify the servlet request and response as illustrated in Figure 4.2. Besides the default interceptor stack you can add extra custom interceptors to the stack. It is therefore possible to create a logging interceptor that can log events from the request, the processing and the response stage. The logging interceptor would be placed on the location of 'Interceptor 3' in Figure 4.2. In contrast to the current in-code logging, by using this approach we can be sure that every stage in the payment session is logged. This is because every action must pass the logging interceptor.

Chapter 5

Make the logs machine readable

Log files need to be written into a machine readable format, because this makes it possible to have an application parse the data efficiently and provide us with the required functionality. This includes important features such as allowing to quickly access the log data and do the required operations like searching for a specific log event or recognising patterns. Because of the need to read the log data into an application its imperative that we pay attention to the format of the data logged and the method of processing. Even though we are mainly concerned with the conversion logs in this project, we will focus on a generic log format which meets all the requirements of the Adyen platform on the (application) log data as described in Section 3.7.

5.1 Analysis of log formats

As stated in Section 3.3 there are several formats in which the log information can be stored. In this chapter we will discuss some of the pros and cons of the various formats and contrast them to one another. The different formats can be characterised up to a certain extent by some high level properties. Table 5.1 contrast the various logging formats with respect to these high level properties.

- **Human readable** The raw logs can be read and understood by developers (humans) using simple tools as text editors and database viewers. One can easily see which data belongs to which attribute and processing large sets of data is possible. This is important because when the system fails, log data needs to be available immediately.
- **Searching** The logs can easily be searched for specific occurrences of values. This makes finding problems in the log data easier.
- **Processing** The cost of processing in terms of disk space and processing power required by the parsing, storing and loading of the format. A small footprint enables the HPP servers to process more transactions.
- **Distribution** The log data can be centralised without much effort and network load. Distribution needs to be kept simple because we don't want to lose log data. In complex systems many things can go wrong, especially as we run many systems in parallel.
- **Log4j support** The log4j framework supports the format natively, which makes it much easier to output the format.

In the following sections we will discuss each log format and give their main pros and cons. For the file formats one conversion log statement is listed as example. After these sections a performance overview of the log formats is given before concluding which format is best suitable and will be used in the Adyen platform.

Table 5.1: High level properties of logging formats

	Human readable	Searching	Processing	Distribution	Log4j support
CSV	+	+/-	+	-	+
XML	+/-	+/-	-	- -	++
Encoded Java property file	- -	+/-	+	-	+/-
Serialised Java objects	- -	- -	+/-	+/-	+
Directly to a database	++	++	+	++	-
Directly to a web service	- -	+	+	+/-	+/-

5.1.1 CSV

A Comma Seperated Values (CSV) file is used for the digital storage of data structured in a table form. Each line in the CSV file corresponds to a row in the table. Within a line, fields are separated by commas, each field belonging to one table column [38]. The main pros and cons are listed below:

+ Human readable; especially in big log files the fact that there is only one line per log event pays off.
+ Very small processing time and storage size.

- CSV hasn't got the field names along with the field values.
- Need to use escape characters, which can be non trivial for log4j part.
- Need for custom scripts to gather files on a central system. (These scripts also need to be monitored centrally).

Listing 5.1: Sample log event in CSV format [396 Bytes]

```
1 "2009-06-04T09:11:21.52Z","INFO","com.adyen.actions.hpp.conversion.
    ConversionLogger","TP-Processor31","TUDelftWebShop","arq2cl7r","
    PYsBL8MKtjki3xTGF9JD3bxttg0=","OrderID=7","2009-06-04T09:11:21.70Z
    ","7700","EUR","review","complete","CompleteCardPayment","visa","
    AUTHORISED","1412326705785803","62.194.174.193","Mozilla/4.0 (compatible
    ; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727)"
```

5.1.2 XML

The eXtensible Markup Language (XML) is a general-purpose specification for creating custom markup languages. It allows the user to define their own mark-up elements and was designed to carry data between applications [36]. XML however introduces a significant processing time and storage size overhead in the case of application logging, since it is a large regular data set. This is clearly described by R. Lawrance in his paper [9]:

> *Although most applications are not affected by the additional overhead in XML documents, certain applications involving large, regular, data sets will incur a significant overhead and performance penalty by encoding in XML.*

The main pros and cons of XML as a log format are listed below:

+ Human readable.
+ Well known standard.
+ Easily extendable.

- Log files result in large file sizes.
- Requires more memory / processing power to create / read.
- Difficult to get an overview of several lines. (Requires a lot of room on the screen.)
- Need for custom scripts to gather files on a central system. (These scripts also need to be monitored centrally.)

Listing 5.2: Sample log event in XML format [881 Bytes]

```
<log4j:event timestamp="2009-06-04T09:11:21.52Z" level="INFO" logger="com.
    adyen.actions.hpp.conversion.ConversionLogger" thread="TP-Processor31">
  <log4j:message>
    <merchant>TUDelftWebShop</merchant>
    <skincode>arq2cl7r</skincode>
    <sessionId>PYsBL8MKtjki3xTGF9JD3bxttg0=</sessionId>
    <merchantReference>OrderID=7</merchantReference>
    <timeWindow>2009-06-04T09:11:21.70Z</timeWindow>
    <amount>7700</amount>
    <currency>EUR</currency>
    <previousStage>review</previousStage>
    <stage>complete</stage>
    <class>CompleteCardPayment</class>
    <paymentMethod>visa</paymentMethod>
    <resultcode>AUTHORISED</resultcode>
    <pspReference>1412326705785803</pspReference>
    <ipAddress>62.194.174.193</ipAddress>
    <userAgent>Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
        2.0.50727)</userAgent>
  </log4j:message>
</log4j:event>
```

5.1.3 Java property file

Java property files are mainly used in Java related technologies to store the configurable parameters of an application. They can also be used for storing strings for localisation. One property file can only assign one field (key)-value pair, this would mean a new file for every log event. This approach is obviously not feasible for large logs. This problem can be solved by encoding the entire property file on one line. This has as added benefit that data encoded in this way is easily streamable. An overview of pros and cons is given next:

+ Extra encoding layer can provide more security.
+ Easily parsable.

- Only key-value pairs.
- The required encoding results in a non human readable file.
- Impossible to search since data is encoded.

Listing 5.3: Sample log event in Java property format [545 Bytes]

```
timestamp=2009-06-04T09:11:21.52Z
level=INFO
logger=com.adyen.actions.hpp.conversion.ConversionLogger
thread=TP-Processor31
merchant=TUDelftWebShop
skincode=arq2cl7r
sessionId=PYsBL8MKtjki3xTGF9JD3bxttg0=
merchantReference=OrderID=7
timeWindow=2009-06-04T09:11:21.70Z
amount=7700
currency=EUR
previousStage=review
stage=complete
class=CompleteCardPayment
paymentMethod=visa
resultcode=AUTHORISED
pspReference=1412326705785803
ipAddress=62.194.174.193
userAgent=Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
    2.0.50727)
```

Listing 5.4: Sample log event in Java property format encoded in base64 [740 Bytes]

```
dGltZXN0YW1wPTE5MTItMDYtMjNUMzQ6NDE6NTAuNTRaCmxldmVsPUlORk8KbG9n
Z2VyPWNvbS5hZHllbi5hY3Rpb25zLmhwcC5jb252ZXJzaW9uLkNvbnZlcnNpb25M
b2dnZXIKdGhyZWFkPVRQLVByb2Nlc3NvcjMxCm1lcmNoYW50PU1hcmsmT25uby0+
Qid2byEKc2tpbmNvZGU9YXJxMmNsN3IKc2Vzc2lvbklkPVBZc0JMOE1LdGpraTN4
VEdGOUpEM2J4dHRnMD0KbWVyY2hhbnRSZWZlcmVuY2U9T3JkZXJJRD03CnRpbWVX
aW5kb3c9MjAwOS0wNi0wNFQwOToxMToyMS43MFoKYW1vdW50PTc3MDAKY3VycmVu
Y3k9RVVSCnByZXZpb3VzU3RhZ2U9cmV2aWV3CnN0YWdlPWNvbXBsZXRlCmNsYXNz
PUNvbXBsZXRlQ2FyZFBheW1lbnQKcGF5bWVudE1ldGhvZD12aXNhCnJlc3VsdGNv
ZGU9QVVUSE9SSVNFRApwc3BSZWZlcmVuY2U9V2hlcmVJc0FsYW4/CmlwQWRkcmVz
cz02Mi4xOTQuMTc0LjE5Mwp1c2VyQWdlbnQ9TW96aWxsYS80LjAgKGNvbXBhdGli
bGU7IE1TSUUgNy4wOyBXaW5kb3dzIE5UIDUuMTsgLk5FVCBDTFIgMi4wLjUwNzI3
KQ==
```

5.1.4 Serialised Java objects

Serialisation can be used to persist Java objects. When serialisation is used, objects are converted into a sequence of bits so that they can be stored on a storage medium. The opposite procedure, de-serialisation, can be used the invert the process and create object from a sequence of bits. The pros and cons of this log format can be found below:

+ Reading and writing native to Java.
+ Straight forward log4j support.

- Requires that the entire object to be read for a single property.
- Needs to be backwards compatible if logged object changes.
- Non human readable.
- Impossible to search since data is encoded.

Listing 5.5: Sample log event in Java serialised object format [935 Bytes]

```
1  sr*com.adyen.conversionratio.logfiles.LogLine‘ Y> yO    L levelt Ljava/
     lang/String;L loggerq ~ L messaget /Lcom/adyen/conversionratio/
     logfiles/LogMessage;L threadq ~ L timestampq~ xptINFOt 1com.adyen
     .actions.hpp.conversion.ConversionLoggersr –com.adyen.conversionratio.
     logfiles.LogMessageJ N4Pc L amountq ~ L currencyq ~ L
     handling_classq ~ Lip_addressq~Lmerchantq~Lmerchant_referenceq ~ L
     payment_methodq ~ L previous_stageq ~ Lpsp_referenceq ~ Lresultcodeq
      ~ Lsession_idq ~ L skincodeq ~ L stageq ~ L time_windowq ~
     Luser_agentq~xpt 7700t EURt CompleteCardPaymentt 62.194.174.193t
     TUDelftWebShopt OrderID=7t visat reviewt 1412326705785803
     tAUTHORISEDtPYsBL8MKtjki3xTGF9JD3bxttg0=t arq2cl7rt completet2009
     –06–04T09:11:21.70Zt FMozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;
      .NET CLR 2.0.50727)t TP–Processor31t 2009–06–04T09:11:21.52Z
```

5.1.5 Directly to a web service

The Adyen platform conforms to a Service Oriented Architecture. The individual web services use SOAP [34] for communication. The messages are not managed, but are directly linked to (transaction) processing. A high transaction load therefore results in high load on the central web service. Writing directly to a central system means that we can more easily combine the data on the central location, but keeping the connection open will negatively affect the network load and processing time. The main pros and cons are listed next:

+ No extra step for getting it on a central system.
+ No delay in data on the central system.
+ Excellent for pattern recognition at the central system.
+ Perfectly searchable (although dependant on a central system implementation as discussed in Chapter 3.4).

- Dependent on a external service which can cause a delay.
- High transaction load result in high logging load on the central logging service.

5.1.6 Directly to a database on the local system

Looking at the characteristics at hand, logging to a database might seem logical because we are dealing with a large regular data set which needs to be easily searchable and should be able to recognise patterns. A distinction should be made between the log4j part of the log event and the application specific message part. Both parts should be stored in different tables. We believe logging to files is common due to major concerns that touch the primary purpose of logging. Accessing a database is much less reliable than accessing a file and generally requires more resources.

When looking at the distribution of the logs, there are three benefits of having a database push the data to a central system. First this functionality is natively supported by the database engine. Secondly this can be scheduled on times when there is a low system load and thirdly this process is already used in other parts of the Adyen platform for some of the application databases. Along with the clear benefits for the distribution the following pros and cons can be distinguished:

+ Very easily searchable.
+ Good for automatic pattern recognition.
+ Optimised process for copying data (one minute interval) already available in database engine.
+ Database replication & archiving already in use in the Adyen platform.
+ Log4j data separated from the actual log data (in different tables).

- Also need for logging to a file, since the database might be down, in which case logging is especially required.
- Expected to be slower compared to logging to a file.
- Might delay handling the request.

5.2 Performance overview of parsing log formats

Performance test were executed so we did not have to rely solely on external sources and get more insight into parsing log files. Although the performance of the different formats may also differ with respect to storing the log file, the tests only measured the performance of parsing the log files. This approach was taken because the performance of parsing the log files on the central system will have the biggest impact on the performance of the overall system.

In this performance test we only included the different file based formats because the other two formats have an infrastructure associated with them and make an unfair comparison. Performance characteristics of the database and web service are interesting but fall outside the scope of this project. Also the reliability of these two formats is a good reason not to use them as primary log format.

Table 5.2: Performance of logging formats (in milliseconds)

	Size (in Bytes)	Run 1	Run 2	Run 3	Average	Normalised avg.
CSV	396	1520	1527	1636	1561	1,55
XML	881	29205	29037	29457	29233	29,12
Java property file	545	964	969	1080	1004	1
Serialised Java objects	935	9819	9760	9860	9813	9,77

Table 5.2 lists the execution times of the various formats in milliseconds. For each of the specific formats as described in the previous sections, a sample file containing one event was created. The created files were read back and parsed 10,000 times for each format. The last column containing the normalised average is especially interesting because it illustrates the large difference in performance between the different formats.

It should be noted that the Java property file was spread over several lines and not encoded on a single line. Getting the property file encoded on a single line will add some overhead to the performance listed in Table 5.2.

5.3 Implementation format

Based on this chapter it is clear which log formats are most suitable. File based formats will be discussed first. After that we will contrast the two non file based formats. Finally we will pick the

most suitable format keeping in mind the generic goal of creating log files.

Experience of Adyen learns that human readable log files are essential, but are not very efficient in use. The size of the log files also has to be kept to minimum to prevent the hard disk from flooding.

The purpose and use of the log files combined with the discussion in this chapter on log file formats clearly eliminates Serialised Java objects due to performance, size and readability. As Java property files only have a small performance benefit compared to CSV and does not provide extra benefits in comparison to the other file formats, property files are dropped from further consideration. Encoded property files only makes performance worse and are not acceptable due to lack of readability.

Looking at the two remaining formats, CSV and XML, CSV files should clearly be preferred as it outperforms XML by a factor of 18.7 and has a footprint of 0.45 times the size of XML. Furthermore the readability of CSV is far better when concerned with reading large amounts of data on a screen.

Storing the logs on a web service or in a local database differ greatly in a single aspect. Although both systems rely on external services, the database is hosted on the local system whereas the web service is hosted on a remote machine.

The web service configuration greatly influences how the transaction load is multiplied through the complete platform by communicating multiple log events per transaction to a central system. Another factor is the processing latency which will be greater when communicating to a central system directly. Both factors clearly indicate that the database format should be preferred instead of a web service.

When choosing between CSV and database logging we need to consider that CSV log files introduces minimal dependency. Because log information needs to be available in case of a database failure, which still occurs, file logging is vital. The database format however has very clear advantages in readability, searchability and pattern recognition. We therefore conclude a combination of the CSV and database format is the optimal situation due to their complementing characteristics.

The log4j framework supports this configuration by outputting to multiple destinations / formats concurrently. Alternatively the CSV log files can be read directly into a local database, this functionality is supported by the database. In Chapter 6 we will discuss the available centralisation strategies in depth.

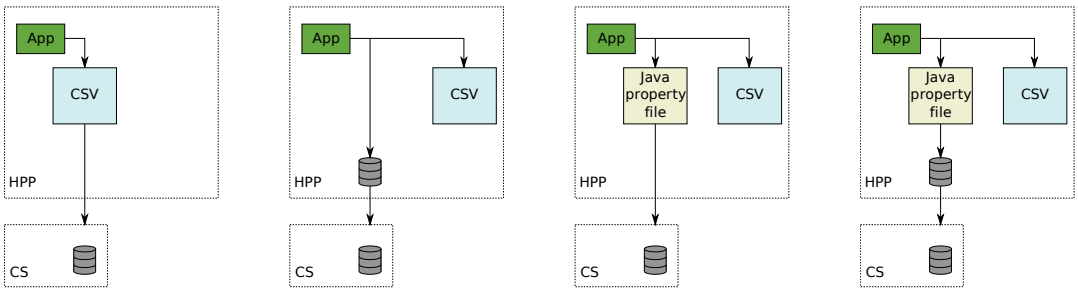
Chapter 6

Centralise the log information

There are a few reasons why a central database is essential for the (conversion) logging, not the least of which is the fact that the HPP run on redundant systems (as explained in Section 2.1). This means that the log events of one session can be spread across multiple machines. To reconstruct the payment sessions from the conversion log data we need to have all events of the session present on a central location. The sessions need to be reconstructed so accurate statistics can be generated. Which brings about the next reason for having a central database, namely the HPP server load. These systems need to be freed as much as possible from tasks other than processing actions on the HPP.

6.1 Centralisation strategies

As is explained in Chapter 5, the conversion logs have to be transferred to a central system in order to analyse the data and produce meaningful statistics. In the following sections we will look at different centralisation strategies and their characteristics. Some possible centralisation strategies are depicted in Figure 6.1.



(a) Periodic transfer of CSV logfiles

(b) Logging directly to a database and transfer with database replication, keeping CSV as backup

(c) Logging to Java property files with periodic transfer, keeping CSV as backup

(d) Logging to Java property files, reading into local database and transfer with database replication, keeping CSV as backup

Figure 6.1: Four centralisation strategies

The strategies illustrated in Figure 6.1 can be divided into two categories: periodic file transfer and database replication. Periodic file transfer (Figure 6.1a and 6.1c) is described in Section 6.1.1. Section 6.1.2 explains database replication (Figure 6.1b and 6.1d).

Figure 6.1 only shows the use of two file formats: CSV and Java property files. Off course other file formats are possible, but these formats are the strongest file based formats as discussed

in Section 5.3. In Figures 6.1b through 6.1d the CSV format is used as a secondary format for the purpose of readability in case of an error, as explained in the previous chapter.

6.1.1 Periodic transfer of rolling log files

When file transfer is used as a centralisation strategy (illustrated by Figure 6.1a and 6.1c) it is assumed that the conversion logs are logged as plain text files on the HPP. To get the files on a central system, scripts are running periodically on the central system in order to transfer the files from the HPP to the central system.

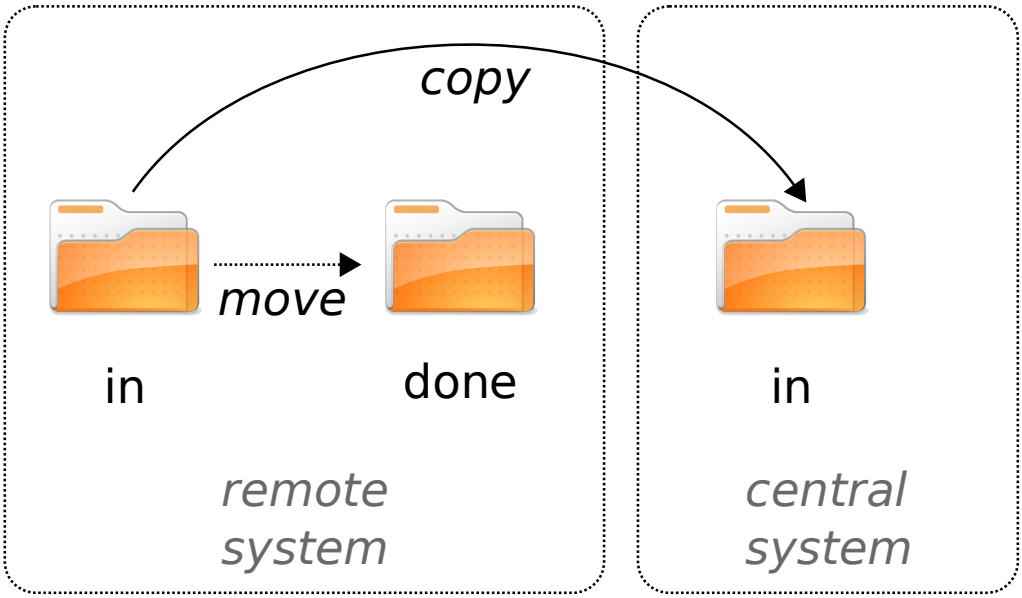


Figure 6.2: File transfer job diagram

The files are put in a special designated 'in' folder by the logging framework so the central system can tell which files have been added recently and still have to be transferred. The job then copies the files to a new 'in' folder on the central system and moves the original files to a 'done' folder on the remote system as depicted in Diagram 6.2

6.1.2 Database replication

If the conversion logs are stored in a local database on the HPP, getting the information on a central system can be accomplished through the use of database replication (illustrated by Figure 6.1b and 6.1d). Replication is the process of sharing data to ensure consistency between multiple resources. A big advantage of this strategy is that replication is built into the database system. This means that provisions are made to cope with for example unavailability of the central database.

Directly putting all the information from the different systems into the central database is called multi-master replication. Multi-master replication can be done in synchronous or asynchronous way. Using the synchronous approach each database transaction will cause inter system communication and therefore a heavy performance penalty will be paid.

The asynchronous approach can be used to minimise the performance penalty but has some difficulties associated with it [19]. These difficulties arise when the information is altered at different locations and result in a conflicting situation. An example of this is when a row is updated in two or more different databases before the changes are propagated to the other databases. The

system will then not know which updated values to use for that row. For the specific case of the conversion project this is not that big an issue because only new records are inserted and no records are updated.

These difficulties can be circumvented by having a one-to-one copy of the remote databases on the central system. The copies on the central system will be kept up-to-date through the use of database replication. A periodically run job is then still needed to combine all the information from the different sources. This job however can perform additional task like aggregation and calculations for the specific log files at hand, in this case the conversion logs as explained in Section 7.2.

6.2 Design considerations of strategies

In the next sections we will compare the different strategies from the previous section with respect to properties like load balance and security.

6.2.1 Load balance

As explained in Section 5.3 when logging to the local database, the logs are also saved as text files to cope with for example the unavailability of the database. Logging to a local database will therefore naturally add more system load because of the resources needed by the database service. The system load is even further increased by the replication process run by the database server.

When copying the text log files the system load will increase for the period of time needed to transfer the files. This increase of system load can be decreased by choosing a compact file format like CSV. A further cut down of system load can be accomplished by transferring the files more often so that the files transferred are smaller in size. This approach has the added benefit that the system load will be spread more evenly. The timing of the transfer can be chosen so that the system load is minimal, for example at 0.00 p.m.

6.2.2 Network capacity

Adyen's platform is hosted at two different physical locations. Within each hosting location Adyen's systems are connected through 100 Mbit network connections. To connect the two hosting locations a Virtual Private Network (VPN) connection is used.

Although the VPN connection between the hosting locations can be the limiting factor with respect to network capacity, with the current high-speed broadband connections this will not be a problem in practice.

6.2.3 Security

When the data is centralised this should be done in a secure fashion because of the highly sensitive nature of the data. Within one physical hosting location the connections are the property of Adyen and are administered as such. The use of VPN to interconnect the two locations as stated earlier also ensures that this communication is secure.

When file transfer is used as a centralisation strategy the Secure Shell (SSH) protocol is used to connect to the remote system. Database replication uses the Secure Sockets Layer (SSL) protocol to connect to the central database. Both SSH and SSL are industry standards to secure network communication between two machines.

6.3 Implemented strategy

With the discussion of the different strategies in the previous sections a centralisation strategy has to be selected. The optimal solution would be to use database replication. This strategy has the major benefit that is built into the database system and has mechanisms to cope with the unavailability of the central system. A second good characteristic is that the system load is spread more evenly across time because the replication strategy uses small increments to propagate the changes made to the data.

Although database replication is the preferred centralisation strategy, the periodic transfer of files is used in the project. This is because changing the log format has a big impact on the platform as a whole. Therefore the files are still logged in their current format. Listing 6.1 shows a file transfer script that was adapted for the conversion logs.

Listing 6.1: File transfer script

```
#!/bin/bash

DEBUG=0

# Remote systems where to retrieve the logs from
REMOTE="hpp_system1 hpp_system2 hpp_system3"

# File locations on the remote system
NEWFILELOCATION=/usr/local/data/conversion/in
DONEFILELOCATION=/usr/local/data/conversion/done

# Location to transfer the files to on the cental (local) system
LOCALFILELOCATION=/usr/local/data/conversion/in

# Flag to indicate that new files were transfered
HASNEWFILES=0

# For all remote machines
for sshhost in $REMOTE
do
  [ "$DEBUG" = "1" ] && echo "Begin with $sshhost"
  # Connect to remote machine
  NEWFILES="$(ssh $sshhost ls -1 $NEWFILELOCATION)"
  [ "$DEBUG" = "1" ] && echo "Received [$NEWFILES]"
  if [ -n "$NEWFILES" ]; then
    [ "$DEBUG" = "1" ] && echo "NEWFILES is not null $file"
    # For all new files
    for file in $NEWFILES
    do
      [ "$DEBUG" = "1" ] && echo "Receiving $file"
      # Copy file to central system
      rsync -aq $sshhost:$NEWFILELOCATION/$file $LOCALFILELOCATION/$file
      if [ "$?" = "0" ]; then
        [ "$DEBUG" = "1" ] && echo "Moving $file"
        # Move file on remote system from 'in' to 'done'
        ssh $sshhost mv $NEWFILELOCATION/$file $DONEFILELOCATION
        HASNEWFILES=1
      else
        echo "rsync failed"
      fi
    done
  fi
```

```
43    [ "$DEBUG" = "1" ] && echo "Done with $sshhost"
44  done
45
46  if [ "$HASNEWFILES" = "1" ]; then
47    # Execute import job when new files were transfered
48    /usr/local/data/bin/ConversionLogImportJob.sh > /dev/null
49  fi
```

Chapter 7

Store the log data

An efficient and reliable storage format is extremely important for smooth operation of the conversion analysis framework. The design of the storage format needs to be generic enough to be able to hold any logged data, but also needs to be specific enough to allow complex querying of the data. Since the amount of logged data can be huge, the storage format needs to be able to handle large quantities of data and perform queries for statistics retrieval very quickly.

For the generic approach a common design is required, how this is realised is discussed in Section 7.1. The specific conversion log data storage design is discussed in-depth in Section 7.2, touching on aggregation, validation of the data and performance of the storage format. Finally the implementation for this project and the required technologies are discussed in Section 7.3 for two storage formats.

7.1 Data storage in a generic logging framework

As discussed in Section 3.2 Adyen is planning to create a generic logging framework. The design of the conversion logging framework has to be compatible with this generic logging framework. Therefore a split between the log4j-part and the message-part of the log data has been proposed, making it possible to add different types of log data in a single database table. Diagram 7.1 shows how this setup is realised in the case of a database setup, the information can however also be in-memory or on the file system. Also note that in this case the message part is a conversion event, but it could have been any type of logged data (e.g. application logs, error logs).

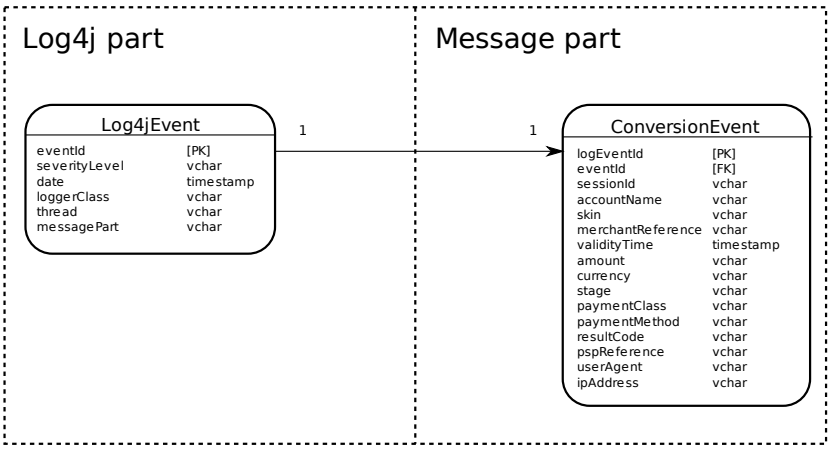


Figure 7.1: Design diagram for generic logging framework

7.2 Storage design for conversion log information

For the design of the storage format is the five step process that Reichle et al. proposed has been adjusted to specifically suit the needs of our project. In the following sections each of these steps is discussed and an extra step is performed for performance enhancement.

7.2.1 Data extension

When the conversion log data has been loaded into its designated storage format, it can be enriched with data from a knowledge base and external sources as illustrated by Figure 7.2. and discussed in Section 3.1. The order in which Reichle et al. performs the data processing steps is however not applicable to conversion data. We propose Data conversion & extension should be done as the final step and not in third step, since the conversion analysis framework only operates on aggregated and validated data. This is also the reason why the LogStatistic table shown in Diagram 7.3 can have a multiplicity of zero, which happens when the data is invalid (when invalid transitions in the state machine are attempted) and can not be aggregated. Also note that the input of the GeoIP database moves along with the Data conversion & extension step.

7.2.2 Aggregation

After data extension has been performed, the data can be aggregated in order to minimise unnecessary redundant information and increase performance when handling the data. Since the logged data is being processed anyway, this is the ideal moment to perform an aggregation step. Common properties that stay the same throughout all the events of a session are moved from the LogEvents to the LogSession. These are properties like account names, Skin codes, IP addresses

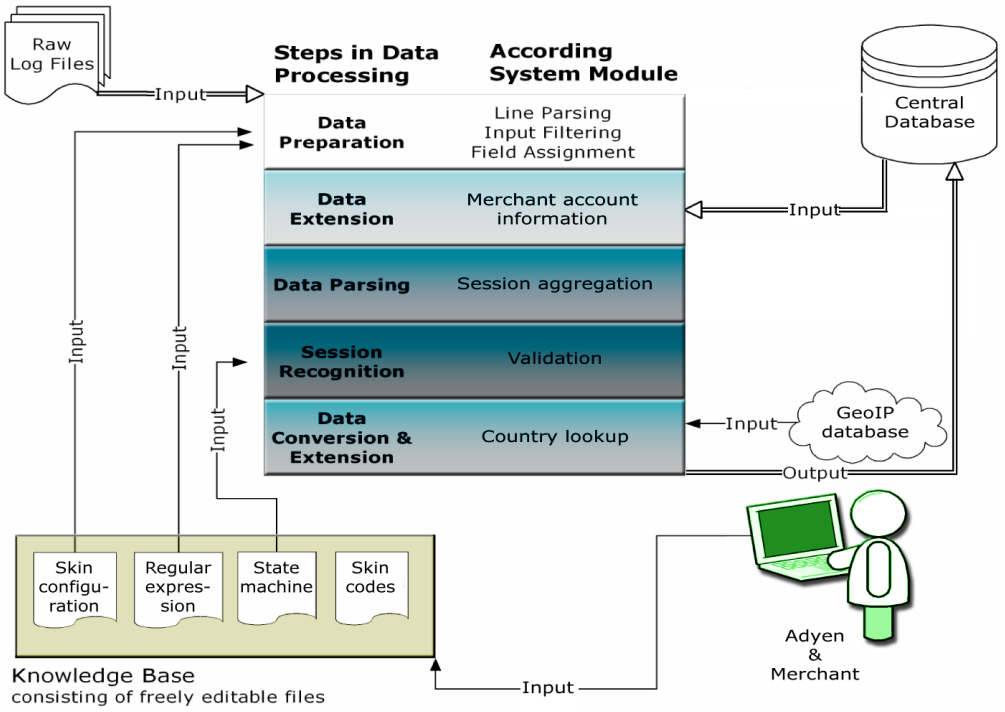


Figure 7.2: Overview of steps in data processing (as defined by Figure 3.3)

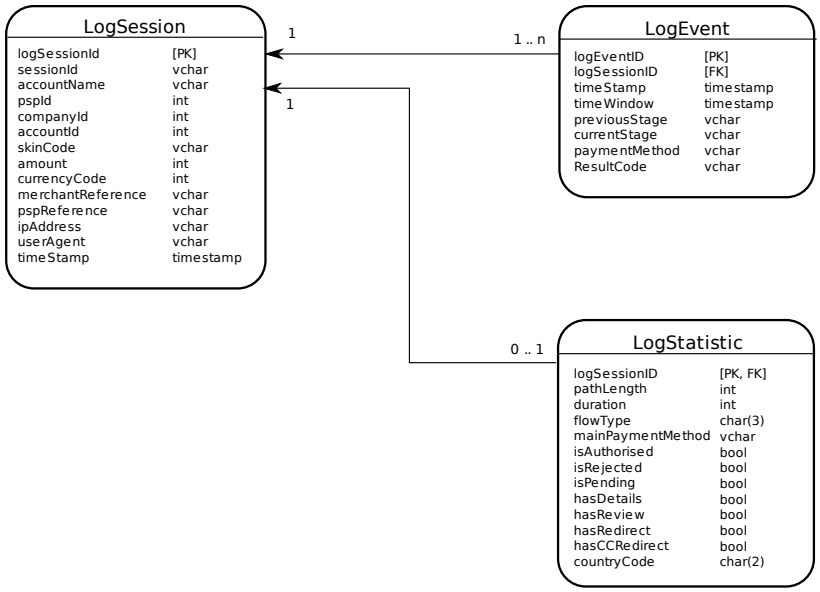


Figure 7.3: Database design diagram for conversion logs

and merchant accounts as can be seen in Figure 7.3. The smaller subset of properties that do change are stored in LogEvents.

7.2.3 Validation

After the properties have been redistributed a state machine is created by the StateMachineFactory by supplying the list of LogEvents from a session. The StateMachine object that is returned by the factory is then executed in an attempt to validate the payment session. After the state machine has evaluated the input data and ends in a valid state, the state machine's properties (as discussed in Section 4.3.2 are moved to the LogStatistic object.

7.2.4 Performance enhancement

In order to decide what information is necessary and what information may be sacrificed, the most important statistics were gathered. For all the individual statistics the compulsory data to get valid results was examined. The most important metrics are the number of authorised, completed and abandoned sessions. These numbers should be available for each merchant. For a merchant it is important to see how these numbers vary geographically and develop through time. This resulted in a database design as illustrated by Figure 7.4.

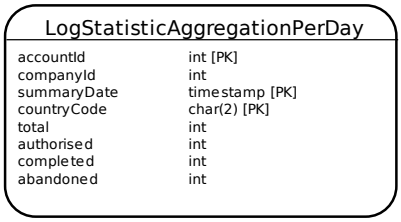


Figure 7.4: Database design diagram for performance enhancement

7.2.5 Archiving

The large amounts of data that are generated by the platform not only mandate performance enhancements, but also an archiving strategy. Without such a strategy the disks that hold the gathered data will flood. Because no information is lost when the data is imported into the central database, the original log files can be deleted as soon as they are imported successfully. The same holds when the logs are written to a database.

Although the data is aggregated, this data will eventually cause the disks to flood as well. The aggregated data should be available for a period of about six months. This ensures, when a new statistic and its corresponding additional aggregation are implemented, enough data is available to produce meaningful statistics right away.

7.3 Implementation

As stated in Section 3.7 the goal was to get the first results as early as possible. Therefore the results of the parsing process were stored in memory in the early stages of the project. As the project progressed, more information became available and storing the results in memory became unfeasible. The most natural way to store the results in a more structured and persistent way is to store them in a database.

The design of the framework facilitated the transition to the database very efficiently because of the LogStoreInterface (depicted in Diagram 7.5). In only a single line of code the swap between the different storage interfaces can be done. All statistics that functioned on the MemoryLogStore automatically function on the DatabaseLogStore. An extra method can however be added to the statistics classes to optimise the calculations with database query's specialised for this specific statistic.

The in-memory storage implementation will be discussed in Section 7.3.1, followed by the database implementation in Section 7.3.2.

7.3.1 Store in memory

In order to get the data from the logs to a datastructure in system memory, the raw CSV logfiles are parsed file by file. After a complete file has been read the file is processed one line at a time. First a regular expression is applied to the line to extract the application specific message part of the log statement. From this message part a RawLogEvent object is created where all the CSV fields are matched to properties of the RawLogEvent object.

The RawLogEvent is then added to a corresponding LogSessionWrapper object. If a corresponding wrapper object does not exist, it is created and added to the data structure. The LogSessionWrapper consists of a LogSession, a list of LogEvents and a LogStatistic object as illustrated in Diagram 7.5

After all lines have been processed the LogSessionWrappers are enriched with external data, aggregated and extended as discussed in Section 7.2.

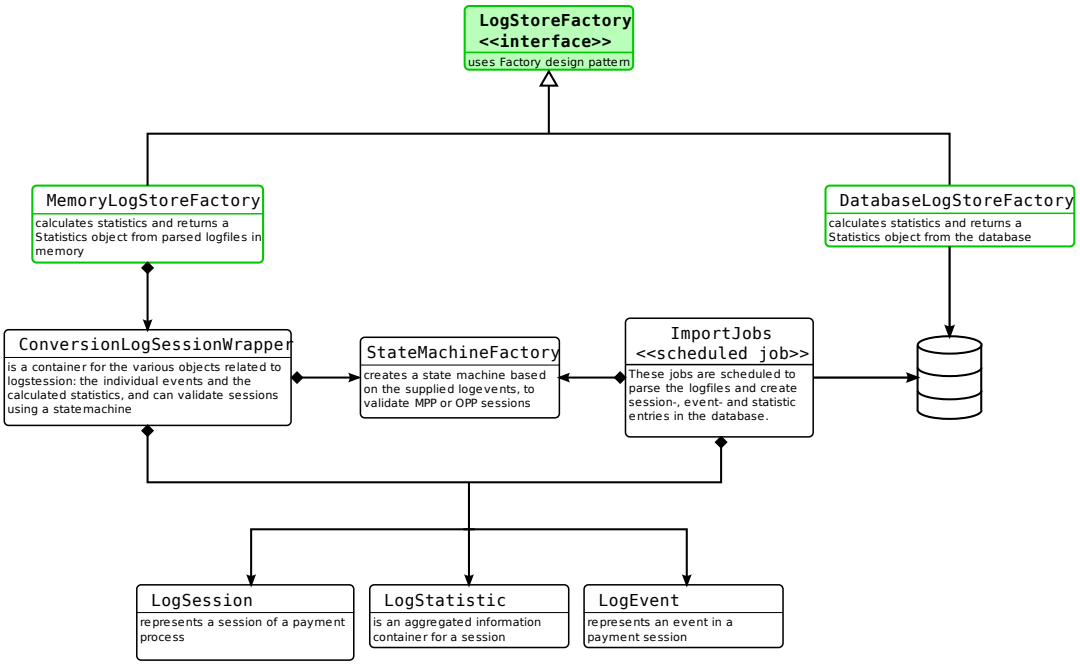


Figure 7.5: Class diagram of the conversion analysis framework back-end

7.3.2 Store in Database

The analyses of Chapter 5 clearly indicated that logging to a local database would be highly beneficial. However due to the big impact of this proposed change, the existing conversion log files are still used. The file transfer setup therefore is the strategy to be used.

Using this strategy the files will be copied over to the central system as is discussed in Section 6.1.1 and a job will be executed to import the log files. This import job looks in the central 'in' folder and picks up a new file by putting it in a 'processing' folder as illustrated in Figure 7.6. When the whole file has been processed the file will be moved to a 'done' folder. This process is repeated until the 'in' folder is empty. If an error occurs during this process the file will be placed in an 'error' directory so the it can be examined at a later stage.

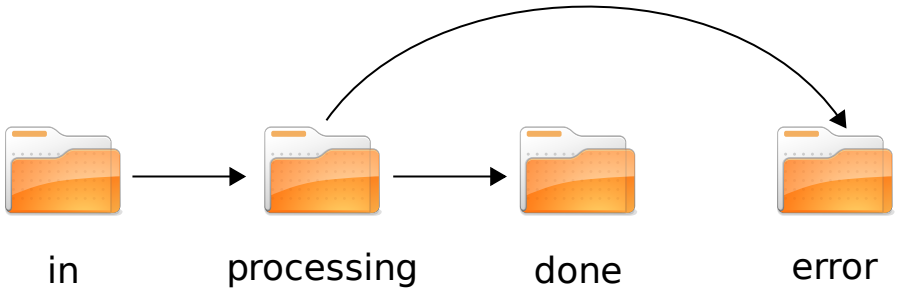


Figure 7.6: File import job diagram

The job then continues to parse the file and the results are put in the PostgreSQL [27] database. To facilitate the communication between Java job and database the iBATIS [2] framework is used.

In a XML configuration file Java Bean [31] classes are mapped to tables in the database. The configuration file also lists the various queries than can be performed on the database.

When the transferred log files are parsed the data is enriched with external data, aggregated and extended as discussed in Section 7.2. Unlike the aggregation as is done when using the in-memory storage method, the aggregation in the database is done by a single query (see Listing 7.1 which performs much better.

In this aggregation step efforts are made to reduce duplicate information as much as possible, however some redundant information is present for performance reasons. Although the company and psp associated with the account can be retrieved with the account Id, this information is retrieved and stored in the LogSession record at creation time. Storing this redundant information greatly reduces the number of joins needed in a query and therefore makes the queries to execute faster.

Listing 7.1: Query to create LogSessions and LogEvents from raw log events

```
1  -- Create LogSessions if necessary
2  insert into LogSession (
3          sessionId,
4          accountName,
5          pspId,
6          companyId,
7          accountId,
8          skinCode,
9          merchantReference,
10         timeStamp,
11         ipadress,
12         useragent,
13         dirty,
14         validationAttempts
15         )
16 select
17         sessionId,
18         merchantAccount,
19         pspId,
20         companyId,
21         accountId,
22         skinCode,
23         merchantReference,
24         min(timeStamp),
25         min(ipAddress),
26         min(userAgent),
27         true as dirty,
28         0 as validationAttempts
29 from RawLogEvent
30 group by
31         sessionId,
32         merchantAccount,
33         pspId,
34         companyId,
35         accountId,
36         skinCode,
37         merchantReference
38         ;
39
40 -- Update LogSession with
41 -- amount and pspReference
42 update LogSession as session
```

```
43  set
44          pspReference = raw.pspReference
45  from RawLogEvent raw
46  where raw.sessionId = session.sessionId
47          and raw.merchantAccount = session.accountName
48          and raw.pspId = session.pspId
49          and raw.companyId = session.companyId
50          and raw.accountId = session.accountId
51          and raw.skinCode = session.skinCode
52          and raw.merchantReference = session.merchantReference
53          and raw.pspReference notnull
54          ;
55
56  update LogSession as session
57  set
58          quantity = raw.quantity,
59          unitId = raw.unitId
60  from RawLogEvent raw
61  where raw.sessionId = session.sessionId
62          and raw.merchantAccount = session.accountName
63          and raw.pspId = session.pspId
64          and raw.companyId = session.companyId
65          and raw.accountId = session.accountId
66          and raw.skinCode = session.skinCode
67          and raw.merchantReference = session.merchantReference
68          and raw.quantity notnull
69          and raw.unitId notnull
70          ;
```

7.3.3 Performance enhancement

After the setup as described in the previous section was established on the local development machines, the application was installed on a staging server. As the amount of data accumulated on the staging server, the performance of the system degraded rapidly.

To keep the responsiveness of the system within bounds a new approach was needed. The biggest problem was the sheer amount of information. To reduce the size of the data set the available information should be aggregated while at the same time maintaining enough detail to enable meaningful statistics.

To aggregate all the information as outlined in Section 7.2.4 a query was developed to create a daily aggregation. Listing 7.2 shows how the data is summarised for the first of July.

Listing 7.2: Query to aggregate daily summary

```
1  begin;
2
3  insert into LogStatisticAggregationPerDay (
4    total,
5    authorised,
6    completed,
7    abandoned,
8    countryCode,
9    companyId,
10   accountId,
11   summaryDate
12   )
13 select
14   count(*) as total
```

```
15     ,coalesce(sum(case when lstat.isauthorised then 1 else 0 end), 0) as
           authorised
16     ,coalesce(sum(case when lstat.isauthorised or lstat.isrejected or lstat.
           ispending then 1 else 0 end), 0) as completed
17     ,coalesce(sum(case when not lstat.isauthorised and not lstat.isrejected
           and not lstat.ispending then 1 else 0 end), 0) as abandoned
18     ,lstat.countryCode as countryCode
19     ,ls.companyId as companyId
20     ,ls.accountId as accountId
21     ,date_trunc('day', ls.timeStamp) as summaryDate
22 from LogStatistic lstat
23 join  LogSession ls using (logSessionId)
24 where ls.dirty=false
25   and ls.timeStamp between '2009-07-01' and '2009-07-02'
26 group by summaryDate
27     ,ls.accountId
28     ,countryCode
29     ,ls.companyId
30 order by summaryDate;
31
32 commit;
```

7.3.4 Archiving

Although enough information is kept when the aggregated data is retained for a period of six months, the PostgreSQL database allows database partitioning.

When using partitioning one logical large table is split into smaller physical pieces. Partitioning can provide several benefits [28].

- *Query performance can be improved dramatically in certain situations, particularly when most of the heavily accessed rows of the table are in a single partition or a small number of partitions. The partitioning substitutes for leading columns of indexes, reducing index size and making it more likely that the heavily-used parts of the indexes fit in memory.*
- *When queries or updates access a large percentage of a single partition, performance can be improved by taking advantage of sequential scan of that partition instead of using an index and random access reads scattered across the whole table.*
- *Seldom-used data can be migrated to cheaper and slower storage media.*

Chapter 8

Visualise the data

A web interface is chosen as a visual interface for the conversion logging framework. This was an easy decision since the Adyen customer area (merchant backoffice) are also web-based. In order to provide the required statistics to the user the system employs a two-stage design. Firstly the page is setup using a customised template. Secondly the data is loaded asynchronously into this template from an XML webservice.

While designing the user interface the important question is: 'how would an Adyen employee/merchant like to see the user interface'. To get a insight into this question a few use cases are given in Section 8.1. The next Section (8.2) gives an in-depth view into the design of this two-stage system and Section 8.3 gives an overview of the implementation process and its challenges.

8.1 Usage of the conversion statistics

Use case: Show the statistics within a world region
Actor: Merchant
Possible reason: The merchant want to see in which European countries little transactions are done so he can start an advertisement campaign there to promote his product.
Steps:

Actor action	*System response*
1. Go to the merchant back-office page	2. Display page
3. Click on the conversion statistics link	4. Display the conversion statistics page
5. Click on Europe	6. Load the map of Europe and show the statistics corresponding to this region

Use case: Find out abandonment rate in The Netherlands
Actor: Merchant
Possible reason: The merchant wants to know if shoppers finalise their transactions in the Netherlands, maybe the shoppers don't understand the website, it might need a Dutch translation.
Precondition: Actor followed *Show the statistics within a world region* case
Steps:

Actor action	*System response*
1. Click on The Netherlands	2. Show the statistics corresponding to The Netherlands
3. Click on the enlarge button of the conversion statistics chart	4. Switch the map with the conversion statistics chart
	5. The conversion statistics chart is now the main chart and is presented in large format in the center of the page
6. Click on the chart options toolbar	7. Slide down the toolbar to show chart options
8. Select the check-boxes 'show values' and 'reveal percentages'	9. Update the main chart to show percentage values

Use case: Find out in what European countries the conversion ratio of credit cards is low
Actor: Adyen sales employee
Possible reason: The sales employee wants to know if localised payment methods should be introduced in countries where credit cards are not used very often.
Precondition: Actor followed *Show the statistics within a world region* case
Steps:

Actor action	*System response*
1. Click on the advanced filter options toolbar	2. Slide down the toolbar to show the filter controls
3. Select all credit card payment methods from the Payment methods filter box	
4. Click on the add button	5. The selected payment methods are moved to the right to indicate that they are active
	6. The statistics are updated to show only the values filtered on the active payment methods
7. Look at the map of Europe and notice the darker coloured countries, these are the important ones since the most transactions come from these countries	
8. Click on any of these countries	9. Show the statistics for the selected country
10. Look at the conversion statistics chart and look at the payment methods chart the actor can now decide on a solution for this problem (maybe introduce a new payment method in this country)	

8.2 Design

For the use cases given in 8.1, and others that might arise in the future, a template system has been developed as discussed in Section 8.2.1. Each template can display a variety of charts and diagrams, how exactly is explained in Section 8.2.2. The user interface (front-end) for the conversion logging framework is build upon the LogStoreFactory interface as is shown in Diagram 8.1. The LogStoreFactory is the interface connecting the front-end to the back-end (shown in Figure 7.5) of the system.

The actual webpage rendering is done using Struts2 MVC and the Velocity template engine. Key components in this diagram are StatsType and Filter and are discussed in Section 8.2.3 and 8.2.4 respectively. These components are used to communicate with the LogStoreFactory and quire the needed statistics. Section 8.2.5 explains how a webservice provides access to the

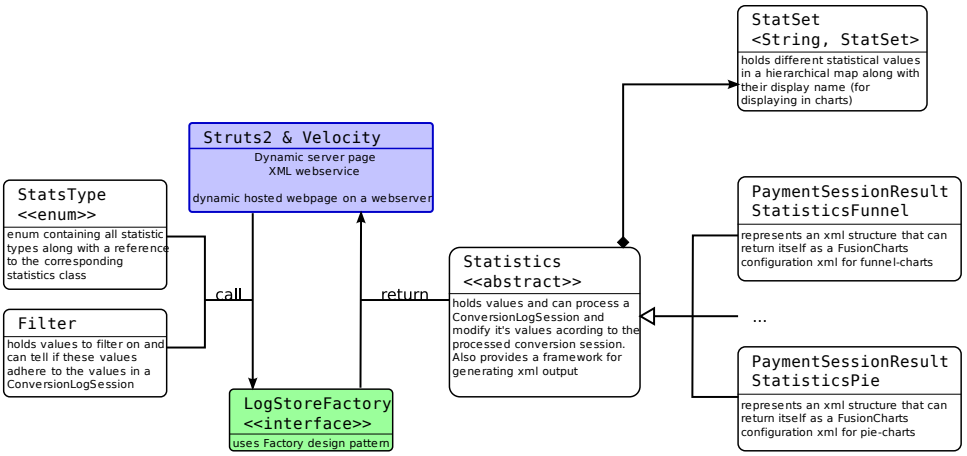


Figure 8.1: Class diagram of the conversion analysis framework front-end

LogStoreFactory.

8.2.1 Chart templates

There are infinitely many statistics which can benefit a specific user, therefore our system needs to be designed in a way that supports easy extension of the user interface. Furthermore any extension should maintain a consistent user experience. A few examples of typical useful functionality can be found in the use cases (8.1) above. Other functionality can be implemented in a way that is analogous to the these cases, but provides completely different information:

- Comparing the effects of different payment page Skins (OPP vs. MPP)
- Find regions where fraud rate is high
- Comparing the performance of different merchants
- Comparing the effectiveness of a payment method with other payment methods

The best way to keep the user interface consistent is to introduce a template. Within each template the contents can be varied, but once a template has been defined the statistics on display are always the same for this template, thereby simplifying the user experience. Finally a unified colour scheme has been devised to ensure a consistent look of the user interface. Each of the colours in this scheme has its own meaning, enabling the user to interpret the visual data in a single glance.

Statistics can still be moved to the large centre area, where the labels and values can be toggled on or off and the values can be set to percentages or absolute values, but the user can always anticipate what will happen when he performs a certain action. When the main statistic is switched with another, it's view settings are persisted in the small display so the user can customise the template view to his or her liking. Figure 8.3 shows two templates with one big (main) statistic visualisation in the center and 4 smaller ones next to it. Diagram 8.2 shows how the template (ChartTemplate) fits into the system design.

8.2.2 Chart types

Statistical information is worth nothing if it's not properly visualised. Improper visualised statistics can give the viewer a skewed impression of the reality. Therefore we need a system that is flexible enough to show any type of visualisation, for this reason we use the Fusion Charts [15] package. This package provides a extensive collection of charts, diagrams and maps, each of which is configurable by XML.

Each class extending the statistics abstract class must define at least one chart type from the ChartType enum (see Diagram 8.4). This way the system can always render the statistic on any template. Additionally the statistics class requires extending classes to implement a getFusion-Chart() method, which returns an object model as discussed in Section 8.2.5, thereby ensuring valid XML output. Any chart, diagram or map can be implemented in this way, as long as it's API (defined by Fusion Charts) is compatible with the object model. Of course the object model is build so it can be extended to suit any Fusion Chart API.

8.2.3 Statistic types

Each type of statistic requires a specific way of processing the available data. This processing is done in a specific calculator class extended from the Statistics class. Since a template should be able to hold any type of statistic, but should never have anything to do with the data processing, a generic way of retrieving the statistics must be defined.

The only information the template has is the type of statistic and optionally type of chart (see Diagram 8.2), a factory class is therefore used to create the right statistics from the raw data on input of a statistic type. All available statistic types are listed in the StatsType enum along with the calculator class that is able to calculate the statistic. The LogStoreFactory class takes a statistic type as input and parses an enum constant from this, which it uses to instantiate the right calculator class, this class is subsequently requested to do it's calculations on the data. The result the LogStoreFactory produces is a readily usable Statistics class with the ability to render XML configuration files for a specific chart, diagram or map in the template.

8.2.4 Filter

To create localised and specialised statistics the data needs to be filtered. There are a number of important fields that must be filtered on:

- Locale (for map navigation)
- Merchants (to create statistics specifically for a merchant or a group of merchant with a common business model)
- Payment Methods (i.e. to compare the effectiveness and locale)
- Skins (i.e. to do A/B testing of Skins)
- Payment page flow (OPP vs. MPP)

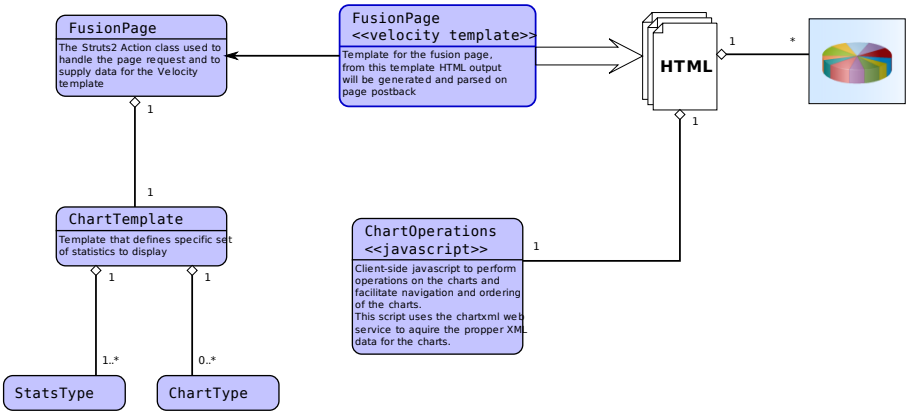
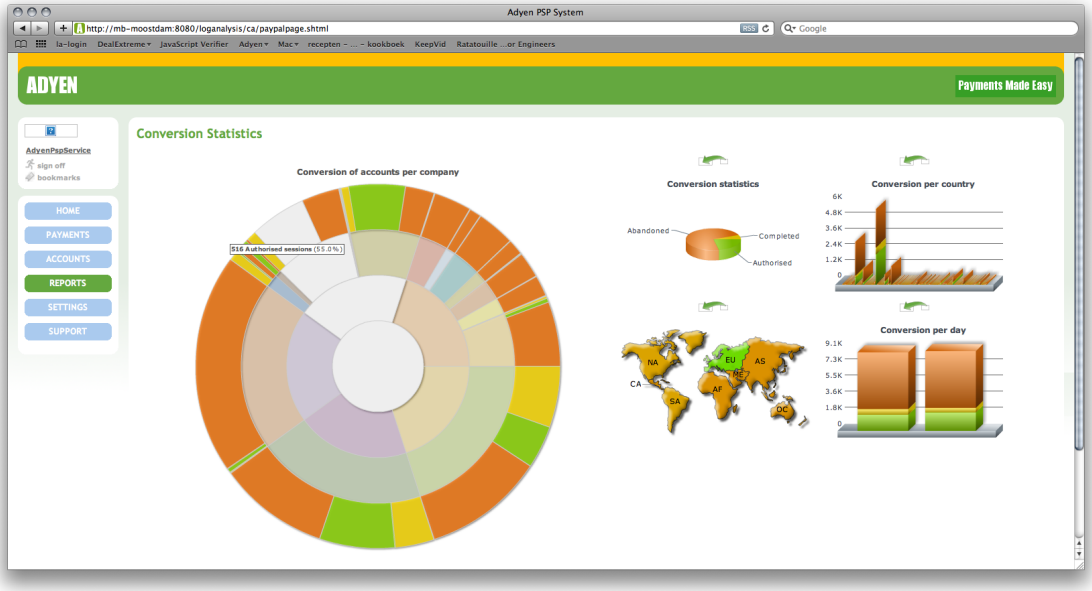
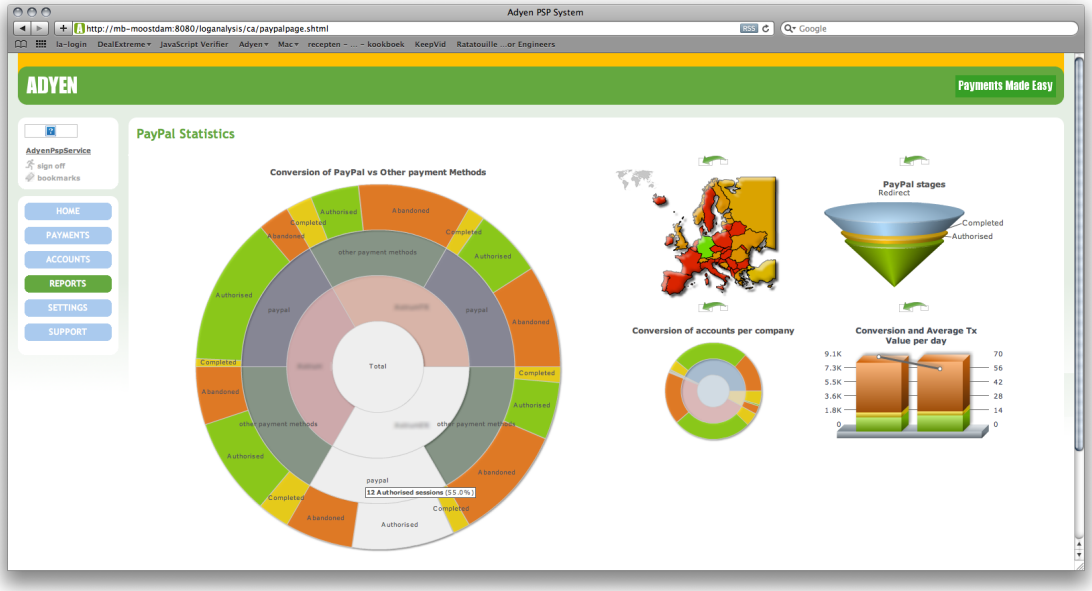


Figure 8.2: Diagram showing the components needed to construct the statistics webpage



(a) Screenshot of the default template with conversion ratio related statistics



(b) Screenshot of the paypal template

Figure 8.3: Screenshots of the statistics page

The filter will be implemented in Java with an adheres() method that takes one payment session as argument, this method returns whether the supplied session should be filtered out or persisted. Using this scheme each request to filter an in-memory dataset will need a linear time investment, since each session should be processed.
The framework design as shown in Diagram (7.5) shows that besides an in-memory log store, the system also provides the ability to attach a database log store. The filter object can be used in combination with the database log store to create query parameters to narrow the result of the query. Since the database has intelligent algorithms, known as query optimiser, this call requires a much smaller time investment. A quote form G. Graefe's article [12] clearly shows that the database provides much more sophisticated access to data than any datastructure provided by the Java platform.

> *A query optimizer is an expert system that finds the best plan given the query semantics, the current database state, and the capabilities of the query evaluation system.*

8.2.5 Statistics webpage and XML web service

For a smooth and responsive webpage asynchronous updating of the page content is a must. To realise this, client-side scripting is used to do all the template manipulations and navigation using Ajax [35] calls. This functionality is located in a single javascript [37] file (FusionOperations.js) as depicted in Diagram 8.2. The Ajax calls make use of the GET method as specified in the HTTP/1.1 Specification Memo [9] as: The GET method retrieves whatever information is identified by the Request-URI.

To create the layout and elements of the page Velocity [5] is used. Velocity is able to parse a web page template with references to objects defined in Java code, hereby creating dynamic web pages. Using a single 'velocity template' we can dynamically add all the statistics from the requested 'chart template'.

The result is a neat looking web page, with asynchronous functionality from the referenced FusionOperations javascript. The statistics data is however not available client-side, so a way of acquiring this data from the server is still needed, in a way that requires no page refresh (postback). This is where the chartxml webservice comes in. Diagram 8.4 shows that the service takes a statistic type, and optionally a chart type and filter and returns an XML configuration stream for the Fusion Charts. This stream is then used to dynamically update the charts on display. An added bonus to having a webservice is that there is a single URL to access all statistical data, thereby facilitating the retrieval of raw XML for debugging purposes.

To generate the XML configuration files for FusionCharts the XStream [6] framework is used. XStream uses an object representation (model) to generate XML output. As a matter of fact XStream is able to serialise any Java object to XML by making nodes from objects and attributes from primitives (and Strings). Because the XStream uses an object model in Java code, comments can be added to the classes and methods of the model which is very useful when dealing with the many properties that can be configured for the FusionCharts.

Most development platforms have auto-complete functionality so the developer can readily see the properties available for the chart you're configuring. An instance of the model is therefore easily created, but it's even more easy to generate XML output. We simply have to supply our model instance to the XStream facade and it gives us XML back.

8.3 Implementation

This section describes the implementation process, visual results and how to extend the user interface with new functionality. The process of implementation will give the reader an idea of benefits and the problems that were encountered. It is described in this section since the quality of the back-end design becomes especially clear when implementing the front-end of the system.

The results of the implementation will encompass the web user interface capabilities. Finally the efforts that were needed to create the user interface will show what is needed to extend the user interface.

8.3.1 Process

There were a few hurdles in the implementation process for the statistics webpage. At first the filter was visualised in the page itself and for each manipulation of the filter values the webpage would post back to the server. The server then calculates how the changes should be reflected in the user interface and returns a response page to the client.

This approach is however not compatible with the asynchronously functioning ChartOperations javascript. Therefore the filter was put into a form which can be posted back to the server, but the reply of the server normally is an updated version of the same page (reflecting the requested changes). There is however a more responsive solution; by making use of the Struts2 framework.

The Struts2 framework is able to return any result in response to a request from the client. So the result of any filter form request is now a JSON [17] response with all the filter settings contained in it. By making use of the Prototype [33] javascript framework the JSON reply is parsed into a javascript object, which in turn is used to update the filter dynamically.

A whole other obstacle was the peculiar choice of the Fusion Maps creators to use 2 letter country codes that do not adhere to the ISO standard [16]. This obstacle was easily overcome by implementing an enum class with all 2 letter country codes as constants and adding a mapping to the fusion map county id's, a mapping to the corresponding region enum is also added. This setup can therefore be used to return a fusion maps country id, and a region on input of a country code, but also all fusion maps country id's for a given region. The latter can be used to populate the world map and generate statistical data for individual regions.

Besides the hurdles there were also some really big benefits from our system design. As is discussed in Chapter 7 the system was initially going to work with an in-memory log store and would eventually switch to the database interface when it was ready. When the design of the statistics web page was already in progress (using the in-memory log store), the database log store

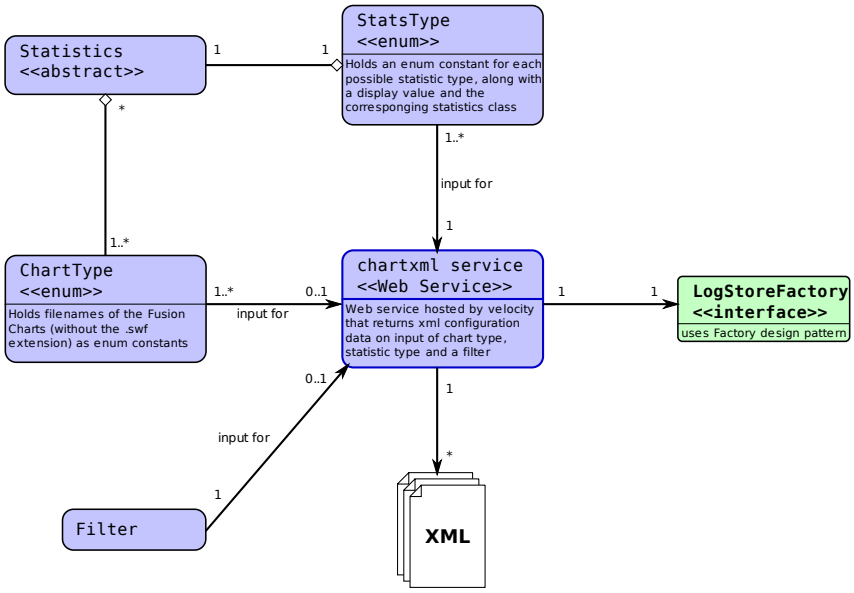


Figure 8.4: Web service class diagram (diagram represents multiple requests to the service)

interface was finalised. The system design proved to be very effective, because only a single line of code needed to be changed to switch the entire system to the database log store. Resulting in much more responsive web requests without any negative side effects or errors.

8.3.2 Results

The web user interface that was developed to give a glimpse of the potential of the conversion analysis framework. This section will give a short overview of the elements currently available on the statistics webpage. On the highest level there are two templates available as can be seen in figure 8.3.

Furthermore there is a map that can be navigated by clicking on countries and regions on the world map, in result all the statistics on the page are filtered to reflect the currently selected location. Currently the following locations are available besides a world map:

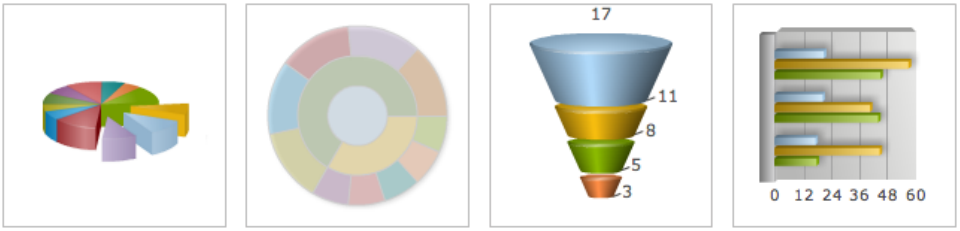
- Asia
- Europe
- Africa
- North America
- South America
- Central America
- Oceania
- Middle East

For each of these regions there is a map available with all the countries in that region, so the statistics can be narrowed to country level. Furthermore there are 4 different charts available on the webpage, Figure 8.5 gives an example for each of these.

8.3.3 Extend the user interface

This last section will give a brief explanation of the efforts needed to implement new templates, charts or statistics into the current conversion analysis framework. Starting from the highest level:

Chart templates: To create a new template very little effort is needed, especially when existing statistics are used. All that needs to be done is to create a new chart template class by extending the ChartTemplate base class. Within this class a list of statistics is build and default settings are defined. Finally in the conversion web page action class the chartTemplate property needs to be set to the new template.



(a) 3D pie chart (b) Multi-level pie chart (c) 3D funnel diagram (d) 3D bar chart

Figure 8.5: Charts currently available on statistics web page

Chart types: Even simpler is the implementation of a new chart type, all that needs to be done is to add the chart flash object into the project charts folder and add an enum constant with the filename to the ChartType enum.

It must noted however, that when the API of the chart differs from the object model that is already available for the already supported charts, it must be extended first in order to generate valid XML configuration files. The fusion charts object model is created in such a way that it can easily be extended and has documentation (javadoc) for each property that is defined.

Statistic types: Creating an entirely new statistic is also easy. The statistics class needs to be extended and an implementation must be given for the abstract methods. Alternatively an already implemented statistic can be extended so implementations of some abstract methods can be reused.

When extending directly from the base class a little more programming skill is required. This is because a (possibly hierarchical) map has to be created to store the statistics. This can be achieved using the session processor (in order to be backwards compatible with the memory log store) or the query processor or both methods. An hierarchical key-value map datastructure is available in the framework to help accomplish this.

In addition the XML configuration object model needs to be created. Using the existing object model makes this task almost trivial. Finally an enum constant needs to be added to the StatsType, with a display name and a reference to the newly created statistics class.

8.3.4 Performance enhancement

The aggregation step as described in Section 7.3.3 improved the performance of the system dramatically. While the queries that were executed on the server to retrieve statistics were analysed to improve the performance, it was noticed that some queries were executed repetitively, because the same statistics are often retrieved multiple times throughout the the user interface.

To deal with this behaviour a cache with the results of the queries was built. This cache uses the Least Recently Used (LRU) approach which proposes evicting the item from the cache that was referenced longest ago when the cache is full. Kleinberg and Tardos describe why this approach is useful [20].

> *It is effective because applications generally exhibit locality of reference: a running program will generally keep accessing the things it has just been accessing.*

With this caching mechanism in place the performance of the user interface improved significantly.

Chapter 9

Software Quality and Testing

In order to improve the quality of the software, testing is needed. A first step in this process is performing unit test. After the unit test have been done additional techniques can be used like code coverage analysis and regression testing.

9.1 Unit testing

For the unit testing we used the JUnit framework [18] which was already used in the development process at Adyen. We structured our project so that test classes are in the same packages as the classes under test, but in a different directory as explained in JUnit in Action [23]. Figure 9.1 shows this setup. Because the state machines are a very important building block of our application we started with unit testing the state machines.

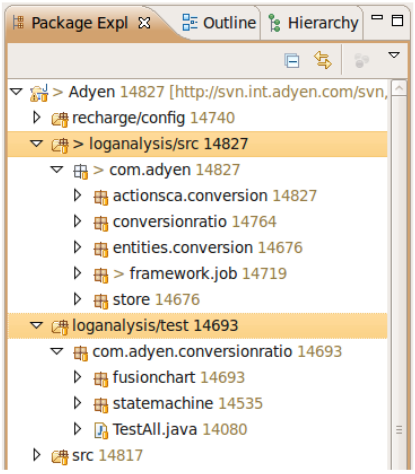


Figure 9.1: Screenshot of directory structure

9.2 Code coverage analysis

Besides testing the correct behaviour of the state machines we also wanted to see how well our code is covered by these tests. For this we used the EclEmma plug-in [25] for the Eclipse IDE [32]. This plug-in marks the source code with different colours depending on how well the statements

are covered. Figure 9.2 also shows that it displays the percentage of statements covered per class and package.

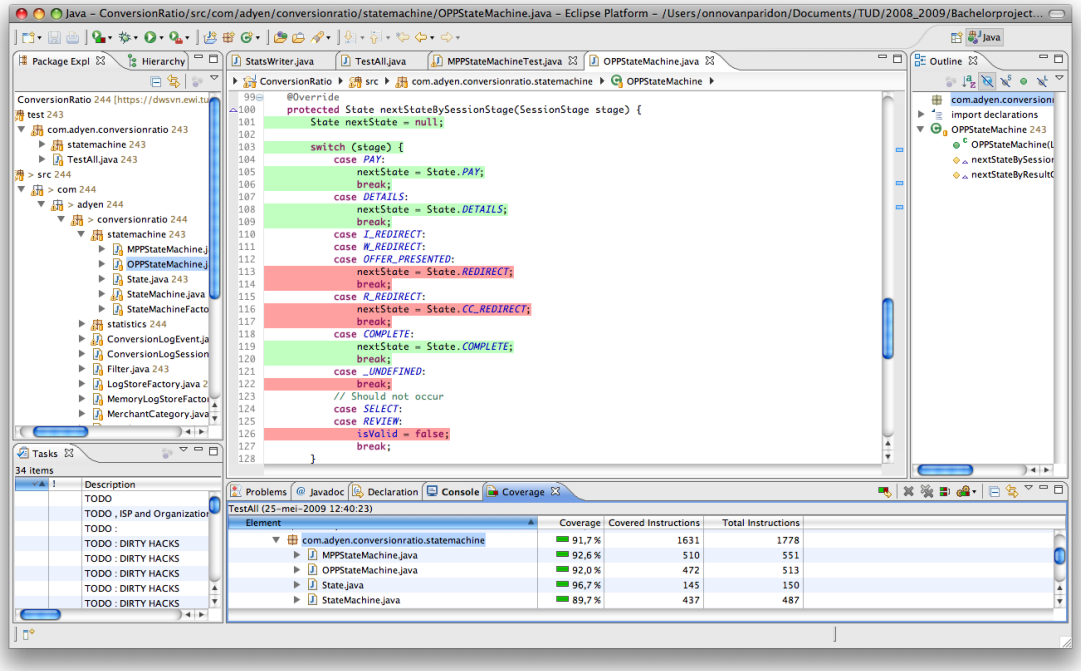


Figure 9.2: Screenshot of coverage analysis of OPP Statemachine

9.3 Regression testing

Regression is the deterioration of software due to changes made to the code through its lifetime. To prevent a drop in quality regression testing can be used. When implementing the state machines to validate the payment sessions, we found that not all sessions adhered to the desired behaviour as indicated in State Diagrams 4.1. These sessions took some shortcuts through the state diagram that are not permitted.

Although these sessions did not adhere to our specifications, we wanted the sessions to validate so that they could be used in our preliminary analyses. For this to happen we introduced some 'hacks' to the state machines allowing the shortcuts. When writing the unit tests for the state machines we wanted all the test to succeed even with the hacks in place.

The inclusion of hacks in the state machines and success of the unit tests posed a risk for the quality of the software. To prevent this unwanted behaviour to remain in the future, test classes were created to specifically test the presence of these hacks.

The presence of the hacks can be toggled with a boolean variable. When this variable indicates the hacks should no longer be present in the code, this will be reflected in the results of the test cases. If a shortcut is attempted successfully while the variable indicates that this should not be allowed, the test corresponding to the shortcut will fail.

Chapter 10

Conclusions

"How to measure and present the Conversion Ratio on the Hosted Payment Pages?"

This specific goal for this project was formulated to answer the generic question on how to monitor and present log information. We can conclude there are five essential steps in the process to measure and present log information, in this case the Conversion Ratio of the Hosted Payment Pages. These five steps are:

- What data needs to be logged
- In which format is the data logged
- How to Centralise log data
- How to Store log data
- How to visualise log information

Since every distinct step presents its own generic and specific problems and challenges we will present the (sub) conclusions in the corresponding paragraph.

Adoption of these conclusions resulted in an application which is deployed and used in the Adyen live environment.

10.1 What data needs to be logged

The process of measuring and presenting log information needs to start by determining which information is required. This has a big impact on the reconstruction of what happened out of the independent log entries. As suggested by Reichle et al. also relevant data available from other knowledge bases needs to be taken into consideration.

There are two main conclusions regarding the data that needs to be logged specifically for this case. First the data can only be determined based on use cases since there is no complete set of data that can be logged. Based on the defined use cases, and validation results of the state machines, the following fields are identified:

- Previous stage, because the HPP are stateless, this makes it possible to reconstruct the exact path taken in the transaction.
- IP address of the shopper, which can be matched against a country database to determine the shopper locale.
- User agent, can give more information about the shopper.

- Error messages the shopper encounters, so decisions of the shoppers can be examined in more detail.

Second the calls to the log need to be implemented in a Struts2 Interceptor. This will ensure a conversion log entry is submitted with every request and response to the HPP.

Both conclusions have already led to changes in the live environment of the Adyen platform, making sure all data required to accurately present log statistics is present.

10.2 In which format is the data logged

In order to facilitate automatic parsing of the log files a machine readable format is required. To determine the best format six formats are evaluated according to five different criteria:

- Human readability
- Searching
- Processing
- Distributing
- Log4j support

From these analyses we conclude that logging to a local database in combination with a CSV backup is the best approach for a generic logging framework. It is thus suggested to write log data to two output formats. The searchability and readability of the database exceeds that of any file format. This can be crucial in an emergency situation of a 24 / 7 online service when a developer needs to determine the root cause as quickly as possible. The reason why a file format backup is required is that it is more reliable and robust compared to a database. Log data must of course be available in all cases, especially in erroneous situations.

Because of the time constraints and the big impact of this proposed change, the decision has been made to use the existing Log4j CSV log files for logging the conversion on the HPP. The performance implications also need to be researched in more depth, but are out of the scope of this project.

10.3 How to Centralise log data

The centralisation step is vital in a SOA, because services run on different (hardware) instances and analysis has to be performed with all data available. Four possible centralisation strategies have been distinguished, as illustrated in Diagram 6.1.

When keeping in mind the protocol robustness, network capacity, archiving strategy and security, database replication is the preferred strategy. This strategy spreads the load and (secure) replication is a standard feature available in most database engines. In order to formulate a conclusive statement about the feasibility of this solution, the performance implications of logging to two formats, as discussed in step two, need to be researched.

For this specific case a decision has been made for step two to work with CSV files and not (yet) store the data in a database. For now the files are periodically transferred to the central log analysis application using a batch script.

10.4 How to store log data

For the storage two important aspects need to be discussed. First the lay-out of the raw data which is gathered on the central system and secondly how the data is processed and enriched to have everything available for the final visualisation step.

A generic log data entry consists of a general (Log4j) part and a specific message part. The individual entries should therefore be stored in two parts in a database. This clear separation allows the log analysis framework to be used for any specific log data thereby allowing easy processing for various applications. In this case the message part is composed out of HPP conversion data.

Reichle et al. defined five steps that are required to merge additional data from other knowledge bases with the logged data. For performance reasons we propose a different order for the log analysis framework. The validation step is moved from the last step to the third step, which ensures that additional data is processed only for valid entries.

The specific application which processes the raw HPP conversion data has been implemented using an iterative approach. This iterative approach first started with the implementation of a straight forward in-memory storage method, resulting in usable data at an early stage. With this preliminary tool crucial insight was gained into the logging of the payment process. When working with larger datasets it was clear that we needed to switch to a database.

The design of this application uses therefore a LogStoreFactory Interface which proves to be very effective, since changing from this line:

```
logStore = new MemoryLogStoreFactory();
```

to this line:

```
logStore = new DbLogStoreFactory();
```

is enough to switch from in-memory storage to database storage.

The visualisation of the HPP conversion data also requires an additional aggregation step for performance reasons. An aggregated table is updated whenever new data has been parsed. This is done in a way that still provides al the necessary information for a specific use case but reduces the entries in the table to 7.7% of its original size. Please note different use cases might require different aggregation tables. The exact performance implications and archiving strategies are outside the scope of this project.

10.5 How to visualise log data

Visualisation of log data is very specific to a project. Some generic lessons can however be learned from the HPP conversion case. Specifying the use cases and creating mock-up screens need to be done in early stages, since they impact decisions made in the other four steps. It is therefore also very important to involve the user who will work with the data when specifying the use cases. The benefit of having a visually pleasing presentation should also not be underestimated, a consistent colour palate makes for a clean look and gives the means for easy understanding of the data Last but not least it is important that the visualisation is very flexible since different users might want to use different views on the data.

As discussed not all use cases can be foreseen and we have therefore implemented a template framework for the HPP conversion analyses. A template is a web page presenting the data required for one type of use case and consists out of different graphs all using the same aggregation table. The data is stored in a cache so real-time interaction is possible.

Showing multiple graphs with different (presentations of the) data on one page proved to be very important in order to understand the impact of the conversion on the HPP. The user is presented with data comparisons so he gets a clear indication of how to improve the conversion.

10.6 Goals

In the first chapter three goals were set that needed to be met near the end of the project. What follows is an evaluation of these goals along with some insight in to how these goals were realised, what the obstacles were encountered in the process and how these were overcome.

During the project extra consideration is put into determining the log format and how to gather the log data on a central location. This is done because Adyen has plans for a Masters thesis/project to completely automate the monitoring, reporting and notification of its entire system using the logged data as discussed in Section 3.2. The solution that has been devised is generic so it can be used beyond conversion logging.

10.6.1 Live data feed for merchants

During the development process a few merchants were invited to participate in the beta phase. All of these merchants were very enthusiastic about the possibilities the system gives them. As explained in Chapter 1, often used tools like Google Analytics can not be used on the HPP. The developed framework enables them to get the information they want. With detailed information about their conversion ratio (per country/payment method/etc.) merchants have already changed their business and marketing strategies. On top of this even the Adyen team has used this beta tool to make decisions in multiple cases.

The system has been designed to visualise statistics on the payment process for all merchants. Therefore creating a 'semi-realtime' data-feed for one merchant is easy. There is a generic filter available which is, among other parameters, able to filter on any number of merchants. This filter therefore only narrows the dataset that needs to be processed, resulting in a smooth user interface for a merchant or a company (consisting of multiple merchants).

10.6.2 Data for marketing material

Adyen can assist with their expertise and the conversion analysis tool to optimise the conversion, promote least cost payment methods and fight fraud. Furthermore the Adyen sales team uses this tool to promote the product and service that Adyen provides. The new user interface is therefore as helpful to Adyen as it is to the merchants. The system really is a USP (Unique Selling Point) for the Adyen sales team. On multiple occasions during the development of the system, sales employees used the statistical data and its graphical representations to pull in new clients.

10.6.3 Form a basis for an article

With the research done in this paper and with the conversion analysis tool, a basis has been formed for a newspaper article, as well as a scientific article. An interesting article for a newspaper concerns the conversion ratio of PayPal payments. This payment method is a high cost payment method which does not perform as well as other (localised) payment methods in the Netherlands. A scientific article can be made on the basis of the generic logging framework and the preparations that have been made during this project for this framework. The conversion logging framework is already unique in the payment industry and a generic logging framework will definitely provide interesting insights for a paper when combined with a Masters thesis at Adyen.

Chapter 11

Recommendations

In this project research has been conducted on a generic logging framework with conversion logging as the specific use case. During this process a broad range of topics was touched, but not all could be researched within the scope of this project. In this chapter several recommendations are made for topics that should be investigated further.

11.1 Generic scalable logging framework

Within the project an initial analysis was made of the different logging formats. Although this analysis indicated the formats to be used, the results may not be applicable to all of the logging that takes place within Adyen's platform. More research into a generic logging framework that can be used throughout the platform as a whole and possibly even outside of Adyen is recommended.

As discussed in Chapter 7 and 8, special provisions were made to improve the performance and responsiveness of the system. Although these factors are now between the desired bounds, it is recommended that more research is done to analyse and possibly improve the performance and responsiveness of the system. This should be on multiple levels ranging database queries to page load times.

As is the case with more aspects of the project, this research should be generic enough so that it can be deployed and applied to the Adyen platform as a whole. Because of the large scope of such a project, this would be highly suitable for a Master project.

11.2 A / B testing

By setting up this A / B framework an answer can be given to the main question in the orientation report (Appendix B). After the orientation phase it became clear that A / B testing should not have priority, but should rather be facilitated in a more generic conversion analysis framework.

The conversion analysis framework developed in this project brings great insight into the conversion ratio. The results of the analyses can be filtered by the Skin that was used to perform the payment. This way merchants can compare the conversion rates of different Skins. It allows them to test differences in the lay-out, security certificates, the order of payment methods, additional costs, MPP vs. OPP et cetera.

It would by highly beneficial for merchants if they could indicate which Skins should be used for A / B testing and what percentage of the visitors gets which Skin. A framework that enables this should also allow the results to be placed side by side to make comparison easier. The scope of this project would make it very suitable for a bachelor project.

11.3 Pattern recognition

In the current setup, about 70% of the sessions can be reconstructed successfully from the log files. It is highly recommended to analyse the sessions that could not be reconstructed and to research the cause(s) in order to develop a more robust algorithm that recognises the pattern formed by a payment session.

Pattern recognition is not only useful within the context of the conversion logs. Automatic recognition of patterns can help to monitor the Adyen platform as a whole. When a suspicious pattern occurs, a developer or administrator can be notified in advance before the actual error occurs. Some work on this topic has already been conducted within Adyen through the use of special log4j appenders.

Appendix A

Glossary

API Application Programming Interface

CSS Cascading Style Sheets

CSV Comma Separated Values

DSS Data Security Standard

HPP Hosted Payment Pages

HTML HyperText Markup Language

JSON JavaScript Object Notation

LRU Least Recently Used

MPP Multi Page Payment

MVC Model View Controller

OPP One Page Payment

PCI Payment Card Industry

PSP Payment Service Provider

SLA Service Level Agreement

SSH Secure Shell

SSL Secure Sockets Layer

URI Uniform Resource Identifier, can be a location (URL) or name (URN)

USP Unique Selling Point

VPN Virtual Private Network

XML Extensible Markup Language

Appendix B

Orientation report

Bachelor project: Conversion Ratio Orientation Report

Mark Oostdam
1174681

Onno van Paridon
1100211

May 8, 2009

Faculty EEMCS, Delft University of Technology
Delft, the Netherlands

Adyen Enterprise Payment Services
Amsterdam, the Netherlands

Contents

Chapter 1

Introduction

Adyen is a Payment Service Provider (PSP), started in 2006 by industry specialists. Adyen focusses on providing one online payment interface for the European and North America online market. The role of Adyen in the online payment process is that online shops (merchants) only have to connect to Adyen in order to support multiple payment methods, like credit cards and local payment methods like iDeal, bank transfers, etcetera. In addition to local payment methods international payment methods can also be made available to the merchants audience. With offices in Amsterdam, London and Los Gatos (CA), Adyen takes care of all the communication and money flows for the merchants for all the payment method providers (acquirers) and offers her merchants one unified type of reporting and money flow with one contract.

Adyen's business model is based on providing innovative features to its merchants at a minimum cost. Adyen is a collecting PSP, meaning that Adyen has contracts with multiple financial institutions. These institutions fund the Adyen Client Management Foundation, which then distributes these funds to Adyen's merchants (Adyen is under supervision by the Dutch Banking Authorities). For the merchants the advantage of this model is that Adyen reconciles the payments (matching the incoming money with the outstanding authorised transactions) and provides one money stream and one type of reporting to the merchants.

The cost for the transaction are twofold for the merchants, a fee to the financial institution and the Adyen processing fee of a maximum of 10 cents per transaction. There are no monthly charges or setup fees for the merchants. The costs of the financial institution are transparent and charged directly to the merchant. Because Adyen has the contracts with the financial institutions it processes large volumes and has a stronger position in negotiating lower costs. Since Adyens margin is only the small processing fee, the only way for Adyen to be successful is to process large volumes of transactions. One of the other benefits, besides providing a low price to the merchants, is that both the merchant and Adyen strive to increase their transaction volume. Giving insight in the conversion rates is not only an innovative feature, it also provides knowledge on how to increase the number of payments and thus generate higher revenues.

Converting website visitors to actual customers spending money at the merchant's site is the challenge of every merchant conducting business on the internet. The design of the site, ease of use of the shopping cart, data and payment entry pages should be no hurdles for the merchant's customers in finalising the sale. The more steps in the merchant's payment process, the less chance that customers will complete the payment successfully. Payment pages that are difficult to understand, counterintuitive and hard to navigate are a major source for fallout which decreases the merchant's online revenue unnecessarily.

The shoppers are redirected from the merchant's website to the merchant's Hosted Payment Pages (HPP) hosted at Adyen's platform (Figure 1.1). Adyen is therefore determined in making the payment process as easy and intuitively as possible, reducing the level of abandonment to a minimum.

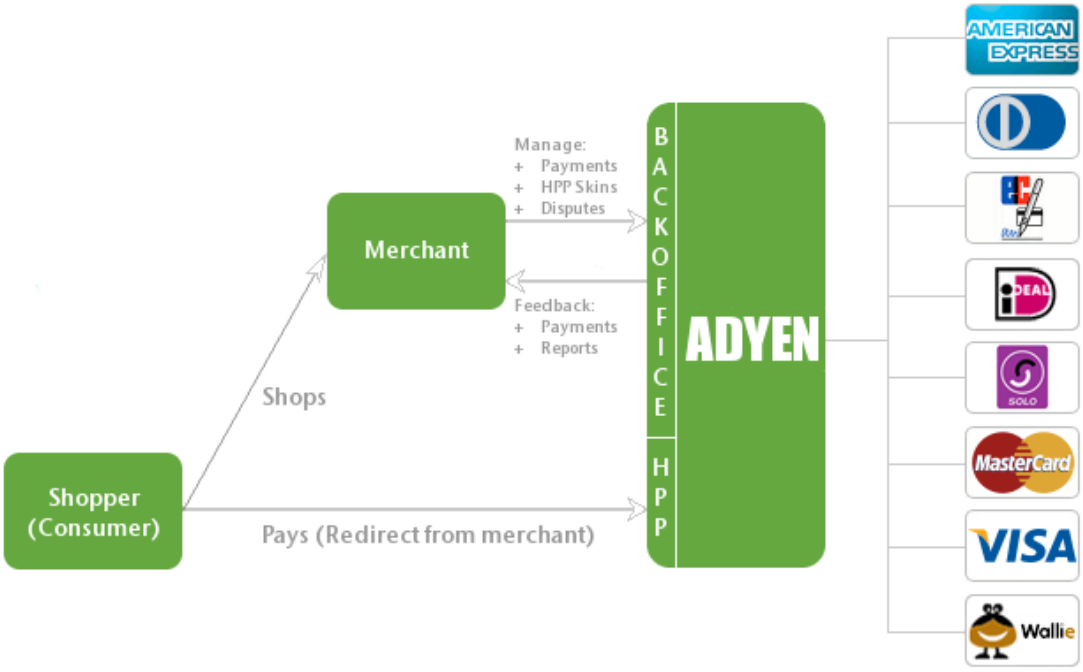


Figure 1.1: Adyen system overview

Chapter 2

Goals

The goal of the Conversion Ratio project at Adyen is to gather information about the course of the payment sessions of consumers buying at online shops. As stated in the introduction not all of the initiated payment sessions will result in successful payments. Some consumers will abandon the payment pages prematurely and abort the payment session. Information about the completed intermediate steps, acquired by logging, can be used as source for statistical analysis. Results of the analyses can be used for several purposes. In this chapter we will discuss some of the possibilities.

2.1 Live data feed for merchants

Adyen provides a merchant backoffice for their clients. Part of the backoffice is a dashboard interface (Figure 2.1) where drag-able widgets enable the merchant to view live data about the payments that have been performed.

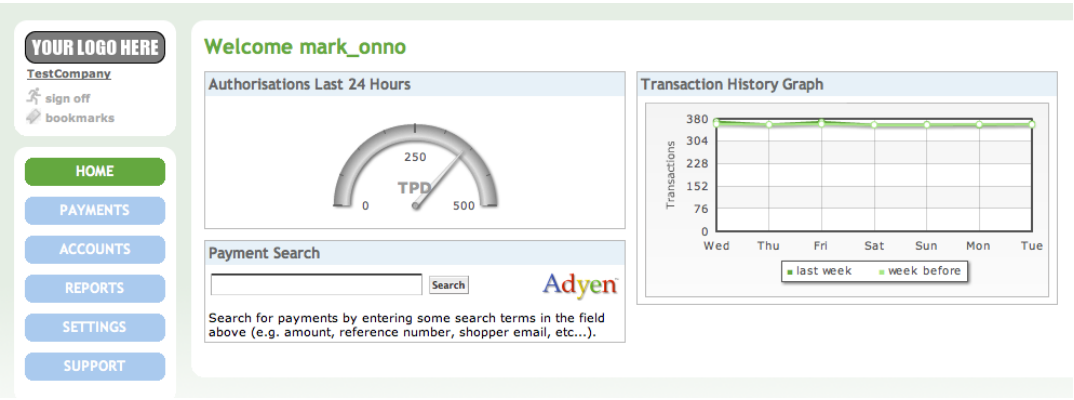


Figure 2.1: Dashboard interface of the Adyen merchant backoffice

One of the uses of the statistical information is to provide feedback to the merchant in the form of widgets. A possible widget would be a funnel diagram (Figure 2.2) showing the conversion ratio for a payment method.

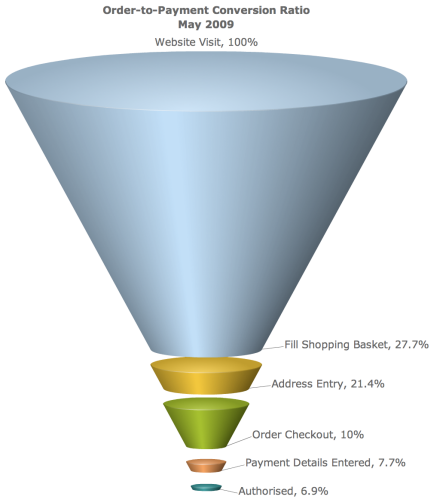


Figure 2.2: Order to payment conversion ratio

2.2 Data for marketing material

Marketing material should emphasise the unique selling points of a company. One of the selling points of Adyen is the partnership it forms with its merchants. Improvement in the levels of abandonment and key insights in consumer behaviour are of beneficial for both Adyen and its merchants. The information can therefore be used as a promotional tool.

2.3 Form a basis for an article

Adyen has an agreement with two Dutch newspapers to publish an article near the end of the project. Results of the analyses and the tests can be used to enrich the articles and promote Adyen.

Chapter 3

What's in place

3.1 Infrastructure

In order to achieve high availability the HPP run on a cluster of identically configured redundant systems as indicated by the simplified overview in Figure 3.1. Because of the statelessness of the HPP the system is highly scalable. These systems run the Apache Tomcat [3] web server to serve the pages to the shoppers. They hand off the requests to other machines in the system to do the risk analysis and the actual payment. These machines are represented by CS in Figure 3.1 and are compromised from multiple hardware systems running multiple applications.

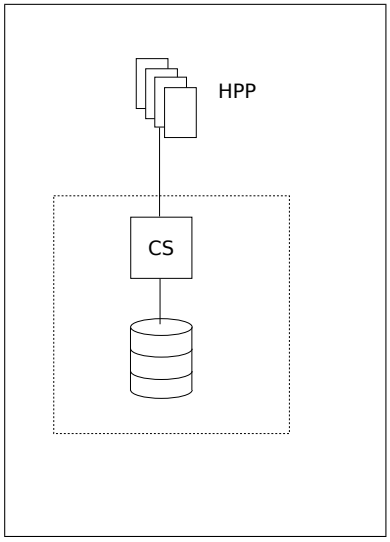


Figure 3.1: Simplified overview of infrastructure at Adyen

3.2 Skins

The payment pages that are presented to the shoppers can be customised by the merchant through skins (Figure 3.2). Skins are ZIP files that contain multiple image, HTML, CSS, Javascript and resource files.

Using skins merchants can not only adjust the look and feel of the pages to match their colouring scheme and have their logo appear, but they can also adjust which of the payment methods will appear and their order. Also the way of interaction can be altered by a skin from a multi page

payment, where several pages will be shown sequentially, to a one page payment where all the interaction will take place on a single page.

Because merchants can tailor these files entirely to their needs the payment pages can completely mimic the layout of the online shop through skins.

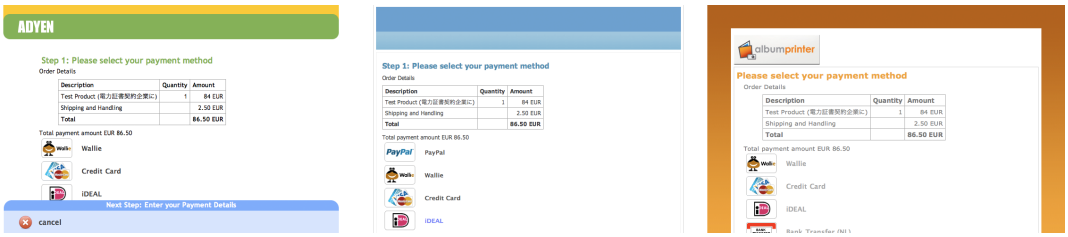


Figure 3.2: Examples of skins for the HPP

3.3 Flow

As stated in section 3.2 the skins can define the way of interaction. When Multi Page Payment (MPP) is used, multiple pages are shown to the shopper in sequence in order to complete the transaction. The payment process consist of the following stages:

- Payment method selection
- Enter payment details
- Review order details
- Result

Each of the pages in Figure 3.3 represents one of these stages.

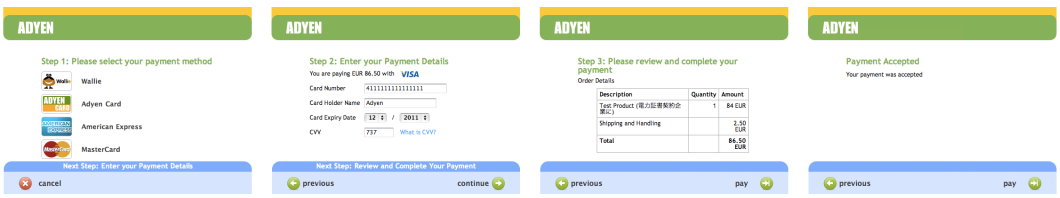


Figure 3.3: Flow when using Multi Page Payment

When using One Page Payment (OPP) all the user interaction takes place on a single page. If the payment method support this the first three stages are combined on one page as illustrated in Figure 3.4.

Figure 3.4: Flow when using One Page Payment

3.4 Log files

In the current system logging is being performed by the log4j framework [2]. The main reason for the application logging that is in place is to detect errors and warnings in the application. Some additional information for the context is also logged to trace the errors and warnings. Besides this log the behaviour of the shoppers is recorded in a conversion log. All actions that result in a new page are currently logged except for error and warning pages. Adyen can therefore use these logs to distil the conversion ratio on the payment sessions that occur within the HPP.

Listing 3.1 shows the first 4 lines of a sample log file. The log file contains information from both the log4j framework and a message part defined by de application.

Listing 3.1: Sample existing logfile

```
1 2009-03-28 00:05:01,004 INFO [com.adyen.actions.hpp.conversion.
    ConversionLogger] {LogTimer} -- Something to roll the logs --
2 2009-03-28 01:18:46,786 INFO [com.adyen.actions.hpp.conversion.
    ConversionLogger] {TP-Processor2} "VodafoneDevelopment","3t8vWW2d","
    dhqxPh+RqLGnzI3BCHMHAWfVrII=","1201408","2009-03-28T00:18:46.78Z
    ",23499,"EUR","review","CompleteCardPayment","mc",,,
3 2009-03-28 01:18:46,786 INFO [com.adyen.actions.hpp.conversion.
    ConversionLogger] {TP-Processor2} "VodafoneDevelopment","3t8vWW2d","
    dhqxPh+RqLGnzI3BCHMHAWfVrII=","1201408","2009-03-28T00:18:46.78Z
    ",23499,"EUR","review","HandlePaymentAction","mc",,,
4 2009-03-28 01:18:54,039 INFO [com.adyen.actions.hpp.conversion.
    ConversionLogger] {TP-Processor2} "VodafoneDevelopment","3t8vWW2d","8
    JRJbd0+QCZCzjrgl4M6ti7R+ww=","1201408","2009-03-28T00:18:54.03Z",23499,"
    EUR","details","PaymentDetails","mc",,,
```

Information supplied by the log4j framework is:

- The exact **DATE** in a unified format (a.k.a. timestamp).
- The severity **LEVEL** assigned to the message, for example INFO, WARNING, ERROR et cetera.
- The **CLASS** which is requested to perform the logging.
- The **THREAD** in which the logger is called.

Information provided by the application is:

- The **MERCHANT** requesting the payment.
- The **SKIN** used to customise the HPP. The code identifying the skin is unique and generated at design time.
- The **SESSIONID** identifying the session is generated by a hash algorithm based on fields provided by the merchant.
- The **MERCHANT REFERENCE** is a reference provided by the merchant to identify the transaction.
- The **TIME** until the payment session is valid as specified by the merchant.
- The **AMOUNT** of the transaction.
- The **CURRENCY** used in the transaction.
- The current **STAGE** the transaction is in.
- The name of the **CLASS** currently handling the transaction.
- The **METHOD** used to perform the payment for example MasterCard, VISA, iDeal et cetera.
- The **RESULT CODE** of the transaction as issued by acquirer, for example AUTHORISED, REJECTED et cetera.
- The **PSP REFERENCE** is a unique code identifying a transaction within Adyen's system.

The current log files can already be used to identify differences in conversion rate between payment methods. Analysing the conversion log can bring a greater insight into the behaviour of shoppers. For example, as stated in section 3.2 skins can influence the user experience. The results of the analyses might therefore indicate that some skins have a greater risk of the shopper abandoning the payment prematurely. Particularly interesting to see for example is how the conversion ratio differs between MPP and OPP because less pages will be shown when using OPP as explained in section 3.3.

Chapter 4

Problem analysis

Before a browser converts into a buyer there are a lot of steps to be taken. In each of these steps a portion of shoppers discontinues their purchase and the remaining part finalises the transaction. The conversion ratio is the ratio of shoppers that finalise their transaction in comparison to the number of shoppers that start a transaction (Figure 4.1). Raising the conversion ratio would mean more shoppers finalise their transactions. Therefore the merchant's goal is to maximise the number of successful payments.

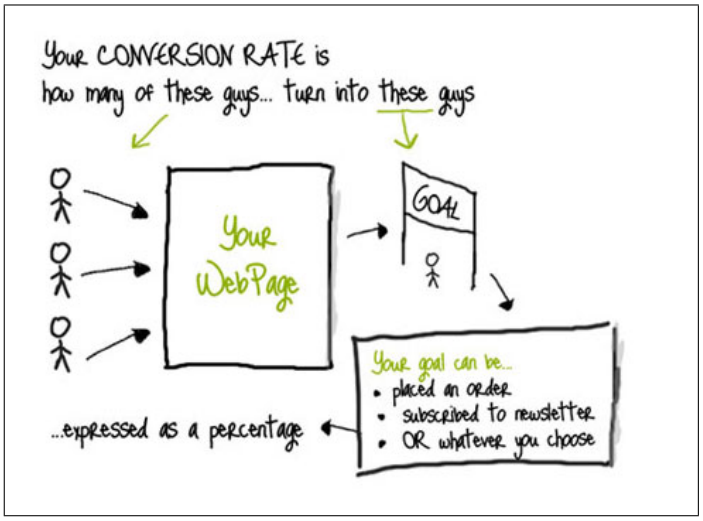


Figure 4.1: Conversion ratio explained [7]

When a PSP is used, the PSP takes over the payment process from the merchant. Adyen is interested in how the conversion ratio is affected by for example banners, menus, movement and animation, number of pages and difference in layout between the merchant's pages and the HPP. These are the main things that can be modified through the use of an Adyen skin.

As stated in chapter 1 Adyen's business model is based on charging the merchant a fixed amount per transaction. Greater insight in how the conversion rate is influenced by these factors and ultimately higher conversion rates mean that Adyen can provide a better service towards its merchants and a higher turnover.

The main question that we want to answer in our project is:

"How to measure and present the influence of different Skin aspects on the conversion ratio?"

To answer this question we will look at the different aspects that play a role in the project in the following sections. We will start by reviewing the log files which are currently in use and the data they contain as indicated by step {1} in Figure 4.2. Step {2} concerns the format in which the data can be stored. The gathering of the available information, step {3}, will be analysed in section 4.3. After that we will look at how to store and archive the information on the central system, step {4}, and perform analyses on the information. Finally we will examine how the results can be presented visually as indicated by step {5}.

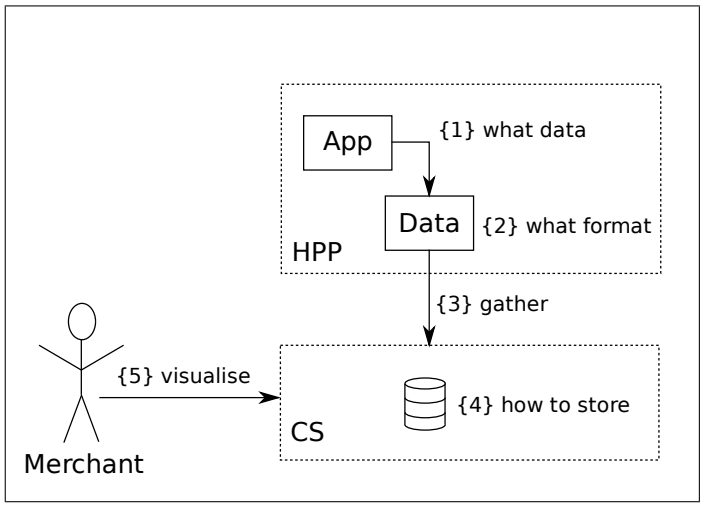


Figure 4.2: Overview problem analysis

4.1 Using the log files

As stated in section 3.4 the log files contain various information. In order to see what information is present and how this can be used we built a simple parser to generate some statistics. A first problem we encountered was the occurrence of delimiters in fields as described in the paper by Reicle et al. [14].

The main question regarding the current log files is whether all necessary information is available. At the moment the IP addresses of the shoppers are not recorded in the log files. This information in combination with a DNS lookup as used in the system proposed by Reichle et al. (Figure 4.3) can enrich the logged information with the company name or the country name. This knowledge can give great insight into the demographic distribution of the shoppers. Other external interesting data which can enrich the conversion data will be researched.

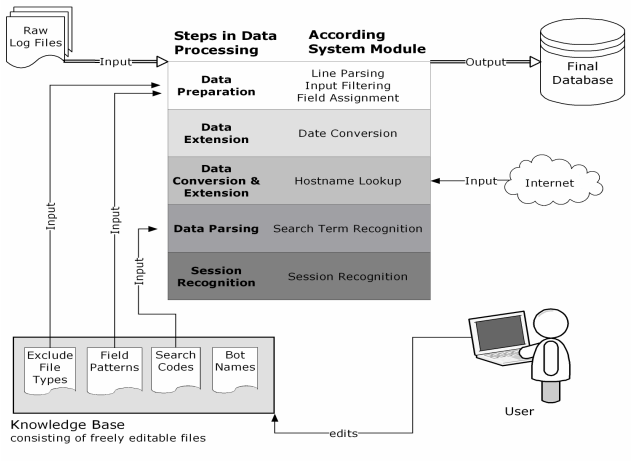


Figure 4.3: Overview of steps in data processing by Reichle et al.

4.2 Make conversion log files machine readable

Currently the log files that are generated by the log4j framework are human readable. However they are not comprehensible for human beings, there is simply to much data to get an idea of what is happening. Because of the high volume of data we need a automated way to parse and process the log information. Therefore we need to make the conversion data machine readable. In order to use the conversion data for statistical analysis we need to read them into a structured format.

Although the current conversion logs are parsable to some extent there are some difficulties associated with this approach. After writing a simple parser to see if we could parse the current log files and to try and get the first simple statistics we quickly noticed that the current log format isn't suitable for our needs. The possible occurrence of delimiters in fields and deviations from the standard format make the parsing hard. Thus our challenge is to put the logging data into a machine readable format so we are able to order and compare different payment transactions. A few formats we have already considered and will be investigating further are:

1. CSV,
2. XML,
3. directly to a database,
4. directly to a web service,
5. Java property files [13],
6. Serialised Java objects.

4.3 Gather the conversion log data on a central system

The HPP applications run on a cluster of identically configured redundant hardware systems as explained in section 3.1. Because of the statelessness of the HPP each of the intermediate actions (and their corresponding payment page) within the payment session may be handled by a different system. In order to perform useful analyses, the data of the individual systems has to be loaded onto a central system to get a complete overview of all the data.

Because of the high volume of log messages we need to take into account the available bandwidth between the systems and the processing power of particular systems. For instance we don't

want the systems hosting the HPP to stall because they have to load the conversion data to a central system. We also need to consider the required disk space for the logs so the hard disks will not flood in these circumstances.

Security also has to be taken into account due to the fact we are dealing with highly sensitive data. The systems also have to deal with situations where the central system is not available. All these considerations influence the message protocol used and the frequency of the messages.

4.4 Manage statistical information on the central system

The data gathered can be stored on the central system in several ways.

1. In a database system.

2. In files on the filesystem.

3. Create (visual) output immediately from data in memory.

Once the logging information is available from a central system in a structured way, analysis can be performed. Queries to the central system and data-sets for graphical representation have to be defined. Within the logging information a distinction has to be made between information provided by log4j and the information in the message part that is constructed by the application as outlined in section 3.4.

Because of the high volume of logging information special attention has to be paid to optimising the queries so that realtime access is possibe. The high volumes also necessitates designing a archiving strategy to manage all the data coming in and prevent the hard disk from flooding. A third aspect to check is whether all the data is available. If for example the information of one system is missing, the statistics can be off and can give the wrong impression.

A possible optimisation step is to aggregate the information based on common denominator for example session id.

4.5 Visualise statistical information

Once we have our shopper behaviour and transaction information neatly organised, we are then able to do statistical analyses on the information. Subsequently we need a way to visualise these statistics. They need to be visualised in a clear and consistent way such that we can easily recognise the interesting conversion trends within the payment process. Even more important is to identify the bottlenecks in the HPP user interface which cause shoppers to leave before the payment is finalised.

After we have analysed the statistics in some detail we might want to provide some of them to the merchants. For this we need an intuitive and aesthetically pleasing way of presentation. Our goal is that the visualisations are self explanatory so the merchants can use them without training. Figure 4.4 shows the user interface of the first implementation cycle. It is still very basic and will be extended as discussed in section 4.6, but it gives a good impression of what we want to achieve.

The payment result codes encountered in Figure 4.4 are listed below.

- **Authorised**, the transaction is authorised by the acquirer.

- **Completed**, the transaction is either rejected by the acquirer or pending final verdict.

- **Abandoned**, the transaction was abandoned prematurely.

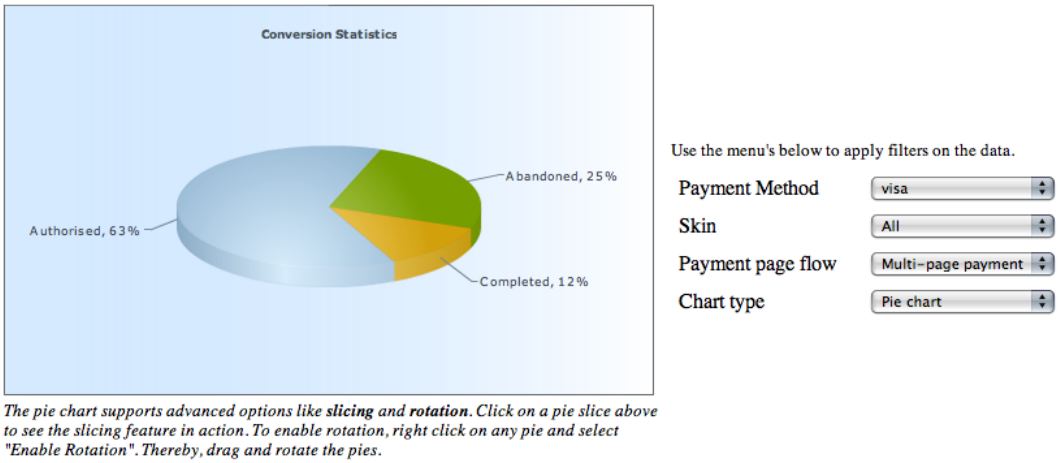


Figure 4.4: Web interface of the first implementation cycle

4.6 Strategy

Because we still need insight in the shopper behaviour in order to formulate proper hypotheses we will use an iterative approach and try to keep our implementation cycles as short as possible. In this way we can take the things we learned at the end of each cycle into account when we update our initial implementation and research. This is why we will start with parsing the conversion logs in their current format, do statistical analysis in memory and provide basic visual representation on our development station before we work on making the log data machine readable.

We will start testing internally with information supplied by the sales team before performing the test externally with merchants. During the design and development we will try to keep things as generic as possible so that parts of the system can be reused.

Chapter 5

Things to learn

5.1 Literature

To our knowledge no research has been done concerning the conversion ratio of the payment process. Some research has been conducted in the field of shopping cart abandonment at online shops. The percentages of online shopping cart abandonment found in literature vary greatly between sources. According to Gutzman [9] the abandonment rate is roughly between 25% and 30% although numbers as high as 70% are found elsewhere.

The paper by Li and Chatterjee [12] mentions some factors that impact shopping cart abandonment:

1. insufficient information on products and shipping costs,
2. limited functionalities on webpages,
3. confusing buttons or icons,
4. low quality of user interface,
5. detailed registration requirement before purchase,
6. unstable or unreliable interactivity at website,
7. multipage time-consuming checkout process.

Eisenberg [8] contradicts the last item and claims that the number of steps in the checkout process does not influence the abandonment rate. His claims are however based on one case. Cohen [6] also mentions points 1, 5 and 7 among others, but adds three interesting factors:

8. use of the shopping cart as a holding bin or wish list,
9. coupon and/or promotion code requests,
10. comparing prices with other retailers.

Some of these items can be used as a starting point for aspects we want to test in the A/B comparisons. In A/B comparisons half of the visitors would be shown Skin A and the other half would be shown Skin B [7]. We could then tell which skin has the highest conversion ratio.

5.2 Technologies needed

In order for the project to succeed several technologies are needed. Most of the technologies listed in this section are already being used in Adyen's products and workflow.

- **log4j** is a Java logging framework.
- **Machine readable formats** are ways to encode information in a form which can be read by a computer and interpreted by software, a few formats are mentioned in chapter 4.2.
- **FusionCharts** [10] are animated and interactive Flash charts for web and desktop applications (source code available).
- **Javascript** [17] is a dynamic scripting language used in web applications. It is mainly used to provide interactivity on websites.
- **XML** [16] is a language to transport and store data. It is an aid for information systems to share data in a structured way.
- **Velocity** [4] is a Java-based template engine that renders data from plain Java objects to text, xml, email, SQL, HTML et cetera (source code available).
- **iBATIS** [1] is a framework to map Java objects to a relational database using a XML descriptor (source code available).
- **Javadoc** [15] is a tool for generating API documentation in HTML format from doc comments in source code (source code available).
- **JUnit** [11] is a Java framework to write repeatable tests (source code available).
- **Clover** [5] is a code coverage tool that shows what parts of the source code are covered by tests.

Appendix A

Glossary

CSS Cascading Style Sheets

CSV Comma Separated Values

HPP Hosted Payment Pages

HTML HyperText Markup Language

MPP Multi Page Payment

OPP One Page Payment

PSP Payment Service Provider

XML Extensible Markup Language

Bibliography

[1] Apache Software Foundation. iBATIS. Available from World Wide Web: http://ibatis.apache.org/.

[2] Apache Software Foundation. log4j. Available from World Wide Web: http://logging.apache.org/log4j/.

[3] Apache Software Foundation. Tomcat. Available from World Wide Web: http://tomcat.apache.org/.

[4] Apache Software Foundation. Velocity. Available from World Wide Web: http://velocity.apache.org/.

[5] Atlassian. Clover. Available from World Wide Web: http://www.atlassian.com/software/clover/.

[6] Heidi Cohen. Shopping cart abandonment and what to do about it. Available from World Wide Web: http://www.clickz.com/3624370.

[7] Conversion Rate Experts. Google website optimizer 101 – a quick-start guide to conversion rate optimization. Available from World Wide Web: http://www.conversion-rate-experts.com/articles/101-google-website-optimizer-tips/.

[8] Bryan Eisenberg. 20 tips to minimize shopping cart abandonment, part 1. Available from World Wide Web: http://www.clickz.com/2245891.

[9] Alexis Gutzman. The truth behind shopping cart abandonment rates. Available from World Wide Web: http://www.ecommerce-guide.com/solutions/technology/article.php/448381.

[10] InfoSoft Global. FusionCharts. Available from World Wide Web: http://www.fusioncharts.com/.

[11] JUnit. JUnit. Available from World Wide Web: http://junit.sourceforge.net/.

[12] Shibo Li and Patrali Chatterjee. Shopping cart abandonment at retail websites - a multi-stage model of online shopping behavior. IEEE International Conference on Data Mining, Springer, 2006.

[13] Sun Microsystems. Java property file. Available from World Wide Web: http://java.sun.com/javase/6/docs/api/java/util/PropertyResourceBundle.html.

[14] Meike Reichle, Petra Perner, and Klaus-Dieter Althoff. Data preparation of web log files for marketing aspects analyses.

[15] Sun Microsystems. Javadoc. Available from World Wide Web: http://java.sun.com/j2se/javadoc/.

[16] W3Schools. Introduction to xml. Available from World Wide Web: http://www.w3schools.com/xml/xml_whatis.asp.

[17] W3Schools. Javascript introduction. Available from World Wide Web: http://www.w3schools.com/js/js_intro.asp.

Appendix C

Data preparation of web log files for marketing aspects analyses

Data Preparation of Web Log Files for Marketing Aspects Analyses

Meike Reichle[1], Petra Perner[1], and Klaus-Dieter Althoff[2]

[1] Institute of Computer Vision and Applied Computer Sciences, IBaI, Leipzig
www.ibai-institut.de
[2] University of Hildesheim
www.uni-hildesheim.de

Abstract. This article deals with several aspects of a marketing-oriented analysis of web log files. It discusses their preprocessing and possible ways to enrich the raw data that can be gained from a web log file in order to facilitate a later use in different analyses. Further, we look at the question which requirements a good web log analysis software needs to meet and offer an overview over current and future analysis practices including their advantages and disadvantages.

1 Introduction

Maintaining an online presence offers numerous advantages to an organization or company. The internet offers a relatively cheap and simple way to reach a high number of people, independent of their location or other circumstances, and present them with information, a company's range of products and maybe even the opportunity to buy them online. Such a web presence is advantageous in many aspects: Information needs to be published only once, it is always and easily available; changes are easy to make and come into effect immediately.

There is however more to an online presence than it seems at first glance: a website does not only offer information to the visitor, it also conveys information about the visitors – and thus the company's potential customers – to the company! It should not be forgotten, that the people visiting a company's web pages are the company's clients and clientele and thus any information about them is valuable.

The log files produced by a web server are a useful source of information for this. However, already with middle-rate traffic the log files of a website grow to a size that can no longer be evaluated manually in an acceptable amount of time. Thus a program is needed that can fulfill several tasks in supporting a company's webmaster or marketing personnel with this [1].

In this paper we are focusing on log file data preparation for later analysis. We describe the issues that need to be addressed when preparing a log file in such a way that it can be semantically understood by humans. As a basis for our study we first briefly describe the common log file format in Section 2. Afterwards we review existing freely or commercially available log file analysis tools in Section 3. In conclusion of that we focus on local analysis tools that allow a user to study their logs onsite. The software requirements for these kind of tools are described in Section 4.

P. Perner (Ed.): ICDM 2006, LNAI 4065, pp. 131 – 145, 2006.

Based on that we develop in Section 5 a system architecture and describe the single components of this architecture exemplarily on samples. Finally, we give an outlook on the analysis of the prepared log file in Section 6. Conclusions about our work are given in Section 7.

2 The Log File

An average web server log consists of one line per executed web server command and is set in the *"(Extended) Combined Log Format"* [2]. It usually looks similar to the example presented in Figure 1.

More data can be included easily, however most ISPs keep more or less to this standard. Additional information could e. g. be the accessed domain or the indication of a proxy.

The single entries are separated by spaces. Multi-word expressions are enclosed in quotes, parentheses or square brackets, other delimiters may occur but are uncommon. An empty entry is usually indicated by a single hyphen.

```
123.45.67.89 - JohnDoe [15/Jan/2005:18:53:37 +0200] "GET /index.html
HTTP/1.1" 200 639 "http://www.google.com/search?q=mydomain+myname"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.12) Gecko/20050919
Firefox/1.0.7"
```

Fig. 1. A log line in Combined Log Format

It includes:

- The **REMOTE HOST'S INTERNET PROTOCOL (IP) ADDRESS:** Each machine connected to the Internet has such an address. It takes the form of four numbers between 0 and 255 separated by dots and identifies each sender or receiver of packets sent across the Internet. In this case the log gives the IP address of the sender, i. e. the address that requested a certain file.
- The **REMOTE LOGIN NAME OF THE USER.** (Usually empty)
- The **AUTH LOGIN.** Set if the page or a file on it is password protected
- The exact **DATE** in a unified format (a.k.a. timestamp). This format can be freely set in the web server's configuration, but most webmasters adhere to the standard presented in the example.
- The **REQUEST.** The exact command that was passed to the web server, stating which file to get and what method to use.
- The **WEB SERVER'S RETURN CODE.** This code indicates whether the request could be carried out successfully. The most common return codes are 200 (OK), 403 (Forbidden) and 404 (Not found).
- The **SIZE OF THE RETURNED FILE** in Bytes
- The **REFERRER** that indicates the website the user came from when requesting this file (not always detectable)
- The **USER AGENT**, usually the browser program or if the visitor it not human a designation of the respective robot or crawler.

Although this standard looks well organized it must be understood, that a web log is a pretty unstructured source of information. The format presented above is merely a recommendation and any webmaster is free to change it according to his or her needs. The given values, their order and formatting can be changed freely and there are only few common standards such as the space as first separator or a hyphen as general place holder for missing values.

Web design and web technologies are ever-changing and these changes will be directly reflected in the web server's log. It is thus important to keep the structure of the program as flexible as possible. The log file has to be considered as a very dynamic source of information in respect to the structure of the data and the content and this has to be taken into account when developing a flexible log file preparation tool and not a special purpose data preparation tool.

3 Related Work

When looking at the current range of web log analysis tools, their supply can be broken down into three different groups: Free web statistics by ISPs, remote analysis services and local software.

When evaluating them we need to consider the difference between a sales oriented website and a site that merely offers information. For a web shop, success can simply be measured in sales. A site that "only" offers information has more complex requirements: A successful page visit does not necessarily end in a purchase or anything else that can be directly measured using the web server logs. For such sites the quality criteria are more fuzzy: How many people visit the site? What do they look at? Where is it linked or referenced? How often are the owners approached on contents within the page and by whom? How quickly do changes or new information get noticed? Also, the line between use and misuse is thin, an informational site wants to be referenced, it does not want its content to be copied or even hot-linked. Thus a thorough analysis of referrers, in this case especially when considering image files or text documents is needed. A simple per page analysis would leave such things unnoticed.

The first group that we looked at, are free web statistics offered by most ISPs for their website hosting customers. These are the most basic analyses. They give a rough overview over the number of requests and some statistics, but lack usability and flexibility. Analyses are presented as-is and the user has no control over what is analyzed and how. Such an analysis can give a quick overview over a page's traffic, it is of no use though for a more thorough analysis or an application in CRM (customer relationship management).

The second group are host based services. These services don't analyze the log files produced by the respective web server. Instead they use a different approach where the webmaster adds to each page that shall be analyzed a remotely hosted element (usually a picture or a java script call). Thus every time the page is requested, the element is requested as well and a log entry is added to the remote service's log. This offers a lot of advantages to the service provider: There is no need to deal with changing log formats, since they only have to parse their own log files, the parsing can be done locally, in real time and on a server rather than a personal computer.

There is but one problem: The company exposes exact and detailed information about it's web presence usage – and thus customers – to another party. Moreover, all analyses can be accessed over the web (usually via a simple username/password authentication) and are thus in the potential danger of being hacked. Many companies may not be willing to take that risk.

This leads us to the third group: Local log file analysis tools. A broad range of analyses is offered by NetLog, made by IBaI Solutions [3]. The program offers access to a whole number of analyses and views accompanied by the according charts. Another possibility is an integrated solution based on the products by Microsoft[4]. They offer a statistical analysis that is based on OLAP and analysis components in their SQL and Commerce Server Products. However, the exemplary statistical analyses we could see showed that they do not process the log file in such a way that more intelligent analyses are possible besides the statistical ones. Other professional solutions are offered by ClickTracks [5], WebTrends [6] and LiveSTATS by DeepMetrix [7]. These programs also incorporate page content modeling (defining the "role" of a site) and customer relationship management (CRM) elements. Their program design mostly aims specifically at web shops and is capable of e. g. recognizing an addition to the shopping basket or a successful purchase. With this information they can distinguish successful from unsuccessful visitor sessions or recognize buying patterns, which, e. g. connected with the entry referrer, allows conclusions as to the effectiveness of a certain ad, or of being referenced on a certain website or portal. Provided with the costs of different ads they can even calculate their ROI (return on investment) based on the actual purchases that resulted from them. An additional feature that can be found in all of them are tied-in CRM tools that can e. g. associate email addresses with visitors using tagged referrers within email campaigns.

These programs offer not only lots of additional eye candy, but also more sophisticated filtering and a large number of criteria that visitors and sessions can be sorted by. As to the actual analysis, all of these programs have a two-fold approach: They offer their analyses either as a client based piece of software, or as a hosted service (as described above). Also, although they do offer the import of raw log files, all of these programs prefer the afore mentioned approach of including a tracking script in each website, that is then either remotely or locally evaluated. This offers numerous advantages, as it again spares having to deal with changing log file formats, funny values, filtering of frames and other nuisances traditional log file analysis brings with it (see further sections). Also, sometimes there is no other choice than to use a remote element for web analyses. For example when a reverse proxy is used that accepts all outside requests and hands them to the respective services. This is a common practice among larger sites in order to deal with load balancing, on-site firewalls and other security measures. In these cases however, the requesting IP is always the IP of the reverse proxy. Here a remote element is one way to get to the actual requesting IP. However, there's also disadvantages to this approach: Firstly it means a lot of initial extra work for the webmaster: The tracking script call has to be inserted into every single page of the web presence. Depending on the number and structure of pages, this can be a wearisome task. In addition an analysis based solely on tracking scripts will only work from the day the scripts were inserted. Months or

even years worth of collected logs would remain unused, a considerable waste of information resources. Also, though evaluating the information gathered by a tracking script is much more comfortable, web server logs are still the most detailed source of information available. Their seeming disadvantage is also their biggest advantage: They list everything! This produces a considerable overhead, but also offers the possibility to retrieve information from old logs that maybe didn't seem interesting (and thus wasn't included in the tracking scripts) then, but is in a later point in time. Also the presented professional programs are mostly tailored for web shops and can only partially fulfill the described needs of information-centered web sites, as mentioned in the beginning of this section.

Finally there is one thing, that none of the presented programs offers: All of them can only group sessions by either pre- or user-defined criteria or patterns. A valuable additional source of information would be to see what new correlations or groups can be found. This needs more intelligent analysis methods such as clustering for discovery of groups of users [8] [9], structural analysis for common path recognition or for finding sequence rules in web log files [10].

4 Software Requirements

Web logs offer a great wealth of information, they are however poorly structured. Thus, software that aims to allow it's user to work in a reasonable way with the information they include needs to fulfill certain requirements:

- It needs to be able to handle the dynamic nature of the structure and of the content (semantic) of the data without doing any programming changes.
- It needs to present the information included in the log file in a clear and concise manner, excluding unnecessary information, presenting the remaining data in a clearly semantically understandable way.
- It also needs to offer several different views on the data, depending on what the interest of the user is. Be it the source of a page's visits, which pages attract most attention or how well the pages are covered by search engines and listing services.
- The software needs to support a quick overview just as well as an in-depth investigation. Therefore, the analysis should work on a single or several data base fields as well as on single or consecutive data base entries.
- Since we are dealing with a temporal medium comprised of web server requests in temporal sequence we also have to determine a user session from our raw data. This session represents one single, continuous visit of a person (or robot).
- It needs to extract all additional information that can be gained from the log data and allow their free combination.
- It should allow detecting and filtering out pre-defined patterns in the log file data such as robot accesses to the website.

Generally speaking, it needs to assist the user in drawing conclusions from the data, either by doing that itself or by providing according hints.

5 System Overview

To fulfill the software requirements described in Section 4 a series of steps needs to be carried out in order to get from the original log file to a usable database [11] (see Fig. 2). These steps are (1) Data Preparation, (2) Data Extension, (3) Data Conversion & Extension (4) Data Parsing, and (5) Session Recognition.

The software should be as flexible as possible. Therefore the structure of our tool is similar to a knowledge-based system comprised of an inference engine and a knowledge base [12]. Everything that is flexible or can change over time is represented by facts in the knowledge base. This knowledge base can be edited by a human or can be obtained by extracting facts from the log file and importing them as a text file into the knowledge base. Another feature of the software is an automated look-up of the DNS information from the DNS server to ensure the identification of the visitor by his hostname or DNS entry. In order to work on the log data with reasonable performance the existing text file is first converted into a local database file. A log file is more or less a raw data file where every single line of text represents a future data set and every separated value goes into another column. Once the initial database is filled, further data processing is performed in order to bring the data in a convenient format.

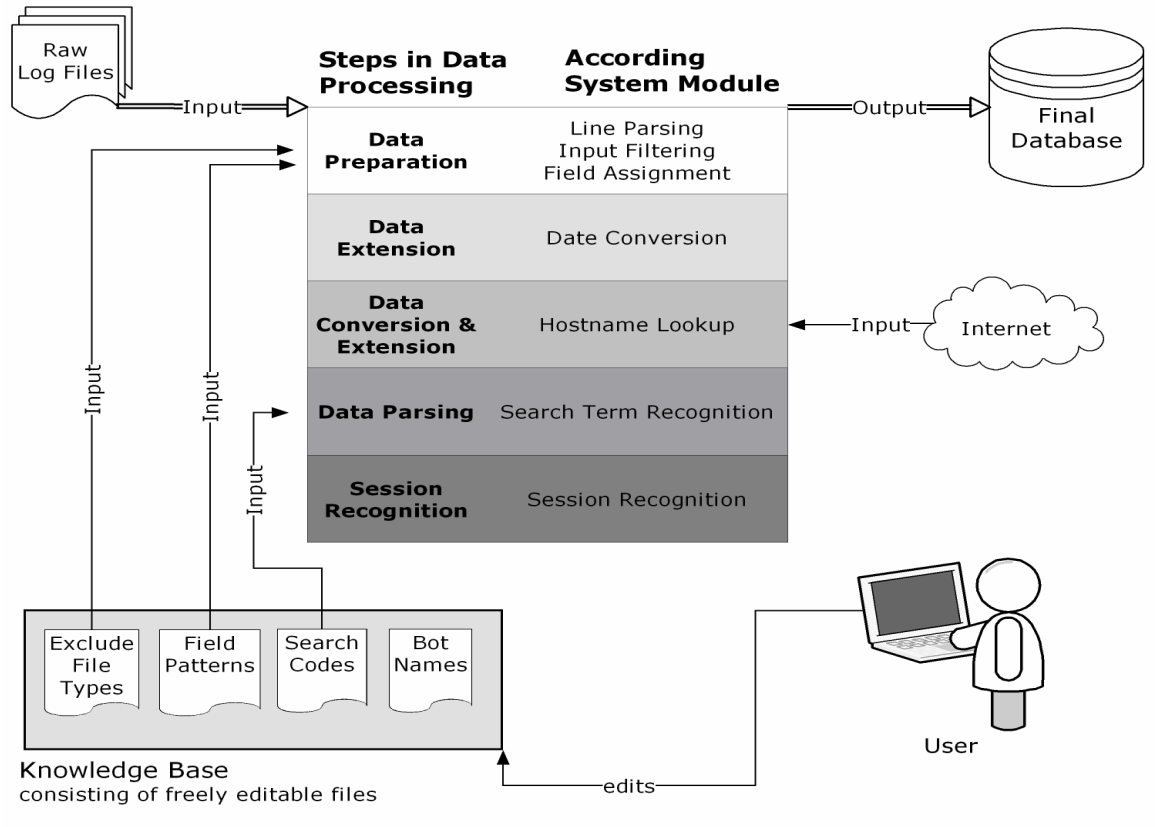


Fig. 2. An Overview over the individual Steps in Data Processing

One of these steps is data parsing where semantically irrelevant data are removed from the data strings in order to get human readable information out of encoded strings, such as the referrer. Date strings are converted to timestamp data types, numeric values (such as IP addresses) are supplemented with symbolic information (in this case the according hostname). This step is referred to as data conversion. Other DNS information leads to data extension since they bring out more information about a visitor such as the company name or the country name. In a last step the data sets are split up into sessions and each session is assigned a unique session ID. After this last step, we have generated a final database that contains all explicit and implicit information that can be gained from the log file. Over this data base we can perform our data analysis.

We would like to point out here that the performance of the software is an important issue that already needs to be considered during the software design phase, otherwise the data preparation step takes too much computation time. Reading values from the database and parsing them or performing a database lookup does not take too much time. However, writing them back takes time. Every new value can only be inserted in the fitting places so with every insertion goes a full database search. Therefore, multithreading techniques should be used and the data base design should be made accordingly. However, since parts of these results depend on each other (e. g. session IDs can only be assigned once the dates have been parsed) this can only be used in some aspects. Thus other sophisticated procedures need to be included. Looking up each individual hostname can cause some delay, especially when an IP cannot be resolved by the “whois” command and it times out. This problem can at least partially be solved by implementing a caching function.

5.1 Creation of the Initial Database

5.1.1 The Database

For reasons of simplicity and portability we are using an embedded database management system, rather than a client server based approach. However, since all database communication is done using simple SQL, interaction with any other database management system is possible too, provided the according driver is included in the program. Storing the data in a SQL database also allows the capable user to not only use the predefined queries in the program, but also freely query the database him-/herself to get whatever information is desired. Doing a direct SQL query, the search can e. g. be reduced to a specific time segment, or limited to datasets matching a certain criterion (e. g. all visitors from a particular country, all accesses of a particular page, or all users who visited more than 5 different pages.). There is literally no limitation to what you can do with the database using SQL. Also data mining methods can work on SQL [13] [14].

5.1.2 Line Parsing and Database Insertion

The creation of the database itself is done by simply parsing line by line of the log file and writing each value into the database. Likewise in any data mining problem we also have to face the problem of missing or unknown data values while parsing the data.

Missing data are data that are not in the log file such as the language of the user agent that is not always indicated.

We consider the unknown data problem as the problem where we cannot parse the data since the parsers has no parsing rules for it. It is clear that this system state cannot remain as it is during the lifetime of system. Therefore our system has the possibility to detect this novel situation and go into a knowledge engineering phase by allowing the user to manually insert the missing parsing rules into the knowledge base.

Thus, the first task that needs to be undertaken is to break each line into its respective elements. The number of these elements is the same in every line, since missing values are replaced by a place holder (usually a hyphen).

In order to distinguish the individual elements during the parsing, we first need to identify the different separators: The default separator is a single space, if an element contains spaces itself, it needs to be enclosed by an additional type of delimiter (see Fig. 3), this can be squared brackets, single or double quotes. Simple parentheses or curly braces are less common but should be considered as well. When doing this, a common mistake is to fall for delimiters within elements, such as within the last element in Fig 3. An element always ends with the delimiter it started with – plus a space.

```
foo bar "a multi word item" baz [there are many different delimiters]
'and yet another item, with "funny" 'stuff'
```

Fig. 3. Different Delimiters

It should also be considered that delimiters that do not break parsing can still cause trouble when feeding the respective string into the database. Especially mixtures or repetitions of quotes and double quotes within a string can interfere with the insertion of that string into the database. Most programming languages offer a quoted string function that can take care of some of these problems. They cannot take care of them all though, and also log data should not be altered too easily since e. g. a referrer url can get useless when exchanging special characters. In these cases a careful choice has to be made between program stability and data integrity.

In order to maintain a database that completely reflects the log file we would now have to stop reading the data, output some kind of error message as to why that log file could not be read and stop the program. However, for the sake of user friendliness and considering that the degree of "data corruption" will most likely remain well below the usually acceptable 5% error rate we simply insert a dummy value for missing values. This ensures that neither session nor path recognition gets broken.

An additional option might be to include some unique key into the replacement string and write both, the key plus the original line into an external file, allowing the user to still look up the full log entry if he/she wishes.

5.1.3 Input Filtering

Already when we are reading in the original log file a first filtering function must be applied in order to reduce the amount of redundant data within the database. For set-up of such a filter we first have to understand the conceptual model of the web site

and the user needs for the analysis. It is a difference whether images are pure decorational elements or the main information like in an on-line image gallery.

Log files can easily have tens of thousands of lines and not every line has the desired informational value. Generally speaking, a line is appended to the web server log for every single command the web server carries out whether it sends an html file, images or a css file, containing styling information.

Not all of these actions are of interest to us though. What we most likely want to see is the user's navigation over the different pages. Styling or decorative elements are – under the aspect of a marketing oriented analysis – of only limited informational value. If a visitor accesses a page that has a background image and buttons underlayed with two different bitmaps, this will result in four lines in the log file: one for the actual html file, one for the background image, one for each bitmap, since these are of course also files that have to be retrieved by the web server. These four log entries will however always appear together, since the images will (in most cases) only be requested when a visitor accesses the html page that references them. The three image requests thus add no informational value and need not be included in the database.

Due to this a first input filter can be applied here that controls, which log lines do get included in the database and which are left out all together. This can for example be done by providing a configuration file that lists file extensions that shall be excluded from the database. Typically this file would include image file extensions such as jpg, gif or png, or files that are concerned with scripts or formatting, such as java script or css. In order to adhere to the afore mentioned flexibility it is important to not hard-code these into the actual source code but export them to a freely editable configuration file. Providing this, new additions can be made anytime (e. g. when a site is switched to css) or entries can be taken out.

5.1.4 Field Name Assignment

Once all desired log entries have been inserted into the database, one last problem remains to be solved: As mentioned above there is a common standard for log file formatting, however, no ISP has to stick to it and it is easy to change the log format within a web server application. Thus we can make no assumptions as to which database column represents which value. There are two approaches to dealing with this problem: One would be to adjust the program to the log file format, either by another configuration file or by presenting the user with one or a collection of sample lines, asking him/her to give each entry the according designation. This approach however depends on the quality of the sample line(s) and the user's knowledge on the matter. A second possibility would be an automated approach: Here the program's settings are not changed, instead it tries to guess the according fields itself e. g. using textual pattern recognition. This would be no problem with entries like the IP or the web server command. The recognition however becomes less trivial when it comes to distinguishing e. g. return code from byte size or classifying entries that are arbitrary, like a user's login. Also such a mechanism would have to be provided with all possible fields plus a description of their pattern and could thus not adapt to new database fields. Thus, we would suggest a user-based approach.

5.1.5 Data Extension and Conversion

Once initial data preparation is finished and an initial database has been created, the database is worked over several times in order to include additional information that

can be gained from the log file information itself. Firstly the dates that are available only as simple strings (see Figure 4) are converted from a string into a date data type (e. g. DateTime or Timestamp) in order to facilitate working with them in the following process.

```
[15/Sep/2005:08:35:37 +0200]
```

Fig. 4. A typical Time String

Once the database has proper date types, queries like "before", "after" or "between" can easily be carried out on these fields without any need for tedious string parsing.

Additionally every IP is looked up using functions provided by the respective programming language. Often the hostname already tells us what institution or company the visitor came from. If the information is insufficient or a hostname is unavailable a full DSN query can be done. An IP's DNS record can provide much useful information on the owner of an IP, such as the country, a contact address and a description of the institution.

This works of course only if the visitor or his/her company or organization have an IP address of their own. Otherwise the DNS record will show the data of the visitor's internet service provider. This can still be helpful though, since it may be assumed that most people have an ISP within their own country.

5.2 Data Parsing

Another source of information are search machine referrers, since they indicate the search terms with which the webpage was found and sometimes also the position in the search machine's page ranking. It may also be of interest which search machine is most frequently used or what internet portal is most popular with the company's clientele. However, the referrer that includes this information will be rather cryptic. Figure 5 shows three different search engine referrers searching for the same key words from the same browser and the same machine, though they vary considerably since every search engine has it's own way of encoding search queries.

```
http://www.google.com/search?q=ibai+institut&sourceid=mozilla-
search&start=0&start=0&ie=utf-8&oe=utf-8&client=firefox-
a&rls=org.mozilla:de-DE:official

http://suche.web.de/search/web/?mc=hp%40suche.suche%40home&su=ibai+in
stitut&webRb=de

http://search.lycos.com/default.asp?loc=searchbox&tab=web&query=ibai+
institut&submit.x=0&submit.y=0&submit=image
```

Fig. 5. Different Search Engine Referrers

The program now needs to be able to e. g. extract the search terms from every of these referrers, and again, there is no guarantee that a search engine will not change it's

encoding or a new one will come up. This problem is solved by again outsourcing the descriptors that indicate the actual search terms (in this case "q=", "su=" and "query=") into an external file, so that new descriptors can be included and existing ones changed. To further illustrate this, we will demonstrate the exact parsing process in detail with the first referrer.

The full version is:

```
http://www.google.com/search?q=ibai+institut&sourceid=mozilla-
search&start=0&start=0&ie=utf-8&oe=utf-8&client=firefox-
a&rls=org.mozilla:de-DE:official
```

We perform a search over it with the search word descriptors we know and cut it from there, which leaves us with:

```
q=ibai+institut&sourceid=mozilla-search&start=0&start=0&ie=utf-
8&oe=utf-8&client=firefox-a&rls=org.mozilla:de-DE:official
```

We also know that the ampersand ("&") is in pretty much every encoding used to append the different elements of a query. Thus we also make a cut before the first ampersand:

```
q=ibai+institute
```

With this cut-out we have almost reached our target. Now all we have to do is to remove the search word descriptor and use a url decoding function on the remaining text. Url decoding [15] is used to encode special characters within urls. (Unfortunately not every site uses them as they should, otherwise a lot of the above mentioned parsing problems wouldn't exist. However we may assume that at least all reasonable search machines use url encoding for their search expressions.) A few examples for url encoding can be seen in Figure 6.

Search term encoded	**Search term decoded**
ibai+institute	ibai institut
%22ibai+institut%22	"ibai institut"
ibai%2Binstitut	ibai+institut
%27ibai+institut%27	'ibai institut'

Fig. 6. A few Examples on URL Encoding

Once the url decoding is successfully finished, the original search terms are extracted from the search machine's referrer and can be presented to the user, statistically analyzed etc.

Of course extracting the search terms isn't the only information that can be gained from a search engine's referrer. Providing some more previous knowledge on the structure of a certain search machine's queries we can also extract additional information such as the browser language or a page's position within the search engine's page ranking system. This information shows another possibility of data

extension but is however not provided by every search machine and should hence not be generally expected.

5.3 Session Recognition

Once all above mentioned steps are carried out, every data set is assigned a session ID (SID). A session is understood as a single visitor's contiguous path over the different pages. Thus we define a session as a set of log entries where every entry has the same origin IP address, and the time interval between two consecutive entries is never above twenty minutes.

6 Views and Analyses

Up until now we have realized two different kind of web log data analyses: statistical analyses and views.

The statistical analyses don't present the log data themselves but meta information about them, such as the number of entries, the number of pages, the number of unique or recurring visitors, average length of stay or the amount of generated traffic. Another common task are rankings: The most popular referrer, hostname, search terms or, in a multi domain log, the most popular domain. Any database field can be ranked like this, though it of course doesn't make sense for all of them. This data representation is most suited to give a quick overview on the general numbers and popularity of a specific website or an entire web presence.

Views present the actual log content but in a more tailored fashion. This can mean rearranging individual columns, filtering or grouping the data. An example for filtering would be excluding automated page hits from entities like crawlers, bots or other information harvesting scripts. The more up-to-date a search engine wants to be, the more crawlers it needs to send out. This can cause a considerable impact on your website's log. You also might want to exclude your own page accesses. Many companies maintain the practice to set their own site as every web browser's starting page. That is okay as long as it points to a local copy or the intranet. Setting it to the company's internet address tends to tamper website statistics, hit counters and other statistical analyses. An example for grouping data would be a session view. Here, instead or presenting every single dataset, the log entries are combined to sessions, which can then again be compared, evaluated or sorted by all desired criteria.

7 Analysis Obstacles

A data analysis like the one presented in this paper offers high informational value to the owner of a web presence. It is not trouble-free though. During the first testing phase we noticed several factors that can tamper or complicate a proper site analysis.

A first obstacle is the use of frames. This means that a site does not consist of only one html document, as it usually is, but it consist of several frames, that each contains an html document. Typically there may be one frame containing the menu, another one for the title or footer and a third one that holds the actual content. Thus, when

entering a site the server has to send three html documents for the user to see a (seemingly) single page.

Frames should generally no longer be employed for a number of reasons: First of all they tend to cause confusion to the user himself because they appear as one single site, though they are not. This can cause problems with a number of things like printing (many browsers put one frame per page.), bookmarking (the bookmarked page will be a different one depending where the focus is at the time of the bookmark's creation) or interaction with other software like screen readers or hardware like a Braille terminal. Also navigation can be made difficult because the functioning of the "forward" and "back" buttons is affected by frames as well. Badly implemented "nesting" frames can even render a page completely useless. These navigation problems also affect crawlers that try to index the site for a search machine. Also, there are also still browsers around, that don't support frames. All this has led to a dislike of frames among the wider internet "population". They are today regarded bad style and "old hat".

Additionally to all this they also complicate a proper path analysis since there is no way to tell whether a given frame offers content, navigational elements or pure decoration, say, a fancy title. So instead of a clear path like this:

```
index -> products -> product1 -> contact -> email_form
```

you might end up with a path like this one:

```
menu -> top -> index -> menu -> top -> products -> product_footer ->
menu -> top -> product1 -> product_footer -> menu -> top -> contact -
> menu -> top -> email_form
```

Thus, used on a page structure that employs frames the software would have to be adapted individually to each frame that is used and does not contain path-relevant content.

Another problem, not for the path analysis but for it's use by a human, are content management systems (CMS) and their automatically generated page names, that leave you with a path like

```
1/1 -> 16/7 -> 16/12 -> 8/1 -> 2/4
```

Here the generated path is valid but does not allow any conclusion as to the site's actual content. There are several ways for solving this, such as configuring the content management system to use descriptive page names or mapping the individual page names to a more meaningful "translation" within the program.

A last factor that should be considered are people who deliberately hide their identity when online, be it for privacy reasons or on a certain purpose. Measures like e. g. using an anonymizing HTTP proxy or forging a browser's user-agent signature can effectively hide a user's identity from web server logs. Other measures like setting cookies or using java scripts to track a visitor can also be evaded easily. It should also be kept in mind, that tracking techniques, when used too aggressively, can also have a deterring effect since nobody likes to by "spied" upon. It needs to be decided individually which tracking techniques are adequate and what conclusions can be drawn from the number of visitors evading them.

8 Conclusion and Outlook

Our objective is, to find a way to present the vast but incomprehensible data that are provided by a web server's log file in a clear and concise way, but without reducing it's informational value or making any kind of data inaccessible. In order to achieve this, the interface was designed to be simple and flexible because a one-size-fits-all approach is not feasible in this application. There can be only little prediction as to a user's intention. He or she might just want to get a quick overview or – in the opposite – trace a specific user's behavior, thus a good log analysis software needs to meet all these requirements.

We have described our work on web server log data preparation. Our intension was to develop a flexible software tool that meets the requirements posed by the dynamic nature in structure and content of a web log file. Therefore we developed an architecture for the system that is similar to a knowledge-based system comprised of a inference engine and a knowledge base. The knowledge base contains information about the parsing criteria and the set up of the filters. These information can be easily changed or up-dated by hand or semi-automatically. We have also point out the special software development requirements to ensure acceptable run-time performance. The missing and unknown data problem was described and method to handle that for log file data. Finally, we presented our first analysis methods. Further work will be done to develop more intelligent analysis methods such as user profiling, path analysis and association mining.

References

1. P. Perner and G. Fiss, Intelligent E-marketing with Web Mining, Personalization, and User-Adpated Interfaces, In: P. Perner (Ed.) Advances in Data Mining, Applications in E-Commerce, Medicine, and Knowledge Management, lnai 2394, Springer Verlag 2003, p. 37 – 52.
2. Apache HTTP Server Documentation Project: Combined Log Format. http://httpd. apache.org/docs/1.3/logs.html#combined (accessed Jan 23rd 2006)
3. P. Perner, Einrichtung zur automatischen Ermittlung der Nutzung von Web-Präsentationen und/oder On-line Verkaufsmodellen, Patent D 198 01 400.7-08
4. Microsoft SQL Server Analysis Services http://www.microsoft.com/sql/technologies/ analysis/default.mspx
5. Clicktracks Software http://www.clicktracks.com/
6. Webtrends Software http://www.webtrends.com
7. Deepmetrix Software http://www.deepmetrix.com
8. Scherbina and S. Kuznetsov, Clustering of Web Session Using Levenshtein Metric, In: P. Perner (Ed.), Advance in Data Mining, Applications in Image Minnig, Medicine and Biotechnology, Management, Environmental Control, and Telecommunications, lnai 3275, Springer Verlag 2004, p. 127-133.
9. M. Halvey, M.T. Keane, and B. Smyth, Birds of a Feather Surf Together: Using Clustering Methods to Improve Navigation Prediction from Internet Log Files, In: P. Perner and A. Imiya (Eds.) Machine Learning and Data Mining in Pattern Recognition, lnai 3587, Springer Verlag 2005, p. 174-183

10. E. Blanc and P. Giudici, Sequence Rules for Web Clickstream Analysis, In: P. Perner (Ed.) Advances in Data Mining, Applications in E-Commerce, Medicine, and Knowledge Management, Springer Verlag, lnai 2394 , 2002 p. 1-14
11. Ahlemeyer-Stubbe, Analyseorientierte Informationssysteme = Data Warehouse, In: P. Perner (Ed.) Data Mining, Data Warehouse, and Knowledge Management, Proc. Industrial Conference on Data Mining ICDM, IBaI-Report 2001, ISSN 1431–2360, p. 30-39
12. J. Rech and K.-D. Althoff, Artificial Intelligence and Software Engineering - Status and Future Trends, Special Issue on Artificial Intelligence and Software Engineering, Zeitschrift Künstliche Intelligenz (3)2004, p. 5-11
13. R. Meo, P. Luca Lanzi, M. Klemettinen (Eds.), Database Support for Data Mining Applications: Discovering Knowledge with Inductive Queries, Springer Verlag 2004, lnai 2682
14. R. Meersman, K. Aberer, Th. Dillon (Eds.), Semantic Issues in e-Commerce Systems, Vol. 111, 2003
15. W3 Schools: HTML URL-encoding Reference. http://www.w3schools.com/tags/ref_urlencode.asp (accessed Jan 23rd 2006)

Bibliography

[1] Apache Software Foundation. Apache struts. Available from: http://struts.apache.org/2.1.6/index.html.

[2] Apache Software Foundation. iBATIS. Available from: http://ibatis.apache.org/.

[3] Apache Software Foundation. log4j. Available from: http://logging.apache.org/log4j/.

[4] Apache Software Foundation. Tomcat. Available from: http://tomcat.apache.org/.

[5] Apache Software Foundation. Velocity. Available from: http://velocity.apache.org/.

[6] Codehaus. Xstream. Available from: http://www.xstream.codehaus.org.

[7] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Verion Control with Subversion*. O'Reilly, 2004.

[8] Conversion Rate Experts. Google website optimizer 101 – a quick-start guide to conversion rate optimization. Available from: http://www.conversion-rate-experts.com/articles/101-google-website-optimizer-tips/.

[9] The Internet Engineering Task Force. Hypertext transfer protocol – http/1.1 specification memo. Available from: http://www.ietf.org/rfc/rfc2616.txt.

[10] Apache Software Foundation. Commons logging. Available from: http://commons.apache.org/logging/.

[11] Google. Google Analytics. Available from: http://www.google.com/analytics/.

[12] Goetz Graefe. Query evaluation techniques for large databases. *ACMComputing Surveys*, 1993.

[13] Samudra Gupta. *Pro Apache Log4j*. Apress, 2 edition, 2005.

[14] Steve Holzner. *Ant, The Definitive Guide*. O'Reilly, 2 edition, 2005.

[15] InfoSoft Global. FusionCharts. Available from: http://www.fusioncharts.com/.

[16] ISO 3166 Maintenance agency (ISO 3166/MA). Iso 3166-1 (2-letter country codes). Available from: http://www.iso.org/iso/english_country_names_and_code_elements.

[17] json.org. Json - javascript object notation. Available from: http://www.json.org/.

[18] JUnit. JUnit. Available from: http://junit.sourceforge.net/.

[19] Brian Keating. Challenges Involved in Multimaster Replication. Available from: http://www.dbspecialists.com/files/presentations/mm_replication.html.

[20] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2006.

[21] Timothy C. Lethbridge and Robert Laganière. *Object-Oriented Software Engineering.* McGraw Hill, 2 edition, 2005.

[22] Shibo Li and Patrali Chatterjee. Shopping cart abandonment at retail websites - a multi-stage model of online shopping behavior. IEEE International Conference on Data Mining, Springer, 2006.

[23] Vincent Massol. *JUnit in Action.* Manning, 2004.

[24] MaxMind. GeoLite Country. Available from: `http://www.maxmind.com/app/geolitecountry`.

[25] Mountainminds. EclEmma. Available from: `http://www.eclemma.org/`.

[26] OASIS. SOA. Available from: `www.oasis-open.org/committees/soa-rm/`.

[27] PostgreSQL Global Development Group. PostgreSQL. Available from: `http://www.postgresql.org/`.

[28] PostgreSQL Global Development Group. PostgreSQL partitioning. Available from: `http://www.postgresql.org/docs/current/static/ddl-partitioning.html`.

[29] Meike Reichle, Petra Perner, and Klaus-Dieter Althoff. Data preparation of web log files for marketing aspects analyses.

[30] Sun Microsystems. Java property file. Available from: `http://java.sun.com/javase/6/docs/api/java/util/PropertyResourceBundle.html`.

[31] Sun Microsystems. Java SE Desktop Technologies - Java Beans. Available from: `http://java.sun.com/javase/technologies/desktop/javabeans/index.jsp`.

[32] The Eclipse Foundation. Eclipse. Available from: `http://www.eclipse.org`.

[33] The Prototype Core Team. Prototype javascript framework. Available from: `http://www.prototypejs.org/learn/json`.

[34] W3C. Soap version 1.2 primer (second edition). Available from: `http://www.w3.org/TR/2007/REC-soap12-part0-20070427/`.

[35] W3Schools. Ajax tutorial. Available from: `http://www.w3schools.com/Ajax/`.

[36] W3Schools. Introduction to XML. Available from: `http://www.w3schools.com/xml/xml_whatis.asp`.

[37] W3Schools. Javascript introduction. Available from: `http://www.w3schools.com/js`.

[38] Wikipedia. Comma separated values. Available from: `http://en.wikipedia.org/wiki/Comma-separated_values`.