



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JARKKO MÄKELÄ
USING RULE ENGINE TO VALIDATE INSURANCE DATA
Master of Science Thesis

Examiner: Professor Hannu
Jaakkola
Examiner and topic approved in
the Faculty of Computing and
Electrical engineering meeting on
September 4th 2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO PORI

Tietotekniikan koulutusohjelma

MÄKELÄ, JARKKO: Sääntömootorin käyttö vakuutustietojen tarkastamisessa

Diplomityö, 50 sivua

Marraskuu 2013

Pääaine: Ohjelmistekniikka

Tarkastaja: Professori Hannu Jaakkola

Avainsanat: Sääntö, sääntömootori, vakuutustietojen oikeellisuuden tarkastaminen

Vakuutusyhtiöillä on edelleen käytössä vanhoja keskustietokonejärjestelmiä ja niiden ikääntyessä niiden ylläpito vaikeutuu ja ylläpitoon liittyvät kustannukset kasvavat. Tästä syystä vakuutusyhtiöt haluavat siirtyä käyttämään uudenaikaisia tietokonejärjestelmiä. Vanhojen vakuutus sopimusten siirtäminen vanhoista mainframe järjestelmistä uusiin nykyaikaisiin tietojärjestelmiin on kuitenkin pitkä ja monimutkainen prosessi, johon liittyy paljon huomioitavia asioita.

Varsinaisessa sopimuskantojen siirtämisessä ensin suoritetaan sopimusten poiminta vanhoista järjestelmistä muotoon jonka uusi järjestelmä vaatii. Tämän jälkeen suoritetaan tarkistus, että sopimukset jotka ovat poimittu vanhasta järjestelmästä ovat oikeellisia ja uuden järjestelmän vaatimassa muodossa. Lopuksi tiedot voidaan siirtää uuteen järjestelmään. Sopimusten poiminnan ja uuteen järjestelmään siirtämisen välillä suoritettava tiedon tarkastuksen avulla voidaan varmistaa, että sopimukset pysyvät muuttumattomina ja että sopimusten tieto on oikeellista. Lisäksi pystytään varmistamaan että sopimukset jatkavat elämistä oikein ja ilman virheitä uudessa järjestelmässä.

Edellä kuvailtu prosessi vaatii tietysti myös suuren määrittelytyön ennen kuin vanhasta järjestelmästä voidaan alkaa poimia sopimuksia uuden järjestelmän vaatimaan muotoon. Lisäksi siirtoprosessin jälkeen tarkkaillaan, että sopimukset alkavat elää oikein uudessa järjestelmässä.

Tässä työssä kerrotaan yleisesti vakuutus sopimusten siirtämisprosessista vanhoista järjestelmistä uusiin järjestelmiin. Työn pääpaino on vakuutus sopimusten tietojen oikeellisuuden tarkastamisessa poiminnan ja uuteen järjestelmään siirtämisen välillä. Sopimustietojen tarkastus suoritetaan sääntömootorin ja suuren sääntömäärän avulla. Sääntömootorista ja säännöistä kerrotaan tässä työssä kattavasti.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY PORI
Master's Degree Program in Information Technology
MÄKELÄ, JARKKO: Using rule engine to validate insurance data
Master of Science Thesis, 50 pages
November 2013
Major: Software Engineering
Examiner: Professor Hannu Jaakkola
Keywords: Rule, rule engine, insurance data validation

Insurance companies still have old mainframe computer systems in use. When these old mainframe systems age their maintenance gets more difficult and more expensive. That is why Insurance companies want to migrate to modern computer systems. Conversion of insurance policies from old mainframe systems to new computer systems is a complicated and long process which contains many things to consider.

Actual conversion of insurance policies starts from data export from the legacy system to a form that the new system requires. After data export, the policies are validated so that the information in the policies is correct and in the form that the new system requires. Also, this way the life cycle of the policies in the new system is confirmed. After validation, policies can be converted to the new system.

This process also requires lots of specifications before the export of policies for the new system can begin. Also, after the conversion phase, the policies need to be monitored for verifying that they start to live correctly in the new system.

In this thesis I shall explain how the conversion process works. Main point of this work is to show how the validation of insurance policies is made. The validation of insurance policies presented in this work is rule-based. The validation uses a rule engine and a large amount of rules. The rule engine and the business rules are introduced in this work.

DEFINITIONS

API	Application Programming Interface
BPM	Business Process Management
BPMS	Business Process Management System
BRL	Business Rule Language
BRMS	Business Rule Management System
CEP	Complex Event Processing
CSV	Comma Separated Values
DRL	Drools Rule Language
DSL	Domain Specific Language
HRS	Hybrid Reasoning System
KRR	Knowledge Reasoning and Representation
PLP	Profit Life&Pension
PRS	Production Rule System
SQL	Structured Query Language

PREFACE

This study was carried out as a master's thesis for the Faculty of Computing and Electrical Engineering at Tampere University of Technology in Pori. This study took place at Profit Software Ltd. in Pori, Finland.

I want to thank following people: Professor Hannu Jaakkola for all the guidance, Teppo Eeva for all the comments and support during the creation of this thesis, Olli Rauhala for guiding me to the world of rules, all the people at Profit software who have helped me, Juha-Pekka Marttila for helping with the English and my wife Minna for all the support during my studies.

Pori, November 2013

Jarkko Mäkelä

TABLE OF CONTENTS

1	Introduction.....	1
1.1	Scope of the Thesis.....	1
1.2	Profit Software.....	3
2	Conversion process.....	4
2.1	Conversion of insurance policies.....	4
2.2	The conversion engine and the conversion process.....	5
2.2.1	Data export from the legacy system.....	8
2.2.2	Validation of the conversion data.....	8
2.2.3	Data conversion to the new system.....	9
2.3	External demands concerning the conversion of insurance policies.....	10
2.4	Different ways to validate insurance data.....	11
2.4.1	Procedural validation.....	11
2.4.2	SQL based validation.....	11
2.4.3	Rule based validation	12
3	Business rule concepts.....	14
3.1	Rule engine.....	14
3.2	Rete algorithm.....	15
3.3	Example of how the Rete algorithm functions in the pre-validation tool. .	16
3.4	Rule-language.....	19
3.5	BRMS.....	19
3.6	Testing of the rules.....	21
4	Modeling validation rulebook.....	24
4.1	Structured Business Vocabulary (Fact model).....	24
4.2	Rulebook.....	26
4.2.1	Conversion engine interface rules.....	26
4.2.2	Business rules.....	29
4.2.3	Rules based on product parametrization	30
4.3	Business process model.....	30
5	Conversion data pre-validation tool.....	32
5.1	Using rules with java program.....	32
5.2	Managing rules and process with Drools Guvnor.....	33
5.3	Collecting data from the CSV files using the rules.....	34
5.4	Validating data using the rules.....	35
5.5	Reading and preserving facts in memory.....	39
5.6	Usage and error reporting.....	42
5.7	Performance of the pre-validation tool.....	43
6	Conclusions.....	47
	References.....	49

1 INTRODUCTION

This is a Master of Science thesis about using rule engine to validate insurance data. Especially utilizing the technology to validate data when existing insurance policy base in migrated from legacy system to new one. Profit Software has ordered this thesis about comparing different ways to validate insurance data and to implement a validation tool with the best technology available.

I shall begin by explaining the scope of this thesis. After that Profit Software Ltd. is introduced and the company's main business and main articles are presented. The reason why insurance policies are converted from legacy systems to modern computer system is explained. The purpose of the pre-validation tool is explained and why it is needed in the conversion of insurance policies. The whole conversion process of insurance policies from collection of the data from a legacy system to the conversion of the data into the new system is also presented.

1.1 Scope of the Thesis

When a conversion of insurance policy base is executed it may involve several thousands of insurance policies with a lot of information in each policy. This makes the total amount of data massive which gives performance requirements to the validation process. The scope of this Thesis is to study how well rule based validation works when it is used to validate insurance data in a conversion process and how rule based validation performs with massive amounts of data. Conversions of policy bases concentrate on pension insurance policies and savings insurance policies. These two types of policies are more complex than, for example, a normal car insurance because pension and savings insurances has investment targets where the money of the policy is invested. This creates a correlation between insurance policies and investment targets which brings more challenges to the conversion process.

Different ways to validate the insurance data are explained and positive and negative aspects of those ways are considered.

Also, I shall explain in this thesis what the rules are and what the operating principle of the rules is. I shall consider what the requirements are when using rules and rule engine in a software project. The specification of the validation rules is shown with the use of a structured business vocabulary and with the help of the rule book.

Functionality of the pre-validation tool is explained and how the whole validation process is made with it. The validation performance of the pre-validation tool is shown by validating different numbers of insurance policies and by comparing the validation times and the number of facts validated. The tests are done with parts of real conversion data. Next I will present an abstract of each chapter in this thesis.

1 INTRODUCTION

Chapter 1 - Introduction

In the first chapter I will tell the scope of this thesis and introduce the company that has hired me to do this masters thesis about the rule-based validation of insurance data.

Chapter 2 - Conversion process

In the second chapter I will explain why conversions of insurance policies are done and what kind of process the conversion process is from beginning to end. In the end of the second chapter I will compare three different ways to validate insurance data and tell the positive and negative aspects of each different technique.

Chapter 3 - Business rule concepts

The third chapter concentrates on theory concerning business rules and the rule engine. The algorithm behind the Drools rule engine is presented with an example. The use of strict rule language is reasoned and the advantages of the use of rule language is told. The Drools business rule management system is presented and it is explained how it helps in the use of business rules. The last part of chapter three concentrates on the testing of rules.

Chapter 4 - Modeling validation rulebook

The fourth chapter explains how the business rules are built from the business. Concepts like, structured business vocabulary, rulebook and business process model are explained with the help of real examples that were used to create the rules for the pre-validation tool.

Chapter 5 - Conversion data pre-validation tool

The fifth chapter concentrates on the pre-validation tool itself. I will show how the rule engine is implemented to a Java program, how data is collected by using rules to do that, how the validation of the data is done and how the error reporting is made. In the last part of this chapter, performance of the pre-validation tool is tested with different amounts of data and with different sets of validation rules.

Chapter 6 - Conclusions

The last chapter contains the conclusions of this thesis. Effectiveness of the use of the business rules is concerned with several aspects including the performance of business rules when both the amount of the data and the number of the rules are massive. Also, the time used to learn the use of business rules and to create the pre-validation tool is measured.

1 INTRODUCTION

1.2 Profit Software

Profit Software is a software company that has three offices in two different countries. The main office is in Espoo, Finland. One branch office is in Pori, Finland and one side office is in Tallinn Estonia. Profit Software employs approximately 90 persons.

Profit Software's main products are Profit Life & Pension and Profit Property & Casualty. Profit Life & Pension is a web-based solution for pension and life insurance companies. The solution covers all sales, care and claims processes needed for managing life, pension and personal risk products.

Profit Property & Casualty is a modern platform including a suite of software modules for the management of private, commercial, personal and vehicle insurance lines. Profit Property & Casualty offers parametrized services for managing insurance products, modules, business rules and work flows.

Since the foundation in 1992 Profit Software served more than 40 clients in nine countries. Main customers at the moment are Fennia, If, Henkivakuutusosakeyhtiö Duo, Mandatum Life, OP-Pohjola, Aktia and Pohjantähti. (ProfitSoftware 2013 [1])

2 CONVERSION PROCESS

There has always been a need for a cover for the consequences of sickness, fire and all kind of damages. Before modern society, family has given cover for individuals in it. When society developed and the cover of family weakened because of people having to move away from their families, a need for bought insurances started. The oldest known insurance is a transport insurance from the ancient Babylonia around 2000-3000 years BC. Soon after the computer was invented, it changed the insurance world as well. Handling and storing of insurance policies changed from paper to the computers. Evolving computer technology created a demand for conversion of insurance policy bases from legacy systems to new computer systems. (Pentikäinen and Rantala 1995, p. 17-18)

2.1 Conversion of insurance policies

Insurance companies still have old mainframe computer systems in use. When those old mainframe systems age, the maintenance of those computer systems becomes more difficult and more expensive. That is why insurance companies are abandoning old mainframe computer systems and are transferring insurance policies to modern computer systems. This requires conversion of insurance policy bases from legacy computer systems to new computer systems. It is impossible to just copy the policies from the legacy systems to new systems because those computer systems are completely different. Another reason, for why the insurance companies want to do conversions of insurance policy bases from one system to another, is corporate acquisitions. For example, when an insurance company buys another insurance company and its policy base, this policy base is physically located in computer systems that the bought company had and the buyer wants to run down the legacy system. Also, if that bought company has had a third party service for storing the policy base, then it could be expensive to keep one extra policy base with the third party when the policies of that policy base could be stored in the company's own computer systems.

The insurance policies and all the information in them must be correct when conversion of insurance policies to a new computer system begins. The policies must stay unchanged through the conversion process unless some data in the policy is wrong even before the conversion process begins. In case an insurance policy has undetectable errors in the legacy system, those errors must be found and repaired in the conversion process. In that way, correct living of the insurance policies is confirmed in the new computer system. Because of these requirements, strict validation is needed in the conversion process.

2 CONVERSION PROCESS

All data must be validated after the insurance policies are exported from the legacy system and all errors must be fixed before converting the policies to the new system. This requires a special program that validates insurance policies. This work concentrates on a rule-based validation program named the pre-validation tool. The purpose of this validation tool is to find errors from the insurance policies and to report those errors to the user. The pre-validation tool does not fix the findings but, based on the validation report, the problematic policies can be sorted out and manually resolved.

Errors in insurance policies may have been there for a while and may have caused many kinds of damage to that policy. That is why the errors that are found must be reviewed by a professional insurance person. The number of insurance policies and the amount of data that is associated to each policy sets the performance requirements for the validation tool. Conversion process must be made in a certain time window because the value of the investment targets, where the money of the pension and the savings insurances are invested to, changes every day. A policy can only live in one system at a time and so the time when a policy is transferred from a system to another is limited.

The pre-validation tool must execute a huge number of validations for large amounts of policies in a reasonable amount of time. Rule-based validation enables an easy way to manage such validations for a huge policy base.

Several validations are complex and are related deeply to the insurance world. That is why most of the validations are defined by insurance business specialists. Typically business-based validations are not easy to translate from business needs to a technical form. Rule-based validation gives a good way for business persons without a coding background to write validation rules. Also, this creates a common language for business analysts and software developers.

2.2 The conversion engine and the conversion process

The conversion engine is Profit Software's proprietary software to convert insurance policies from CSV file format to PLP (Profit Life & Pension). The conversion engine imports data into PLP by reading CSV files. The general idea is that the CSV files provide an interface to import information into PLP and the data must be exported from existing systems into the CSV file format. As such the CSV files function as an intermediary data format used in communication between the systems.

The source data for the conversion will be in comma separated CSV files. The basic principle is that the conversion interface is common in any conversion cases, but the specific values etc. differ somewhat.

In order to preserve the business secrets of Profit Software this does not represent the real database of the product family, but an intermediate data format. Conversion will be done using method agreement-by-agreement (not file-by-file).

2 CONVERSION PROCESS

The CSV file format contains values separated by comma (.). The lines in a CSV file contain data entries where each of the values is separated by a comma. Double-quotes are used to enclose string and date fields.

The conversion engine will form the backbone of any actual client conversion project. The projects will usually have additional requirements of their own for the conversion process on top of the generic requirements. For example, they might employ a further customized model of the core PLP data model. The converted data must be filtered according to some client-specific criteria or the client organization might require additional reports to be generated. Because of these differing requests the conversion engine needs to be designed as flexible and extendable as possible to make it possible for these project-specific requirements to be implemented as easily and straightforwardly as possible as extensions to the engine.

The conversion engine will only convert the actual business data associated to a particular policy (i.e. policies, clients etc.). It will not convert any general data (like product parameters and investment targets) – as these are considered to be internal, implementation-specific data of PLP and thus outside the scope of this conversion. The conversion will also result in some batches being scheduled in PLP to be executed on the newly converted data to recalculate saving values and other transient data. The conversion of the insurance policies does not happen directly from the CSV files. The data from the CSV files is first imported to an intermediate database from where the conversion engine reads the data.

2 CONVERSION PROCESS

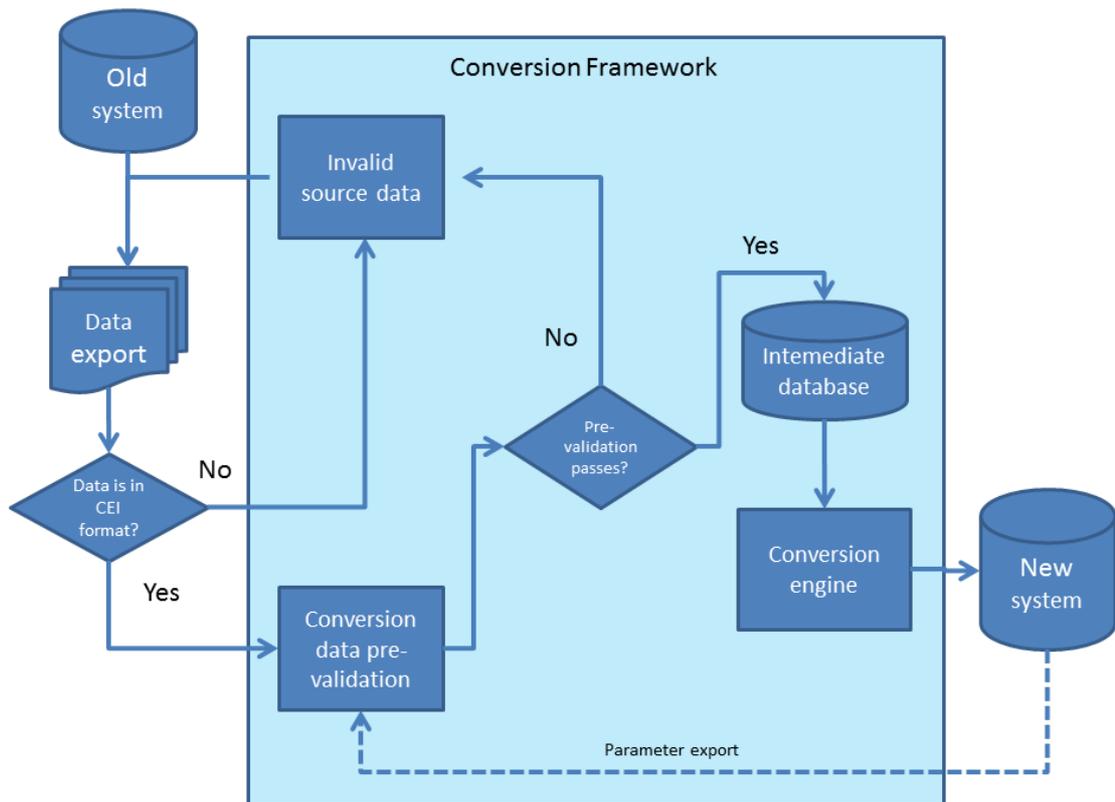


Figure 2.1: Conversion process model.

In figure 2.1 the process model of conversion is shown. The legacy system is a client's old insurance system where the policy base is running and from where the client wants to transfer them. After policies are exported to CSV form the data is validated with the pre-validation tool. Parameter export in figure 2.1 means that the current parameters of each insurance product are read from the new system via API that is implemented for the reading of the parameters. Product parameters are value limits for different kinds of properties that the insurance products have. For example savings insurance can have different kinds of death cover percentage limits, depending on which kind of a savings insurance product is in question.

If validation passes the data is imported to the intermediate database. From there the conversion data is converted to the new system with the conversion engine. If the validation doesn't pass, then the source data is invalid and the initial data or the data export mechanism must be corrected. After the invalid policies have been fixed, the data can be converted to the CSV form again. This phase continues for as long as the source data is valid and it is certain that the data in the policies is correct. Validation of the conversion data can be made weeks or even months before conversion of the policy base. Data can be collected from the legacy system many times before the conversion and it can be validated as many times as wanted. When making many iterations like this it can be certain that the data is valid when the conversion of the policy base begins.

2 CONVERSION PROCESS

2.2.1 Data export from the legacy system

The data must be exported from the legacy system and into a CSV format before its validation and conversion to the new system can begin. The data export can also be done by a third party. The third party is not likely have as much knowledge about the new system as the company that is undergoing the data conversion and is implementing the new system. That is why it is important to validate the exported data as quickly as possible and send a possible error report to the third party as well as to the customer so they can make any required changes to the data extraction process and export the data again.

When the data is converted to the new system from CSV form, the legacy system can be any kind of computer system. The important thing is that which information pieces from the legacy system are connected to the columns of the CSV file. This makes the specification process of the conversion challenging and it needs a lot of accuracy and analyzing. Some data must be generated into the CSV files based on the knowledge drawn from several values in the legacy system, when some data is required in the conversion process, but is not directly available in the legacy system. Some columns in the CSV files are set as default values when there is no appropriate data. If an end date is not defined the date 31.12.9999 can be used which means that the end date is open-ended.

2.2.2 Validation of the conversion data

After the insurance data has been converted into CSV files and it is in a form that supports the conversion process into the new system, it is followed by the validation phase. The validation is done in this phase because the data is in a usable format for Profit Software, the data is in a stable form and there is time to do the validation. When the data is in a format which is defined in the conversion engine interface, the pre-validation tool can be used in any conversion made by this conversion engine. If validation of the insurance data would be done before the data is converted to the CSV format used by the conversion engine, there would be a need to make a new pre-validation tool for every different conversion project.

Validation can also be done while converting the policies to the new system. Some validations are done during the conversion but such extensive validations as the pre-validation tool makes, would take too much time if they would be made during the conversion. In a small conversion process, for example, under one thousand insurance policies, validation can be done while converting policies to the new system, but if there are thousands of policies that must be converted, the validation might take too much time and conversion may not be possible in the given time window. Also, if validation is done at the same time when converting the policies to the new system and there is an

2 CONVERSION PROCESS

invalid policy in conversion, there will be not enough time to fix that policy and that invalid policy would have to be converted again after it has been fixed.

The pre-validation tool produces a log file from the errors it finds in the data. If there is a third party which is responsible for collecting the data from the legacy system and converting it into CSV format, this log file can be sent to the third party and from it they can immediately see where the errors are and make the fixes to the collection of values where the errors occurred. The pre-validation tool is a standalone program and its installation package contains the manuals and everything else that the pre-validation tool needs to work. This way the pre-validation tool can also be given directly to the third party to use, if desired.

The conversion data pre-validation tool is introduced in more detail in chapter 5. That chapter explains how the pre-validation tool is made, how rules are used by the program and how the program reports errors that it finds. In chapter 3, I shall explain how the validation rules are made and what information is needed before they can be made.

2.2.3 Data conversion to the new system

After the data is validated and proven to be correct, it is certain that the insurance policies are valid and intact. After this verification, the data is converted from CSV files to the new system (PLP) by using the conversion engine.

The whole conversion to the new system must be done in a certain time window to ensure that it does not affect the functionality of the new system because the new system is in daily use by insurance companies. This requirement affects the amount of insurance policies in conversion. If there are hundreds of thousands of insurance policies that must be converted it is better to do the conversion in smaller pieces to ensure that the conversion happens in the time period given. During the conversion process there must be time to put aside some policies that create problems during conversion and to transfer those policies to the next conversion round. This way the policies which failed to convert can be examined and fixed before the next conversion is executed. This is because insurance policies can be so complicated that even the pre-validation tool doesn't find all the errors. If some new type of error is found while converting policies to the new system, a new validation rule can be made to the pre-validation tool to find these kinds of errors in the future. This way the same type of error in an insurance policy would not get through the validation in the next validation round.

After the conversion, the insurance policies need monitoring because it must be confirmed that the policies start to live correctly in the new system. This way it is ensured that the conversion went through correctly and that the whole process from specification to conversion was success. It is also important to monitor the old policies in the new system and make sure that the newly converted policies didn't affect the old ones. It would be a major failure in the conversion process if it altered the old policies in

2 CONVERSION PROCESS

the new system. Also, by monitoring the old policies in the new system, it will be confirmed that the conversion process did not affect those policies and that the new system works correctly after the conversion.

2.3 External demands concerning the conversion of insurance policies

There are external demands that concern conversion of insurance policies. These demands should be taken care of before the conversion process can begin.

The insurance policies are confidential material. This means that the people who are working with the conversion of insurance policies and have access to the conversion material have an obligation of professional secrecy. (Alhonsuo et al. 2012, p. 134-136)

Information in the policies must stay unchanged through the conversion process. It would be a major failure if, for example, a money amount would change or insurance beneficiaries would transfer to the wrong person. If any faulty information in an insurance policy is found with the pre-validation tool, then this information should be fixed in the legacy system and conversion should be done with the correct data.

The conversion schedule must be planned with care. The conversion must be done when there are no other users using the system. That is why conversions are done by night or during weekends. The number of insurance policies which are converted at the same time is also limited by the time the conversion takes. When the total amount of policies that are going to be converted comes to several thousands, the whole conversion process will take so much time that the conversion must be done in pieces. So, the conversion of one policy base in a legacy mainframe computer system may contain many separate conversions of insurance policies. When the validation of insurance policies is executed with a separate pre-validation tool, the whole validation process and the possible fixing of policies with errors can be separated from the conversion process and can be done before the conversion. This gives more time for the conversion itself and ensures that the conversion process will go through without errors because it is known that the policies are validated and include no errors.

The insurance company which has ordered the conversion has to inform the customers who own the policies to be converted. This has to happen a certain amount of time before the conversion and must be considered when making the conversion schedule. The insurance companies have an obligation of continuous reporting to the client. (Alhonsuo et al. 2012, p. 153-154)

2 CONVERSION PROCESS

2.4 Different ways to validate insurance data

Rule based validation is not the only possible way to validate conversion data. There are also other methods that can be used in validation. This chapter concentrates on different ways to validate insurance data. Different ways to validate insurance data are explained, and negative and positive aspects of those validation ways are considered.

2.4.1 Procedural validation

Procedural validation means validation made with some known programming language with the help of *if*, *if-else* and *then* statements. Below there is a list of negative and positive aspects of procedural validation.

Negative aspects of procedural validation

- Validations must be made by somebody who knows the programming language and a business analyst often doesn't have this knowledge.
- Documentation and finding documented validations from code might not be easy.
- Maintaining and making changes to the validations could be difficult when validations are inside the program.
- Complex business validations are not easy to present using traditional programming languages.
- When changes are made to validations over time with imperative programming languages, the code can easily become very disoriented. (Bali 2009, p. 7-8)

Positive aspects of procedural validation

- Validations and the program, which uses those validations, are the same.

As we can see, there are many more negative than positive aspects when using procedural validation. When the number of validations increases to hundreds or thousands, maintaining and categorizing the validations will not be easy and maintaining the whole validation process is very difficult. When the validations are all inside the code and changes are made to those validations, it could easily affect other parts of the code and the possibility of bugs increases and finding those bugs becomes harder when there are hundreds or thousands of validations.

2.4.2 SQL based validation

SQL based validations are made with SQL queries from the database. This would obviously mean that the data which is to be validated must be located in the database. This is a positive aspect from a certain point of view but it would also require a connection to the database which is not always wanted.

2 CONVERSION PROCESS

Negative aspects of SQL based validation

- Validations should be made only by an SQL specialist. Few business analysts are familiar enough with the SQL language to do validations with it.
- If the validation is complex then an SQL query sentence will be very long and therefore hard to read and understand.
- An SQL based validation tool would need a connection to the database.
- The documentation and categorization of an SQL based validation is not easy for business analysts to understand.

Positive aspects of SQL based validation

- The data is located in the database, so validations can be made directly from an intermediate database where the conversion engine uses it. So there would be no need to do a transfer of policies after the validation to an intermediate database.

The challenge of an SQL based validation is that SQL queries are hard to understand for business analysts and therefore finding a common language for SQL specialists and business analysts can be difficult. Also, when validations are documented, SQL queries should be converted to a language that business analysts understand. The performance of SQL queries is not high, so it would be very slow to make thousands and thousands of SQL queries to the database and the time consumed in communication with the validation program and the database would be high. SQL is similar to rules from the point of view that SQL describes what is wanted to search, not how to do the search.

2.4.3 Rule based validation

Validation using rules is based on the use of rules with the rule engine. There must also be a computer program which uses the rule package and displays the results of the validation.

Negative aspects of rule based validation

- Rule technology must first be learned by people who are using it before use of the technology is effective.
- Rule engine's memory consumption is high. (Bali 2009, p. 12)

Positive aspects of rule based validation

- Rules are easy to understand for business analysts and program developers and so a common understanding of what the rules do or should do is easier.
- Documentation of rules is easier and understandable for all when a rule language is used.

2 CONVERSION PROCESS

- Rules are easier to maintain and are easily changed if business alters because rules are separate and documented.
- Rule engine uses Rete algorithm which is very effective and in theory the number of rules does not affect the rule engines performance.
- When rules are separate and documented, re-use of the rules is easy.
- Rules can be embedded into existing applications.

(Bali 2009, p. 10-11; Browne 2009, p. 29)

There are many positive aspects but there are also some things which should be considered before starting to use the rules. When using the rules, the developers and analysts who are working with the rules should be trained first. If rules are used without knowledge of what those rules are and, more importantly, how the rules and the rule engine work, it might result to inefficient rules and unpredictable results. Memory consumption is another disadvantage of using the rules. When validating a large amount of insurance policies, the number of facts and properties gets very high and memory usage of the rule engine increases. The price of memory is not very high at the present day, especially when comparing the loss of money to higher amounts of memory with the advantages which use of rule engine brings.

Use of the rule engine is not wise if you have only a small amount of rules, the rules do not change often, or if the rules are very simple and don't contain any complex business logic. In that case, the rule engine would be overdoing it and procedural validation should be enough. (Bali 2009, p. 12-13; Browne 2009, p. 29)

3 BUSINESS RULE CONCEPTS

In this chapter, concepts related to business rule systems are explained. In chapter three, business rules themselves and the modeling of business rules from a structured business vocabulary are presented with examples.

Business rules are a way to align business capabilities with strategies to solve business problems. Business operations are highly complex and becoming more so by the day. The use of business rules will still require lots of structure analysis of business complexity and planning. Because of the declarative nature of the rules, they can be maintained and re-factored easily.

With the use of the rule language, business rules will become unambiguous for all who are working with the business. With the business rule management system, also known as BRMS, business rules and facts that business includes, can be managed effectively. The rule engine with the Rete algorithm makes execution of business rules fast and efficient. With all these things combined, the use of business rules can improve business and business decisions in many environments, including validation of business data. The use of business rules provides agility and flexibility. (Ross and Lam 2011 p. 6-11)

Drools combines all of the features mentioned above. In this work, Drools version 5.5.0.Final is used. This Drools version contains five different modules: Drools Guvnor which is a business rule management system (BRMS) and a business process management system (BPMS), Drools Expert (rule engine), Drools Fusion (CEP), Drools Flow (process and workflow) and planner. Two of the first modules are used and presented in this work. (The JBoss Drools Team. 2012 [b] p. 1)

3.1 Rule engine

The rule engine is a computer program that delivers Knowledge Representation and Reasoning (KRR) functionality to the developer. The rule engine contains three main components which are:

- **Ontology**
Ontology is the representation model of the facts that are needed.
- **Rules**
Rules are the brain of the rule engine and they perform the reasoning.
- **Data**
Data is the target on which the rules are used to get the outcome.

3 BUSINESS RULE CONCEPTS

Drools Expert is the rule engine which is used to implement the pre-validation tool. This rule engine uses implementation of Rete algorithm called ReteOO. ReteOO algorithm is enhanced and implemented for object oriented systems. This enables Drools Expert to be able to scale to a large number of rules and facts.

A rule is a two-parted structure which is formed from a 'when'-part and a 'then'-part. The 'when'-part is also called the 'left hand side of the rule' and it contains the conditions of the rule. The 'then'-part of the rule is also called the 'right hand side of the rule' and it contains the action part of the rule. All existing facts are matched against rules and it is called pattern matching. Rules and facts are stored in to the working memory.

There are two types of Production Rule Systems (PRS). 'Forward chaining' which is reactive and data-driven or 'backward chaining' which uses passive query. In a forward chaining system when a new fact is inserted into the working memory, evaluation starts and if a rule becomes true, it will be scheduled for execution. A backward chaining system starts from the conclusion which the engine tries to satisfy. If it can't, then it searches for a conclusion that it can satisfy. These are called sub-goals and they will help to satisfy some part of the initial goal. The engine continues this process until either the initial conclusion is proven or there are no sub-goals left.

The Drools Expert is a Hybrid Reasoning System (HRS) which means that it can be either a backward chaining or a forward chaining system. (The JBoss Drools Team 2012 [a], p. 2-8)

This work concentrates on the use of the Drools Expert rule engine but there are also other rule engines/expert systems available, such as IBM WebSphere, ILOG Jrules, FICO Blaze Advisor business rules management, Progress Corticon business rules management system, Haley business rules engine, Pegasystems PegaRules, Production Systems Technologies OPSJ, SweetRules (which is the first open source platform for semantic web business rules), JESS Rule engine for Java and many more. It is possible to make a rule engine yourself and it may be the best option in some specific case but there are so many rule engines for different uses, that normally it is wiser to choose the rule engine from the market. (Bali 2009, p. 15)

3.2 Rete algorithm

The Rete algorithm was designed by Dr. Charles L. Forgy of Carnegie Mellon University and it was first published in a working paper in 1974. The Rete algorithm creates a network from rule conditions and each condition is a node in the Rete network. The Rete network is a rooted, acyclic and directed graph. Drools uses an improved version of the Rete algorithm called ReteOO. The performance of the Rete algorithm is theoretically independent from the number of rules used. The Rete algorithm is efficient but memory usage is high because of the large amount of caching required to avoid the multiple evaluating of conditions. (Bali 2009, p.258)

3 BUSINESS RULE CONCEPTS

3.3 Example of how the Rete algorithm functions in the pre-validation tool

The Rete network consists of different node types. The following example describes the node types that the Rete network uses. This example also shows how a real rule in the pre-validation tool is processed by the Rete algorithm. Below, the 'when'-part of the rule “*CV_AGRE Agreement must have at least one payment*” is presented and in figure 3.1 the Rete network which is produced by the rete algorithm using the 'when'-part of the rule. Node types are explained below the figure 3.1.

```
rule 'CV_AGRE Agreement must have at least one Payment'  
when  
  ValidationOptions ( businessValidation == true)  
  Agreement ($aid : agreementID != null)  
  not exists(Payment(agreementID == $aid ))
```

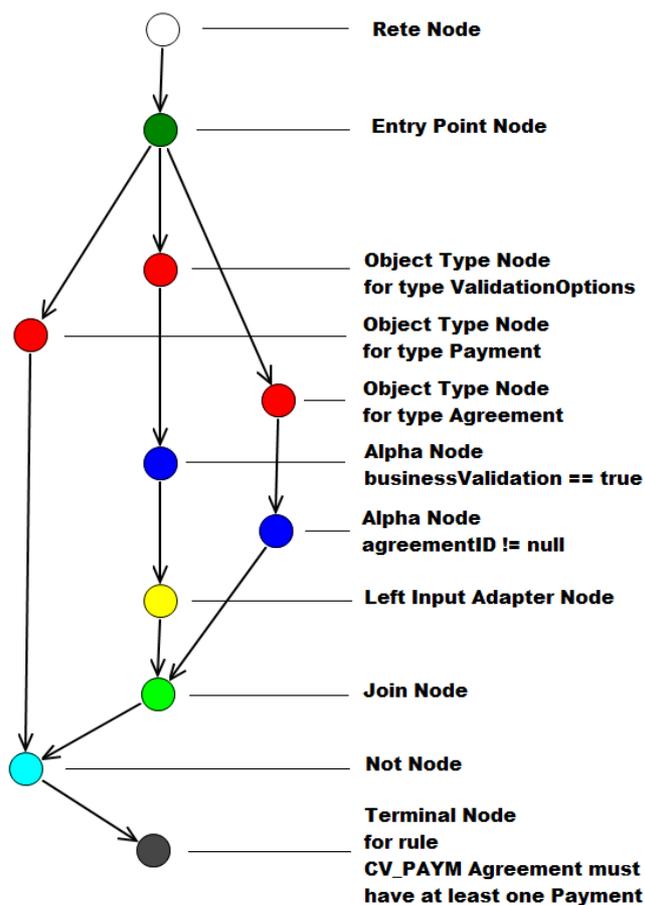


Figure 3.1: Rete network from the 'when'-part of the rule “CV_PAYM Agreement must have at least one payment”.

3 BUSINESS RULE CONCEPTS

Rete Node

Rete Node is the entry point for the facts to the Rete network. When a fact is inserted into the knowledge session, it enters the network through this node. *Rete Node* is the topmost node in the figure 3.1

Entry Point Node

Entry Point Node corresponds to an entry point. There can be many different entry points. In figure 3.1 there is only one *Entry Point Node*. With the use of Drools Fusion it is possible to define more named entry-points and then the Rete network would have multiple entry points for the facts.

Object Type Node

This node functions as a fact type filter. It lets through only fact types that are matching with desired fact type. In figure 3.1 there are three *Object Type Nodes*, one for the fact type Payment, one for the fact type ValidationOptions and one for the fact type Agreement. Only those facts can get forward in this Rete network. The first node of each branch which starts from the *Entry Point Node* must be an *Object Type Node*.

Alpha Node

Alpha Node is for evaluating the properties of a single fact. In figure 3.1 there are two *Alpha Nodes*. One is for evaluating the businessValidation property of the fact type ValidationOptions; if this boolean property is true, then the flow continues to the next node. The execution will stop at this point if the fact doesn't satisfy the constraint. If there are multiple constraint evaluations for a single fact, then there will be as many consecutive *Alpha Nodes*.

The order of the property evaluations is very important. For example, if there is a complex function evaluation in the first *Alpha Node*, then there will be a simple boolean evaluation in the next *Alpha Node*. If the simple boolean evaluation would then restrict the flow to the next node, then the complex function evaluation would be done for nothing. The order of *Alpha Nodes* also affects the re-usability of the Rete network. Re-usability is explained later in this chapter.

Left Input Adapter Node

Left Input Adapter Node acts as an entry point to the *Beta Nodes*. *Left Input Adapter Node* creates a tuple out of a single fact. In figure 3.1 there is one *Left Input Adapter Node* which creates a tuple from ValidationOptions fact. The tuple then continues to the next node which is in this example called a *Join Node*.

3 BUSINESS RULE CONCEPTS

Right Input Adapter Node

Right Input Adapter Node makes the tuple behave like a single fact. This Node allows the creation of more complex Rete networks. The *Right Input Adapter Node* is not presented in this example.

Beta Nodes

The function of *Beta Node* is to evaluate constraints on two or more facts. *Beta Node* has left and right input. The left input is for tuples and the right input is for the facts. Both inputs have associated memory where they store partial matches. There are several different kinds of *Beta Nodes*: *Join Node*, *Not Node*, *Accumulate Node*, *Collect Node* and *Exists Node*.

The Join Node's purpose is to join the tuple with the fact. In figure 3.1 the *ValidationOptions* fact with *businessValidation* equal to true represents the tuple which comes from the left input and the *Agreement* represents the fact that comes from the right input. If all of the conditions are satisfied, a new tuple of size two is created which contains the *ValidationOptions* and the *Agreement* fact. After the tuple is created it continues to the next node.

The Not Node evaluates if a *Payment* fact exists for the *Agreement* fact. If there is a *Payment* where *agreementID* matches to the *Agreements agreementID*, the flow continues to the next node.

Terminal Node

Terminal Node is the end node of the Rete network. If all the rule conditions are satisfied then the rule is placed to the agenda for execution. All the rules have at least one *Terminal Node*.

Node Optimizations

Re-usability is one of the optimizations used by the Rete algorithm. With the re-usability, size of the Rete network can be minimized. This is also the reason why the execution order of the *Alpha Nodes* is important. As explained before, if we have a rule for checking that all the *Agreements* have at least one *Payment* and another rule which would check in a similar way that the *Agreement* has some other fact which would be mandatory for an *Agreement*, then node sharing would be possible. If a branch that has *Object Type Node* for the fact *ValidationOptions*, the *Alpha Node* for the property *businessValidation* and also the *Left Input Adapter Node*, then all those nodes would be re-used when evaluating the new rule. Also, a branch with *Object Type Node* for the fact type *Agreement* and an *Alpha Node* for the property *agreementID* would be re-used. In fact, both branches and the *Join Node* which combines those two branches can all be re-used in this new rule to check another mandatory fact for the *Agreement*. Only a new branch for a mandatory fact type is needed for this new Rete network. The re-usability

3 BUSINESS RULE CONCEPTS

is the reason why it is important to put conditions in rules in the same order as they are in the similar rules.

Node indexing is another optimization that is used. Fact values can be indexed with a hash table and so use of those facts is faster. (Bali 2009, p. 257-272)

3.4 Rule-language

One of the challenges when creating the business rules is to make the rules understandable to all who are working with the rules. This is possible when using clear sentence forms, so that different people working with the business rules express and understand the business rules in the same way. It is not important which rule engine or which programming language is in use, the most important thing is effective business communication.

A well written business rule is always a sentence. Every business rule can be expressed by using words *must* or *only*. These two keywords make the business rules unambiguous and remove freedom of interpretation. Using words shall, should or might instead of must and only doesn't make the rules unambiguous and different people who are reading that rule might understand it differently. If that happens the business rule is not strict and mistakes may happen.

Some business rules need a statement of advice and those statements should be done using words *may* or *need not*. These kinds of guidance sentences should always have a subject. If the guidance sentences have explicit subjects and are clearly written, then those sentences are easy to follow.

For example, if we have a business rule that says "*Agreement should have agreementID*", then we are not certain that the agreementID for an Agreement is mandatory. But if the business rule is written "*Agreement must have agreementID*", the business rule becomes unambiguous and it is understood by all. Statement of advice can be written in a rule like "*Agreement whose status is 'in force', need not to have Claims*". (Ross (2009) [b] p. 2-8)

3.5 BRMS

The Business Rule Management System (BRMS) is a software which is used for creating, managing, maintaining and monitoring of the business rules and the processes in a multi-user environment. Also, the term 'authoring rules' is used when changing or maintaining the rules is in question. Drools Guvnor is a web based BRMS which was used in the making of the pre-validation tool. The Guvnor is useful in situations when version management of rules is needed, multiple users should have an access to the rules with different user levels, there is no existing infrastructure for the managing of rules or if there is a large amount of business rules. With the help of Drools Guvnor the rules and the business processes can be in one place and can be created, tested and

3 BUSINESS RULE CONCEPTS

reviewed by different people. User roles for business analysts, rule experts, developers, administrators etc. are possible. In figure 2.2 the web-based user interface of the Drools Guvnor is shown. (The JBoss Drools Team. (2012) [c] p. 1-2)

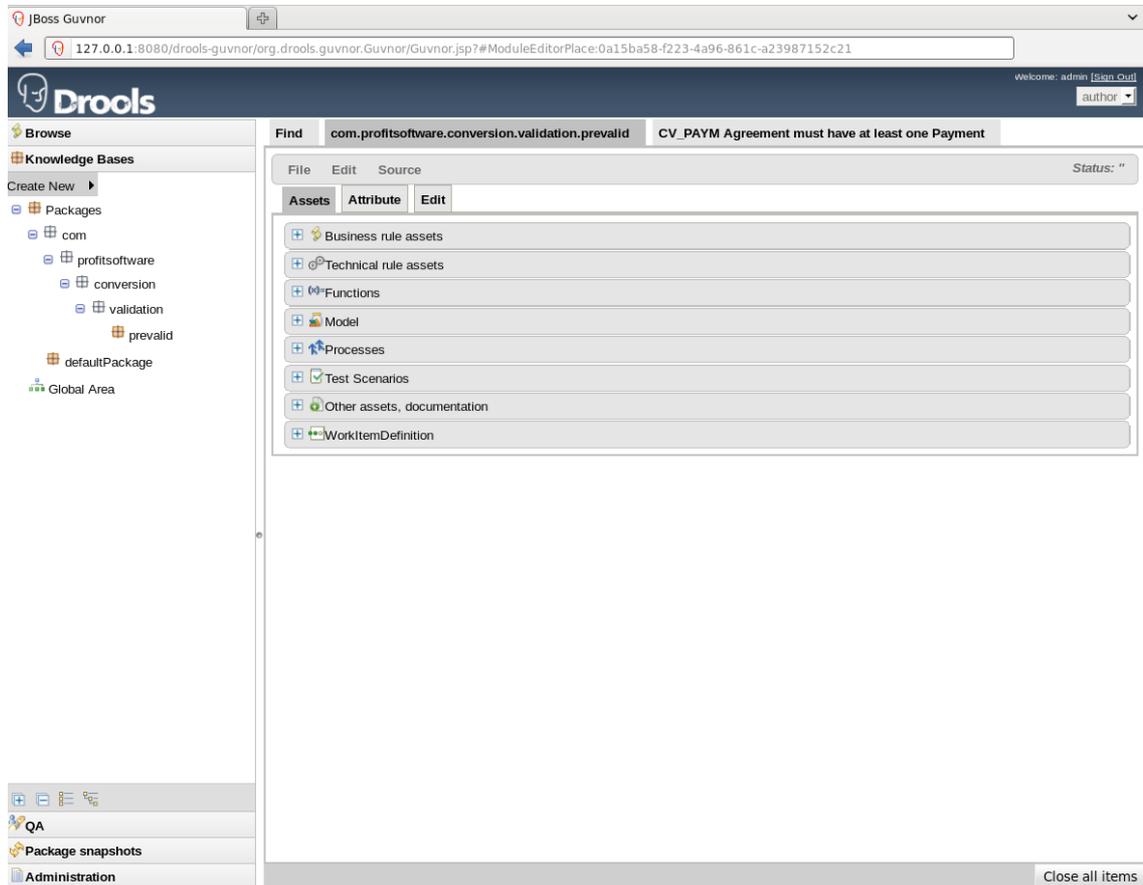


Figure 3.2: User interface of Drools Guvnor.

There are different ways for the Guvnor to make the rules. The rules can be made with the guided editor (BRL) or just in a plain text (DRL) with the technical rule editor.

The guided editor uses the knowledge of the current fact/concept model and in this way helps the user to make the business rules. Figure 5.2 shows a rule made with the guided editor. The editor can also augment DSL sentences. The DSL rules are textual rules that use a natural language configuration asset to control how they appear.

Decision table rules are stored in spreadsheet form and one row in the spreadsheet represents one rule and each column is a condition, action or option. The Guvnor has a guided decision table editor which works similar to the guided business rule editor, but it creates decision table spreadsheets. The Guvnor translates the decision table rules, the business rules and the DSL rules into a DRL format after rules have been written. (The JBoss Drools Team. (2012) [c] p. 26-40)

3 BUSINESS RULE CONCEPTS

The Guvnor offers management of rules by categorization. The user can make a category hierarchy to the Guvnor and sort the rules to those categories. This makes the managing and the finding of the rules from a rule set easier when there is a large amount of rules. Rule categorization of the pre-validation tool is shown in the chapter 5.2.

The Guvnor allows the user to manage several rule packages at the same time. Each rule package contains its own assets (rules, models, functions, processes etc.). The rule packages are independent from one another. When the rule package is ready to use, it must first be built and after that the package can directly be embedded into business application.

3.6 Testing of the rules

Testing of the rules is important because if there are errors in the business rules, it can result to big losses in the business. If there are errors in the rules of the validation tools final version, the conversion could fail because there may be erroneous policies in the conversion. This would conclude to re-doing the conversion of those invalid policies. When the amount of rules is in the hundreds or thousands, it is almost certain that there are human made mistakes in those rules. The mistakes may originate from the person who wrote the rules, the analyst who defined the rules or practically from anybody who was working with the creation of the rules. Creation of the rules should follow the cycle presented below.

- 1. Writing of the rules.** The rules should be written strictly based on the specifications or on the requests of the end users. If the rules are based on the requests of the end users and are not documented, then the specification of those rules should be done while writing the rules.
- 2. Testing.** After the rule is written, it should be tested immediately. Also effects to the other rules should be tested. If you write lots of rules and test them afterward, it could be difficult to identify the root cause of the error. The errors and the fixes in the rules might affect to the other rules and the testing of many new rules at the same time can be troublesome.
- 3. Reviewing.** The rules should be reviewed after the testing by the people who have defined the rules. In the definition of the rules, mistakes can also happen and after the rule has been written, it might be easier for the definers to point out the mistakes in the rules. The rule might be written correctly as per the specifications, but there can also be mistakes in the specifications due to the complexity of the business.

3 BUSINESS RULE CONCEPTS

These steps should be repeated as many times as is necessary for creating flawless business rules. Because the nature of the business, it is important that the business rules are correct and those rules must be tested well due to the complexity of the business rules. (Browne 2009, p. 145-146)

The testing of the rules is also a function of the Guvnor and test scenarios can be done with the guided editor. In the test scenario, the user of the editor can insert facts into the test scenario with the desired property values. Then the user can define which rules are active in the test scenario. In the 'expect'-part of the test scenario, the user can produce desired results after the facts have been inserted and the rules fired. In figure 3.3, a test scenario created to test the “*CV_AGRE Agreement must have at least one PaymentPlan*” rule is presented.

The screenshot shows the Guvnor test scenario editor interface. At the top, there are two tabs: "Test CV_AGRE Agreement must have at least one PaymentPlan" and "CV_AGRE Agreement must have at least one PaymentPlan". The status is "Draft". The interface includes a menu bar with "File" and "Edit", and sub-menus for "Attributes" and "Edit". A "Run scenario" button is visible. The results section shows "Results: 100 %" and a summary: "Rule [CV_AGRE Agreement must have at least one PaymentPlan] was activated 1 times." The audit log section has a "Show events" button. The "GIVEN" section includes: "Insert 'ValidationOptions' [\$vo] businessValidation: true", "Insert 'Agreement' [\$a] agreementID: 123456", "Insert 'PaymentPlan' [\$pp] agreementID: 999999", and "Activate rule flow group validate_paymentplan". The "CALL METHOD" section has "Add input data and expectations here." and "1 rules fired in 0ms. :Rules fired: CV_AGRE Agreement must have at least one PaymentPlan [1]". The "EXPECT" section has "Use real date and time" and "Expect rules CV_AGRE Agreement must have at least one PaymentPlan: fired at least once". The "More..." section has "Allow these rules to fire: CV_AGRE Agreement must have at least one PaymentPlan". The bottom right has a "Close all items" button.

Figure 3.3: The test scenario for the rule “*CV_AGRE Agreement must have at least one payment*”.

3 BUSINESS RULE CONCEPTS

In this scenario, one Agreement fact and one PaymentPlan fact with different agreementID values have been inserted. The rule which is tested, tells that all Agreements must have at least one PaymentPlan. This rule should fire because in this scenario, there is one Agreement fact but no PaymentPlan fact where agreementID property value would be the same as in the Agreement fact. In the 'expect'-part, it is expected that the rule “*CV_AGRE Agreement must have at least one PaymentPlan*” should fire.

After the scenario has been run, we can see that the rule “*CV_AGRE Agreement must have at least one PaymentPlan*” has fired. The results are 100% and the test has been successful. There can be one or more test scenarios for every rule, to confirm that the rules will create desired results. (The JBoss Drools Team. (2012) [c] p. 83-87)

4 MODELING VALIDATION RULEBOOK

Before the data can be validated it must be in a known format. The conversion engine reads the data from the CSV files and those files contain all the information that the policy has. The content of the CSV files is defined in the conversion engine interface specifications. In this chapter, it is presented how the fact model and the rulebook are created from the conversion engine interface specification of the Payment.csv file. The business aspect also defines more business rules to the rulebook. One line in the Payment.csv file is one single Payment fact for an Agreement fact. The payment is linked to the agreement with an agreementID.

4.1 Structured Business Vocabulary (Fact model)

The structured business vocabulary is a verbal model of operational business knowledge. With the help of the rule language, it is possible to make a business vocabulary which allows talking and writing of the business in a consistent manner. Structured business vocabulary is defined as the set of terms and their definitions, along with wordings, that organizes the operational business know-how. (Ross and Lam 2011, p.103-105)

In table 4.1 the conversion engine interface specification of the Payment.csv file is shown. The Payment is a fact type and contains 14 properties that are shown below.

- **Fact type:** Payment
- **Fact properties:** rowID, agreementID, paymentID, companyAccount, companyDate, agreementDate, payerAccount, sum, referenceNumber, payerNotes, payerName, riskPremium, paymentType, paymentSubType.

Table 4.1: Conversion engine interface description (fact model) of Payment.csv.

Required:	Yes, at least one Payment for each Agreement
Identifiers:	RowID identifies each row in the CSV.
CSV file name:	Payment.csv

No	Name	Data type	Description	Req.
1	RowID	String	Unique for each row in CSV.	Yes

4 MODELING VALIDATION RULEBOOK

2	AgreementID	String	Agreement to which the payment is associated.	Yes
3	PaymentID	String	Unique id for the payment.	Yes
4	CompanyAccount	String	Company account number.	No
5	CompanyDate	Date	Date when insurance company receives the client payment in their bank account.	Yes
6	AgreementDate	Date	Date when payment is matched to policy (start date of payment matching batch)	Yes
7	PayerAccount	String	Payer account	No
8	Sum	Float	Payment gross sum	Yes
9	ReferenceNumber	String	Reference number	No
10	PayerNotes	String	Payer notes	No
11	PayerName	String	Payer name	Yes
12	RiskPremium	Float	Risk premium	No
13	PaymentType	Int	Payment type: 0 Unidentified 10 Payment refund 20 Normal payment 30 Recourse payment 40 Risk payment 50 Risk payment refund 60 Risk recourse payment 70 Lump sum payment for a group policy 80 Refund the lump sum payment for a group policy	Yes
14	PaymentSubType	Int	Payment subtype: 0 Normal 10 BankCorrection	No

4 MODELING VALIDATION RULEBOOK

The data type field, the description field and the required field give requirements to the Payment fact properties and those requirements create the rules to the rulebook. This conversion engine interface description works as the structured business vocabulary for the Payment fact.

4.2 Rulebook

The rulebook is a collection of the business rules, along with the terms, definitions and wordings that support them and are presented in the fact model. The rulebook needs maintaining and it is never finished. If a change happens in the business, then it also affects the business rules and those changes must be updated to the rulebook to get the most out of it. There is no use for a rulebook if the business rules inside are not valid anymore.

If there is a change in the conversion engine interface specifications for the Payment.csv, then it would affect the validations also. If the rulebook is not updated and the validations are done with old rules, then unnecessary validation errors or errors in the data that will not be found can occur. (Ross and Lam 2011, p. 227-231)

4.2.1 Conversion engine interface rules

The conversion engine interface rules are named as the format validation rules in the pre-validation tool. These rules are created based on the knowledge from the fact model of the Payment. The validation rules are created for each property of the Payment by going through the whole fact model of the Payment column by column and evaluating the possible demands in each column for a validation rule. The rules which concentrate on single properties in the fact model and are defined in the conversion engine interface description are placed to the *formatValidationRule* category in the Drools Guvnor.

The rules are named so that first there is an abbreviation of the conversion validation, *CV*. This identification is used because there are also other kinds of projects done by Profit Software that include usage of the rules and identifying the rule sets so that there will be no complications between different projects is desired. The second part is the abbreviation of the name of the CSV file to which the validation points. The rules which are used to validate the Payment fact have the abbreviation *PAYM* and the rules which are related to the Agreement start with *AGRE*. The third part is the column number of the property of the CSV file to which the validation points, expressed with three digits. The last part of the whole rule name is the purpose of the rule and the reason for validation presented in the rule language.

4 MODELING VALIDATION RULEBOOK

Creating rules from the fact model

Next it is shown how the rules are created using the fact model (presented in figure 3.1). In the example, all the properties of the Payment are evaluated for the possible need of a validation rule or rules.

rowID

The first property of the Payment is rowID. The type field tells that the rowID is a string value. The format of the string value can not be validated because it may contain anything, so this column does not give any reason to make a validation rule.

The next column tells that the rowID is unique for each of the lines in this CSV file. That creates a demand that there can not be two identical rowIDs in this CSV file. That demand creates the first validation rule *“CV_PAYM_001 rowID must be unique”*. The third column of the rowID property tells that the rowID value is mandatory. This creates a demand that the rowID can not be empty. The validation rule *“CV_PAYM_001 rowID must be unique”* will give an error if it finds two empty rowIDs because then those rowIDs are identical. The name for this new rule is *“CV_PAYM_001 rowID must not be null”*. After this rule is added, a null check for the unique checking rule that was created before can be added. In case there are many lines without a rowID, there would be many 'rowID must be unique' errors and many 'rowID must not be null' errors. If there is a condition 'rowID must not be null' in the unique checking rule, then the rule engine does not have to do unique checks for the rowIDs whose values are null and there would not be double error messages in the case that there are many empty rowIDs.

agreementID

The next property for the Payment is agreementID. The type of this property is 'string', so it needs no validation rules. Description 'Agreement to which the payment is associated' tells that a Payment is connected to the Agreement with this agreementID. If the Payments agreementID would refer to an Agreement which would not exist, the Payment would be unnecessary and this could derive from an error in the system. The rule name for this rule is *“CV_PAYM_002 There must be Agreement for each Payment”*. The last column tells that the agreementID is required and the rule to check this is *“CV_PAYM_002 agreementID must not be null”*.

paymentID

PaymentID is also a 'string' type and that type needs no validation. The description reads 'Unique id for the Payment', which means that this id must be unique for each Payment of an Agreement. So there can be identical paymentIDs in a Payment.csv file, but all the Payments that are connected to one Agreement must have unique paymentIDs. The rule name for this rule is *“CV_PAYM_003 paymentID must be unique for each Payment”*.

4 MODELING VALIDATION RULEBOOK

concerning one Agreement". The paymentID is also required and the rule to validate that is *"CV_PAYM_003 paymentID must not be null"*.

companyAccount

The companyAccount property is a 'string' type, and it is not required. The description tells that this is a company account number and there are no requirements written to that number. There are no validations that could be made for this property, because this property is not required and its content could be anything.

companyDate

The companyDate is a 'date' type property. All the properties are first collected in a 'string' form and after that converted to the desired data types. This is explained in chapters 5.4 and 5.5. The result of that type conversion can be used to ensure that is the data in the desired form. The 'date' type check is made with the rule *"CV_PAYM_005 companyDate must be in date form"*. The description of this property does not give reason for a validation rule. CompanyDate is required and the rule to validate it is *"CV_PAYM_005 companyDate must not be null"*.

agreementDate

The agreementDate has similar requirements as companyDate and the validation rules are also similar, *"CV_PAYM_006 AgreementDate must be in date form"* and *"CV_PAYM_006 agreementDate must not be null"*.

payerAccount

The payerAccount property is similar to the companyAccount and also does not have anything to validate.

sum

The sum property is the amount of money in each payment. The type of the sum is 'float' and it is required. The validation rules *"CV_PAYM_008 sum must be in float form"* and *"CV_PAYM_008 sum must not be null"* are required.

referenceNumber

The referenceNumber is a 'string'-type property and it is not required, so there are no validations to be done for the referenceNumber property.

payerNotes

The payerNotes is a 'string'-type property and it is not required, so there are no validations to be done for the payerNotes property.

4 MODELING VALIDATION RULEBOOK

payerName

The payerName is a 'string'-type and it is required and so there must be the rule “*CV_PAYM_011 payerName must not be null*” to confirm that the value of this property is not empty.

riskPremium

The riskPremium is a 'float'-type but it is not required. If the riskPremium value exists, then the type of the value must be 'float'. This requirement makes the validation rule a little bit different than before: “*CV_PAYM_012 if riskPremium exists it must be in float form*”.

paymentType

The paymentType field is 'integer'-type and it is required. The description tells that there are nine different types of payments. Because the paymentType is required and only those nine types of payments are allowed, the rule name is “*CV_PAYM_013 paymentType must be one of the following (0,10,20,30,40,50,60,70,80)*”.

paymentSubType

The paymentSubType is similar to paymentType, but it is not required. The validation rule for this property is “*CV_PAYM_014 if paymentSubType exists then it must be one of the following (10,20)*”.

Now we can see that some properties did not need validations at all and some properties would need more than one validation rule. This fact model gave reason to create 14 format validation rules. There are also many similar validations, just with different properties. These similarities continue through all the CSV files and therefore the writing of the rules becomes faster after the rules for one CSV file are created.

4.2.2 Business rules

The business rules are defined by business jurisdiction; this means that the business can enact, revise and discontinue the business rules as needed. The business rules give guidance needed in business operations and a rule serves as a guide for a conduct or an action. All expressions of the business rules should be based on the structured business vocabulary (fact model). (Ross 2009 [a], p. 83-86)

The business rules for the validation are defined by the business analysts and the mathematicians. For the Payment there is one business rule requirement defined and that is that all agreements must have at least one payment. This requirement creates the business rule “*CV_PAYM Agreement must have at least one Payment*”.

There can be many different kinds of business requirements for the validations. For example, there could be some payment with a paymentType which may not have a

4 MODELING VALIDATION RULEBOOK

negative value for the sum property and there should be a validation rule “*CV_PAYM if paymentType is x then sum must not be negative*”. If the paymentType is some defined value, then a payerAccount would be required and the validation rule for that would be “*CV_PAYM if paymentType is x then payerAccount must not be null*”. These types of business validations are real in other CSV files.

4.2.3 Rules based on product parametrization

The rules based on the product parameters are dependent on the agreement's product type. The product parameters consist of the value limits for the properties for different insurance products. Bringing the product parameters to the validation session is implemented with the use of special internal API which is designed for accessing the parameter values.

The payment fact does not have any product parameter based rules but, for example, there is a rule in the pre-validation tool for checking that the percentage value of death cover is within limits, which depend on the insurance product's parameters. The rule name for that is “*CV_COVE death cover sumInsured must be in product parameter limits*”. The Agreement fact has a productID property that tells which insurance product is in question and this property is used to connect the right product parameters to the agreements.

These product parameters are read to memory only if the product parameter based rules are enabled from the properties file of the pre-validation tool. Reading of the parameter values consumes a small amount of memory but all the available memory is wanted to save for the rule engine when the parameters are not needed.

4.3 Business process model

“The best definition of business process I have found to date is Janey Conkey Frazier's: *the tasks required for an enterprise to satisfy a planned response to a business event from beginning to end with a focus on the roles of actors, rather than actors' day-to-day job.*” (Ross 2009 [a], p. 123)

The business process and the business rules are not the same. The business process takes operational business things as inputs and transforms them to outputs when the business rules do not transform anything. The result of a business rule might have an effect that transforms something. When the business rules and the business process are separated, it creates a stable business process. The biggest and fastest change in the business happens in business rules, not in the business process. (Ross and Lam 2011, p.71-73; Ross 2009 [a], p. 123-124)

In figure 4.1, the business process model of the pre-validation tool is presented. This is not an actual business process but it is a validation process and it can be expressed

4 MODELING VALIDATION RULEBOOK

like a business process. This model is created with jBPM Web Designer which is included in the Drools Guvnor.

Each task in the validation process has its own ruleflow group and this is how the rules are separated to their own process tasks. The validation starts with the 'Prepare Validation' task which includes the initialization of the CSV parser, the creation of the FactContainer fact, the reading of properties file and the creating of the ValidationOptions fact with the knowledge from the properties file.

After the preparation, the processing of the first CSV file begins. First, there is a collection task which includes reading of the data from the CSV file and conversion of the desired values to the correct type. The second task in the processing of the CSV file is the validation itself. The validation task contains all the validation rules concerning the CSV file which is in validation. The validation task continues for as long as there are any validations to be made to the CSV file. After the validation task, the flow continues to the next task. After the last validation task, the program writes all the ValidationError facts to a log file.

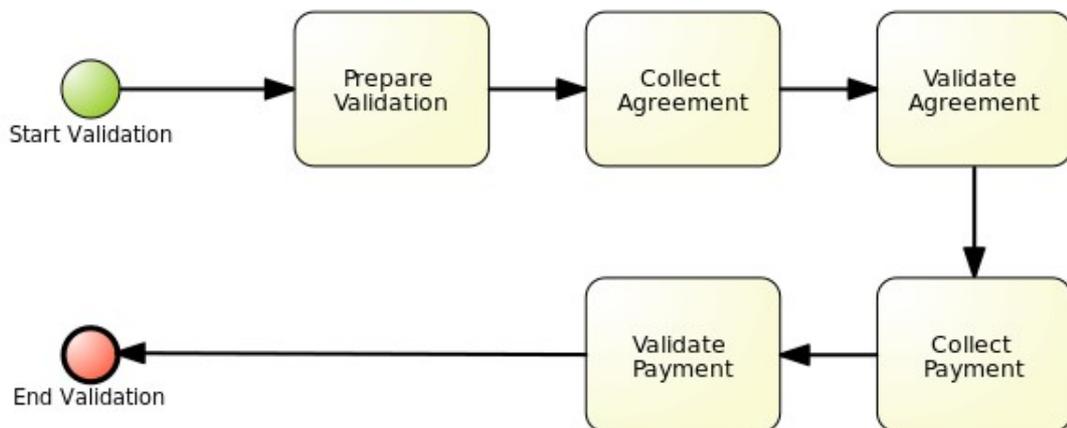


Figure 4.1: *The Business Process Model of pre-validation tool.*

In figure 4.1, the validation process with only two CSV files is shown (the Agreement.csv and the Payment.csv). In the pre-validation tool, the process would continue to follow the same pattern as shown in this figure, only the amount of collection and validation tasks would increase whereby the amount of the CSV files increases.

5 CONVERSION DATA PRE-VALIDATION TOOL

The conversion data pre-validation tool is a program that is used to validate conversion source files which are in CSV format. The tool is a console based, stand-alone program that analyzes the CSV formatted files and it writes a log file based on the findings.

The pre-validation tool is implemented using Java and it uses The Drools rule engine to validate the conversion data. There is a validation rule for each data value that can be validated. The pre-validation tool also has rules that are used for reading the data from the CSV files and to convert the data to the correct type. The analyzing logic is based on 1204 rules. There are 562 rules that are used for data validation and the rest of the rules are for other functionalities of the pre-validation tool.

The pre-validation tool works so that it first selects a file that it starts to validate. The progress is logged to the console and it can be seen which file is under validation. The program reads one line from a CSV file and collects the data in 'string' format from the columns in that line. Next, the program converts 'string' values to different formats if needed ('integer', 'float' or 'date'). After that the pre-validation tool validates all the values from that line with rules that concern the CSV file being validated. After validation, if the pre-validation tool finds errors, it writes error messages to the memory and starts to read the next line to the memory. After the last line of the CSV file is validated, the program changes the validation phase to the next CSV file and starts to validate that file. When the pre-validation tool has validated all the CSV files, it writes all error messages to console and to log file.

5.1 Using rules with java program

The Drools Guvnor contains all fact models, rules, processes, functions and everything else that the rule engine needs to function. In the Drools Guvnor, the user can make a binary package which can be used by a java program. Building a package will collect all the assets, validate and compile those assets into a deployable knowledge package.

First, *KnowledgeBuilderFactory* creates a new *KnowldegeBuilder*. The *KnowLedgeBuilder* is an interface which is responsible for building the *KnowledgePackage* from the knowledge definitions (rules, processes and facts). Then the package from the Drools Guvnor is fed into the *KnowledgeBuilder* which creates a *KnowledgePackage*. *KnowledgeBase* is created by *KnowledgeBaseFactory* and the *KnowledgePackage* is fed into it. The *KnowledgeBase* is an interface that manages a collection of rules, processes and internal types.

KnowledgeBase is used to create a stateful or a stateless knowledge session. The pre-validation tool uses *StatefulKnowledgeSession*. The *StatefulKnowledgeSession* is the

5 CONVERSION DATA PRE-VALIDATION TOOL

- **Businessvalidationrule**

The Businessvalidationrule category contains the rules that need information from more than one CSV file and so need more than one fact to do the validations. These rules are defined by the business analysts.

The validation process is implemented with jBPM Web Designer which is included in the Drools Guvnor. The work-flow model that the validation process uses is presented in chapter 3.3.

Functions that the rules in the Pre-validator use are stored in the Drools Guvnor. There are functions for converting the data from a 'string' format to the correct data types, the functions to calculate check sums (IBAN, Finnish Y-tunnus, Reference number, Social Security Number) and functions to do different kinds of date comparison. For example, getting a person's birth date from their social security number for comparing it to the person's 'birth date' property and to make sure they match.

The dynamic model contains all the fact types and the properties that those facts have. Each individual CSV file creates one fact type and the properties within it. The facts that are created from the CSV files can also include other properties that are not found in the CSV file. Those properties are used to store data that the later validations use. An example of this is shown in chapter 5.5 when making the rule “*CV_PAYM Agreement must have at least one Payment*”. The dynamic model also contains other facts that the pre-validation tool uses. There are also the following fact types:

- **ValidationError**

This fact is always created when a validation error occurs. The properties of this fact contain the name of the rule which found the error, a description of the error, the false values that are related to the error, information that specifies the agreement which had errors and information that specifies the name of the CSV file and the line number where the value that contains the error was found.

- **FactContainer**

This fact contains lists of data from the CSV files that are too big to read to the memory in whole and unique checking is made for those CSV files using these lists. This is explained in the chapter 4.5

- **ValidationOptions**

ValidationOptions contains information about which validation rules are enabled and which are disabled.

5.3 Collecting data from the CSV files using the rules

Data collecting from the CSV files is implemented using rules. The rules use a CSV parser program to do the actual reading from a text file. The CSV parser takes care of

5 CONVERSION DATA PRE-VALIDATION TOOL

the reading of the CSV files including splitting out fields. Each line of values is returned individually with the number of the line from which they came. Before the reading of the data begins, CSV parser must be created and initialized.

One line in the CSV file forms one fact and the columns in that line are the properties of that fact. In a data collection rule, which column is connected to which property is mapped. The CSV reader reads all the properties in 'string' format. A data collection rule does not try to convert the values directly to the desired type because if the value is not valid and the reading does not succeed then the value remains null. The data is first read in 'string' format because it is important to preserve the false values as well. The false values are needed for error reporting.

After all the properties are read for the fact in 'string' format then there are rules which try to convert the properties to the desired formats. If the conversion of the property does not succeed, then that property is left null. In this way, validation of that property can be done by comparing the property which is in 'string' format and the property which is in the correct type format. If the property in 'string' format is not null and the property in the correct type format is null, then the conversion of that value did not succeed and therefore the original value is not valid. This way it is possible to make sure that the values are in the correct type format and to preserve non-valid values for error reporting. After the whole CSV file is read, the data collection rules wait for the validation rules to finish. After the validation, the data collection rules change from the reading phase to the next CSV file and start to read again.

5.4 Validating data using the rules

Validation of the facts begins after all the data in one CSV file is collected and the values are converted to the desired data types. If a validation rule finds an error, the rule creates a `ValidationError` fact with the needed information from that error.

The validation rules have switches for enabling and disabling the rules depending on what kind of validations the user wants to perform. In the pre-validation tool, the rules are divided into different validation sets depending on the validation type of the rule. That is because the user might not to want make all the validations at once. If the user has CSV files that contain a huge amount of agreements, it takes more time to make all the validations. The rules use the `ValidationOptions` fact to conclude if they are activated or not. There are five different types of validation rule sets from the pre-validation tool:

5 CONVERSION DATA PRE-VALIDATION TOOL

- **Unique checking**
These rules check that a given value is unique among the given facts.
- **Format validation rules**
The format validation rules check that the type changes have succeeded and the given values are within the given limits.
- **Business validation rules**
The business validation rule category contains the rules that need information from more than one CSV file and so need more than one fact to do the validations. These rules are more complex and are defined by the business analysts.
- **Product parameter based rules**
Product parameter rule set contain rules that need product parameters in the validations. The product parameters are read to memory only when this validation is enabled.
- **Customer specified rules**
Conversion projects from different customers sometimes contain customer-specific insurance information and the rules that validate that information varies between different customers. The customer specified validation rule group enables use of the pre-validation tool in different insurance conversion projects with changes only to this particular rule group.

There are many kinds of validations but most of the validations are similar in different CSV files, only the values are different. Next, the main types of validation rules are explained. These types of validation rules are also found in the rulebook which was presented in the chapter 4.

- **Unique checking**

This type of rule is made for checking that some value is unique among the facts. Each line in the CSV file is marked with a unique line number. Next is an example rule “*CV_AGRE001 rowID must be unique*” where the ValidationError fact is created if there are two or more agreement facts with the same *rowID* property.

```
rule 'CV_AGRE001 rowID must be unique'  
ruleflow-group "validate_agreement"  
when  
    ValidationOptions ( uniquesChecking == true)  
    $a1 : Agreement( $id1 : rowID != null )  
    exists ($a2 : Agreement( rowID == $id1 , this != $a1 ))  
then  
    ValidationError fact0 = new ValidationError();  
    fact0.setDescription( "Agreement rowID must be unique" );
```

5 CONVERSION DATA PRE-VALIDATION TOOL

```
fact0.setSeverity( "error" );
fact0.setRowid( $id1 );
fact0.setRulename( drools.getRule().getName() );
fact0.setId( $a1.agreementID );
insert( fact0 );
```

- **The data must be in the correct data type form**

These types of validations check if the data type conversions from 'string' format to the desired type succeed. The rule checks that if a property with a 'string' value is not empty, then there must also be a value in the property to which the pre-validation tool tried to convert it. Next, there is shown the rule “*CV_PAYM011 validate calculationDate*” which checks that if the *calculationDateTxt* property is not empty and if the *calculationDate* is empty, then the conversion of that date did not succeed and the date is not valid and a *ValidationError* fact is created.

```
rule 'CV_PAYM011 validate calculationDate'
ruleflow-group "validate_payment"
when
    ValidationOptions( formatValidation == true )
    $p : Payment($cdT : calculationDateTxt != "" && != null,
        calculationDate == null)
then
    ValidationError fact0 = new ValidationError();
    fact0.setDescription( "Payment calculationDate value not
        valid" );
    fact0.setSeverity( "error" );
    fact0.setRowid( $p.rowID );
    fact0.setRulename( drools.getRule().getName() );
    fact0.setFieldvalue1( $p.calculationDateTxt );
    fact0.setId( $p.agreementID );
    insert( fact0 );
```

- **Value must be in the given limits or in list of values.**

These types of rules check that the given value is within the given limits, or that the given value is in the list of values. Many values in the insurance policies must be within certain limits; for example, the percentage values must be within certain limits, some money amounts must not be negative and some enumeration values must be in a given list. The following rule “*CV_PAYM016 validate paymentType*” checks that the value of the *paymentType* property is in the given list of values.

```
rule 'CV_PAYM016 validate paymentType'
ruleflow-group "validate_payment"
```

5 CONVERSION DATA PRE-VALIDATION TOOL

```
when
    ValidationOptions( formatValidation == true )
    $p      :      Payment($pt      :      paymentType      not      in
        (0,10,20,30,40,50,60,70,80,90,100))
then
    ValidationError fact0 = new ValidationError();
    fact0.setDescription( "Payment paymentType value is not
valid" );
    fact0.setSeverity( "error" );
    fact0.setRowid( $p.rowID );
    fact0.setRulename( drools.getRule().getName() );
    fact0.setFieldvalue1( $p.paymentTypeTxt );
    fact0.setId( $p.agreementID );
    insert( fact0 );
```

- **There must be a fact related to an Agreement**

There are over 30 CSV files and therefore there are over 30 facts which are linked to one agreement fact. The exact amount of the CSV files in the conversion process depends on what kinds of insurance policies are in the conversion. For example, if there are no policies which are in a claim state, then there are no CSV files which contains information concerning claims. Some of the facts are mandatory for each agreement. For example, all agreements must have a PaymentPlan. The PaymentPlan contains information about the amount of payments, payment frequency, due dates, etc. Mandatory facts to the agreement are, for example, payer, policy holder and insured.

5 CONVERSION DATA PRE-VALIDATION TOOL

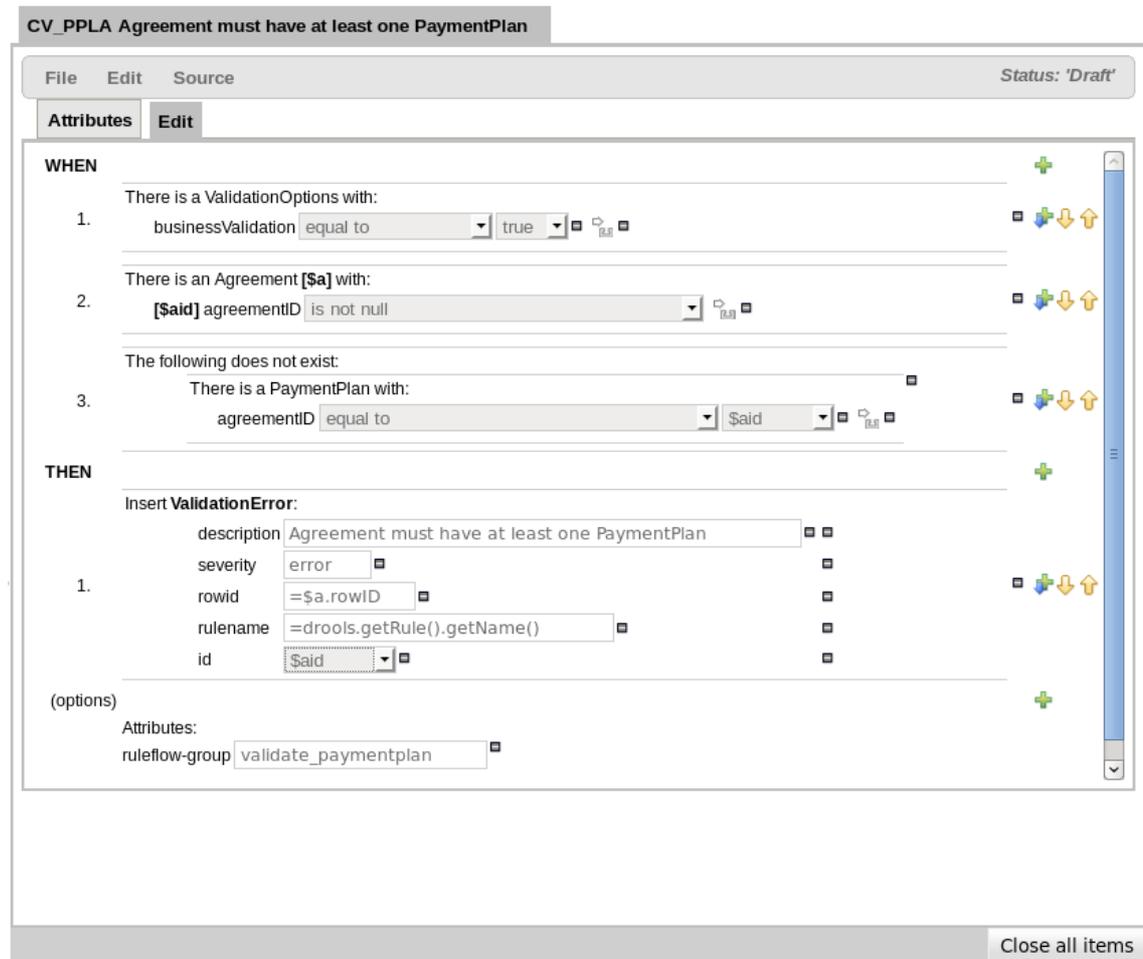


Figure 4.2: Rule “CV_PPLA Agreement must have at least on PaymentPlan“ is created with the Drools Guvnor Business rule editor.

In figure 4.2, a rule which checks that an agreement has at least one PaymentPlan is shown. This rule is created with the Drools Guvnor Business rule editor. In figure 4.2, we can see that the creation of the business rules does not need any coding skills and the rules with the business rule editor can be created by analysts without those skills.

5.5 Reading and preserving facts in memory

The facts are read to memory and validated after that. After the properties of the fact are validated, the fact can be removed from the memory, unless the fact is needed by some other rule than the format validation rules. Removing and preserving the facts in the memory requires great attention.

The memory usage of the pre-validation tool depends of how many agreements are in the CSV files which are to be validated and how much information the agreement contains (i.e. payments etc.). It would be easy to read all the facts to the memory and start the validation but when the CSV files contain 100000 agreements there can be over

5 CONVERSION DATA PRE-VALIDATION TOOL

ten million payments concerning those agreements. It is not possible to read all of those facts to the memory and start to do validations because it would need more memory than normal computers have at the present day. This is due to high memory usage of the rule engine. The pre-validation tool must be implemented so that it can be used for validating between one to 100000 agreements with a reasonable amount of memory. The pre-validation tool's memory consumption is planned so that the validation of 100000 agreements would go through with 8 Gb of memory.

When validating the properties from a single payment when there is no dependency between payments, there is no need for the other payments to be in the memory. But when making unique checks for the payment's rowID like explained in the previous chapter, it would require that all the payments must be in memory when executing that validation.

The payment's rowID unique checking rule *“PAYM_001 rowID must be unique”* is implemented by using the FactContainer fact which contains a list property where the payments' rowIDs are preserved. After one payment is read from the CSV file, the payment's rowID unique checking rule examines the list in the FactContainer to see if it contains the rowID of this payment. If the list contains this rowID already, then the rowID is not unique and a ValidationError fact is created. If the list does not contain that rowID, then it is added to the list and the validation continues to the next payment.

```
rule 'CV_PAYM001 rowID must be unique'
ruleflow-group "validate_payment"
when
    ValidationOptions ( uniquesChecking == true)
    $p1 : Payment( $id1 : rowID != null )
    FactContainer( payments contains $id1 )
then
    ValidationError fact0 = new ValidationError();
    fact0.setDescription( "Payment rowID must be unique, two
    same rowID's found" );
    fact0.setSeverity( "error" );
    fact0.setRowid( $id1 );
    fact0.setRulename( drools.getRule().getName() );
    fact0.setId( $p1.agreementID );
    insert( fact0 );
```

All agreements are read to memory in the beginning of the validation. The agreements are kept in memory for the whole validation session because there are many business rules that require information from many fact types to be in the memory at the same time and the agreement fact is usually one of those facts needed.

The rule *“CV_PAYM all agreements must have at least one payment”* requires that all the agreements and all the payments are in memory if the rule is implemented like

5 CONVERSION DATA PRE-VALIDATION TOOL

presented in chapter 3.3. This problem is solved by adding a 'boolean' property `hasAgreementPayment` to the agreement fact. It is set to 'false' when the agreement fact is created. When the payment fact is created to memory and all the properties are read to that fact, there is a rule “*CV_PAYM collect Agreement payments*” that converts that agreement's (to which the payment is related to) `hasAgreementPayment` value to 'true'.

```
rule 'CV_PAYM collect Agreement payments'  
ruleflow-group "collect_payment"  
when  
    ValidationOptions( businessValidation == true )  
    $a : Agreement($aid : agreementID != null, hasPayment ==  
        false)  
    exists $p : Payment(agreementID == $aid)  
then  
    modify ($a){  
        hasPayment = true  
    };
```

After all the payments have been processed, there is a rule that checks the value of the property `hasAgreementPayment` from all the agreements and if an agreement is found where that value is 'false', then a `ValidationError` fact is created with the information needed.

```
rule "CV_PAYM Agreement must have at least one Payment"  
ruleflow-group "validate_agreementparam"  
when  
    ValidationOptions( businessValidation == true )  
    $a : Agreement( hasPayment == false )  
then  
    ValidationError fact0 = new ValidationError();  
    fact0.setDescription( "Agreement must have at least one  
        Payment" );  
    fact0.setSeverity( "error" );  
    fact0.setRowid( $a.rowID );  
    fact0.setRuleName( drools.getRule().getName() );  
    fact0.setFieldValue1( $a.agreementID );  
    fact0.setId( $a.agreementID );  
    insert( fact0 );
```

It would be best that first all the facts would be read to memory, but it is not possible because of the memory consumption. This is a way to bypass reading all the facts to the memory by using helper properties in those facts. It is also a way to make the

5 CONVERSION DATA PRE-VALIDATION TOOL

complicated rules simpler by dividing the rules and using the helper properties as an information storage between different validation rules.

5.6 Usage and error reporting

The pre-validation tool is a console-based program. It can be used in Windows and in Linux. The pre-validation tool program is delivered in a .zip package and it is ready to use after the package is decompressed to the desired folder. The .zip package contains all files that the program needs to function, except test material in the CSV form.

The pre-validation tool is configured with the *prevalidator.properties* file which is found in the */conf* folder. The properties file contains settings for a file path where the CSV files to be validated are found, which rule sets are active and a separator mark which is used to separate columns in the CSV files. Below is the context of the *prevalidator.properties* file.

```
prevalidator.filepath=c:\vallidator\testset\  
prevalidator.uniquecheck=true  
prevalidator.formatValidation=true  
prevalidator.businessValidation=true  
prevalidator.productParameterBasedRules=true  
prevalidator.productSpecificRules=true  
prevalidator.separator=,
```

The pre-validation tool is started via console. After the initialization, the pre-validation tool writes to console the folder path from where the CSV files are validated. Then the pre-validation tool writes to console that the validation is started and shows which CSV file is in validation. After the first file is validated and the validation of the next file begins, the pre-validation tool writes to console that the validation phase is changed to the next CSV file. If the CSV file is missing, pre-validation tool writes a warning message to console and changes the validation phase to the next file. After the last file is validated, the pre-validation tool writes error messages from all the errors found during validation to console. Also, the total number of errors are shown and a message that the validation has ended is shown.

Error logging

The pre-validation tool writes *prevalidator.log* file to the root folder of the program. The log file contains all the error and warning messages that the validation creates. The log file also contains a time stamp of when the validation started and when the validation ended and the total number of errors and the folder path where the validated files are.

5 CONVERSION DATA PRE-VALIDATION TOOL

The error message always contains *agreementID* which specifies the agreement that the error was referring to. The error message also contains the information about which fact was erroneous and therefore specifies the CSV file where the error is. The rowID where the invalid value was found is also specified in the error message. With this information it is quite easy to find a single incorrect property from a CSV file.

5.7 Performance of the pre-validation tool

The purpose of these tests is to see how long the validation takes with different amounts of agreements and with different settings. Also, tests to check how much the number of the validation rules affects to the validation times are done. The tests are done with Lenovo ThinkPad L430 laptop with 4Gb of memory. The total amount of the rules in the pre-validation tool is 1204 and the rules are divided into the following categories:

- collection rules: 642
- format validation rules: 461
- unique checking rules 40
- business validation rules: 61

There are five different sets of agreements used in this test. The agreements in the test material are taken from real conversion material and so the number of facts connected to the agreements are not identical because, for example, each agreement has a different amount of payments depending on the agreement's age and type. We can see this when comparing validation times in table 5.1 between the set with 15000 agreements and the set with 30000 agreements. There the number of agreements doubles but the total number of facts increases only with 61 percent and so the validation times are not doubled either. Each fact has 10 – 25 properties, the exact total number of properties is not counted. The number of agreements and the total number of facts in these test sets are:

- Set with 60000 agreements has total amount of 5117476 facts
- Set with 30000 agreements has total amount of 2418182 facts
- Set with 15000 agreements has total amount of 1495684 facts
- Set with 7500 agreements has total amount of 657999 facts
- Set with 600 agreements has total amount of 54723 facts

5 CONVERSION DATA PRE-VALIDATION TOOL

Table 5.1: Validation times of the pre-validation tool with different numbers of agreements and with different settings.

	60000 agreements	30000 agreements	15000 agreements	7500 agreements	600 agreements
All validations enabled	68min 21s	15min 43s	8min 2s	3min 25s	26s
business validations disabled	43 min 45s	13min 10s	6min 59s	3min 6s	23s
Unique checking disabled	40 min 23s	10min 50s	6min 27s	2min 58s	22s
Business validations and unique checking disabled	28min 7s	9min 29s	5min 53s	2min 42s	21s
All validations disabled	21min 36s	9min 0s	5min 29s	2min 30s	20s

From table 5.1, we can see that when all the validations are disabled, the time to read and convert the property value types rises quite straightforwardly when the amount of the agreements doubles. This trend continues also when only the format validation rules are enabled. However, when the business rules or the unique checking rules or both are enabled, validation time becomes four times greater when the number of agreements rises from 30000 to 60000. That is because the unique checking rules and many of the business rules have to compare facts and properties between each other and the amount of work done rises when the number of the agreements becomes higher.

Table 5.2: Validation times with business validation disabled and business validation rules removed.

	60000 agreements	30000 agreements	15000 agreements	7500 agreements	600 agreements
Business validations disabled	33 min 45s	13min 10s	6min 59s	3min 6s	23s
Business validation rules removed	30 min 22s	12min 28s	6min 58s	2min 56s	22s

5 CONVERSION DATA PRE-VALIDATION TOOL

In the first row of table 5.2 are the validation times that are made with the pre-validation tool with business validations disabled. The validation times in the second row are made with the pre-validation tool which has been modified so that the business validation rules are removed from the rule package of the pre-validation tool. The number of rules should not affect much to the validation times. When all 61 business rules are only disabled from the properties file, the rule engine must evaluate the 'boolean' condition “*businessValidation == true*”, but when the business rules are removed from the rule package, that evaluation is no longer done and the validation times are somewhat shorter.

Table 5.3: *Validation times with business validations and unique checking disabled and with business validation and unique checking rules removed.*

	60000 agreements	30000 agreements	15000 agreements	7500 agreements	600 agreements
Business validations and unique checking disabled	21min 7s	9min 29s	5min 53s	2min 42s	21s
Business validations and unique checking rules removed	19min 40s	8min 46s	5min 28s	2min 16s	20s

In the first row of table 5.3 are the validation times that are done with the pre-validation tool with business validations and unique checking disabled. The validation times in the second row are made with the pre-validation tool which has been modified so that the business validation rules and the unique checking rules are removed from the rule package of the pre-validation tool. These times confirm the conclusions which are made based on table 5.2; that simple boolean condition evaluations in the rules consume only a little time even when there are millions of validations done.

After these tests, we can see that the pre-validation tool implemented with a rule engine and using rules to do the validations is powerful and has good performance. The validation of 60000 agreements with a total number of 5117476 facts each with 10 - 25 properties using a total of 562 validation rules and 642 collection rules took 68 minutes and 31 seconds with a Lenovo ThinkPad L430 laptop with 4Gb of memory. Using a more powerful computer with more memory, the validation times would be much shorter. When the agreements have been converted from a legacy system to the CSV

5 CONVERSION DATA PRE-VALIDATION TOOL

form with the use of the pre-validation tool, possible errors are found quickly from the data. After all the errors have been found, changes to the collection of the data can be made. Or if there is a single agreement which is faulty, it is easy to find with the help of the error message created by the pre-validation tool. After the faulty agreement has been found, it can be sent to the business analysts for closer inspection.

6 CONCLUSIONS

The objective of this thesis was to study how the rule-based validation tool is implemented and to find out if the rule engine would be the best technology to implement such a program. This thesis shows how to make a fact model from the business requirements and how to make business rules based on that fact model. The goal was also to consider other ways that could be used in the validation of insurance policies.

As we can see from this thesis, the use of the rule engine in the validation of insurance policies is reasonable. The amount of insurance policies and the amount of data which each policy contains, gives great performance requirements to the pre-validation tool. Huge amounts of different validations make the usage of the rules very rational. The use of rules makes it easier for business analysts to make the validation rules. When making complex validations for insurance policies, those rules must usually be created by analysts who are familiar with the insurance business rather than the program developers who do not have such a deep knowledge about the insurance business. With these requirements, the use of some other technology than the rule technology would not make the pre-validation tool as good as it is now.

Performance of the rule engine is based on the Rete algorithm and the function of the Rete algorithm is presented in chapter 3.2. It is important for the developer of the system which uses the rule engine to understand how the Rete algorithm functions so the use of the rules will be optimal.

The creating of the rulebook is presented from the composing of the structured business vocabulary to the creation of the rules based on it. It is shown that the use of the rulebook is a good way to manage the rules and with the structured business vocabulary, the creation of the rules is effective and comprehensive.

The validation performance was measured with parts of real conversion data. The validation of 60000 agreements which had 5117476 facts related to those agreements and each fact had 10 – 25 properties, took 68 minutes and 21 seconds. With a more powerful computer than the laptop that was used in the test runs, the validation time would decrease significantly.

The people who are working with the rules and the rule engines should first be educated properly to the world of the rules to achieve an understanding of what the rules are and how they work. However, learning how the rule engine works and how to use it in an effective way is quite fast. When I started to make this rule-based pre-validation tool in the beginning of the year 2013, I did not have any knowledge about the rules or the rule engines. After six months of work, the pre-validation tool was finished and my knowledge of the rules was good enough to do many kinds of solutions which use the

6 CONCLUSIONS

rules. Use of the rule engine does not require as much coding skills from the developer, as other methods would, to implement a validation tool.

The use of the rules is reasonable when dealing with a large number of complex insurance policies containing lots of information. Now we can see that rules could be used for other purposes than validation when working with conversion of insurance policies. Based on this thesis, one place where the rules could be used in the conversion process is the collection of the data from the legacy system. The business analysts can specify which pieces of information from insurance policies from the legacy system would be placed in the column of the specified CSV file and the implementation of this could be rule-based.

REFERENCES

Alhonsuo Sampo, Nilsen Anne, Nousiainen Satu, Pellikka Tuula, Sundberg Sirpa (2012), Finanssitoiminnan käsikirja. Second edititon. Bookwell Oy. 329 p.

Bali Michael. (2009), Drools Jboss Rules 5.0 Developer's Guide [eBook]. Packt Publishing Ltd. 301 p.

Browne Paul. (2009), Jboss Drools Business Rules [eBook]. Packt Publishing Ltd. 285 p.

Pentikäinen Teivo and Rantala Jukka (1995), Vakuutusoppi Gummerus Kirjapaino Oy. 527 p.

ProfitSoftware. About Us. [WWW] (cited: 20.6.2013.), Available at: www.profitsoftware.com.

Ross Ronald G. (2009) [a], Business Rule Concepts. Third edition. Business Rule Solutions, LLC. 158 p.

Ross Ronald G. (2009) [b], RuleSpeak Sentence Forms, specifying Natural-Language Business Rules in English. Version 2.2. Business Rule Solutions, LLC. 10 p.

Ross Ronald G. with Lam Gladys S.W. (2011), Building Business Solutions, Business Analysis with Business Rules. First edition. Business Rule Solutions, LLC. 304 p.

The JBoss Drools Team. (2012) [a], Drools Expert User Guide. [WWW] Version 5.5.0.final. 358 p. Available at <http://docs.jboss.org/drools/release/5.5.0.Final/drools-expert-docs/pdf/drools-expert-docs.pdf>

The JBoss Drools Team. (2012) [b], Drools Introduction and General User Guide. [WWW] Version 5.5.0.final. 190 p. Available at <http://docs.jboss.org/drools/release/5.5.0.Final/droolsjbpm-introduction-docs/pdf/droolsjbpm-introduction-docs.pdf>

The JBoss Drools Team. (2012) [c], Guvnor User Guide, for users and administrators of guvnor. [WWW] Version 5.5.0.final. 222 p. Available at <http://docs.jboss.org/drools/release/5.5.0.Final/drools-guvnor-docs/pdf/guvnor-docs.pdf>