BUSINESS RULES MANAGEMENT SYSTEMS

Cliffred van Velzen

Table of Contents

1. INTRODUCTION

Businesses run on policies and procedures. These are made actionable by business rules. Business rules may apply to people, processes, or corporate behaviour. A business might state that customers of a car insurance company get a ten percent discount after being subscribed for more than five years, or that a bank loan is denied when the customer's salary is below some threshold. Examples of business rules are endless. They are ubiquitous and many critical business applications rely on an enormous amount of business rules.

1.1. Business rule management systems

Rule-based systems have been around for many decades (Hayes-Roth, 1985). The rules used by these systems can be highly dynamic, e.g. rules are affected by changing laws, policies and other sources. Therefore they change more often than the applications that run based upon them (Arsanjani, 2001). This makes it ineffective to manage business rules within an application's source code. In situations where many of the rules behind an application can be subject to change, a business rule management system (BRMS) can be a very suitable solution.

The key principle behind business rules management systems is the separation of application logic and business rules. And these systems should provide the means to easily manage the business rules. Every business rule management system also has a rule engine. A rule engine can be viewed as a sophisticated interpreter of if-then statements (Mahmoud, 2006). It is a software component that executes (business) rules that have been externalized from the application source code into a separate rule base. The externalization of business rules is obviously a vital step in the process of implementing a business rules management system and shall therefore also be covered in this thesis.

This separation of rules and application logic combined with rule management functionality should allow rules to be formulated and changed without resorting to IT specialists, hence allowing applications to be more agile and robust. This way business applications are built to merely communicate with the rule engine, which deals with the separately managed rules.

A lot has been written about the value of business rules in the recent years, however, few scientific literature deals specifically with BRMS. As mentioned before, rule-based systems are widely used in all business critical applications and the business rules behind them are generally vital for an organization's survival (Chisholm & Ross, 2007). Business rules management systems aim to help businesses externalizing these business rules and make them more manageable. Practitioners in this field can choose from a number of BRMS products today. However, each product has a different range of features and advocates a different approach to development. They also differ in aspects such as performance, underlying algorithms, target market and ease of use. And although multiple systems are available, only a few vendors have well-established business rules management systems.

The process of selecting and implementing the appropriate system is a complex one, with many aspects to consider (Rymer & Gualtieri, 2008). But even before diving into this selection process one should first establish that a business rules management system is a sensible solution.

This thesis aims to clarify the suitability of business rules management systems and serve as a guide in the selection and implementation process. Furthermore, this thesis tries to provide insights in how business rules management systems relate to traditional knowledge systems. Also, attention will be paid to related technologies and how business rules management systems fit into the picture of these modern developments. Service oriented architecture is one example of such a technology that is often coupled with business rules management systems (Taylor, 2005).

This thesis is carried out within CGI, formerly know as Logica. CGI is a multinational IT consulting, systems integration, outsourcing and solutions company. CGI is interested in exploring the role BRMS's could have in outsourcing solutions for their clients.

1.2.Research questions

The problem outlined above results in the following research question:
What guidelines are there to assist in the process of selecting and implementing a business rules management system?

This question can be further divided in the following sub questions:

1. *What comprises a business rule?*
2. *How do business rules management systems relate to traditional knowledge systems?*
3. *How do you recognize and extract (externalize) the business rules to be implemented in the business rules management system?*
4. *When is a business rules management system suitable?*
5. *How do business rules management systems fit in the context of other related developments?*
6. *What are the requirements of a business rules management system in a given situation?*
7. *What does the process of implementing a business rules management system typically look like?*

The outcome of the sub questions will serve as a basis for formulating the answer to the main research question.

1.3.Methodology

During the course of this thesis project, several phases can be distinguished. Each of these phases is mentioned below. The first phase consists of a literature study. Existing literature regarding business rules, rule based systems, and knowledge systems will be studied in order to answer the first four sub questions. Also, a literature study will be performed to discover related developments and examine their connection with business rules management systems. The outcome will serve as an answer to the fifth sub question.

The second phase consists of interviewing CGI employees who have experience with projects where a BRMS was used. Also CGI internal documents and message boards are studied. The goal of these interviews and document analysis is to give insights in the suitability, requirements and implementation process of a BRMS. Together these will be used as a basis to answer question six and seven.

In order to gain more insight in BRMS's a typical use case will be implemented in several market leading BRMS's. These hands-on experiences – together with the insights gained from the interviews and analysis in the previous phase – will serve to answer question five, six and seven.

The remainder of this thesis is structured as follows: The next chapter is a more thorough introduction of business rules. Chapter 3 provides a detailed theoretical discussion on BRMS and in chapter 4 BRMS's will be put into context of related developments. After this theoretical background chapter 5 discusses BRMS's in practice. A typical use case will be implemented in a few of the main vendor's systems in chapter 6. Chapter 7 discusses the results of this thesis in terms of requirements and guidelines. Finally, chapter 8 will present the concluding remarks and future work.

2. BUSINESS RULES

This thesis revolves around Business Rules Management Systems (BRMS), before diving into the details of these systems it is important to get a clear idea of the role of business rules. Therefore this chapter starts with clarifying what business rules really are and why they are important.

2.1. What are business rules?

Only two pieces of luggage per passenger is allowed and *orders of $30 and over have free shipping* are perfect examples of business rules. It is easy to think of numerous examples and therefore it should not be difficult to get a grasp of what is meant by the term *business rule*. However, it means different things to different people. The term can be used so widely that its exact meaning starts to fade.

The term *business rule* first appeared in 1984 (Appleton, 1984). Appleton defines a business rule as 'An explicit statement of a constraint that exists within a business's ontology.' Since then many authors have redefined the term. Like Appleton's, most early definitions are formulated from a system's perspective, i.e. conflate business rules with database constraints. Other definitions take a purely business point of view. Although there is no industry standard definition for the term business rule, or even for rule (von Halle, 2001), the most commonly cited definition is the one put forth by the Business Rules Group (BRG) (Hay & Healy, 2000): 'A business rule is a statement that defines or constraints some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business.' Although this definition is still very broad, it should suffice in establishing a clearer vision on the subject. The problem with this definition in the context of this research is that it does not cover how rules should be expressed. Therefore a definition that also takes the system's perspective into account is a better fit. The definition used throughout this research will be the definition proposed by Graham (Graham, 2006):

A business rule is a compact, atomic, well-formed, declarative statement about an aspect of a business that can be expressed in terms that can be directly related to the business and its collaborators, using simple unambiguous language that is accessible to all interested parties: business owner, business analyst, technical architect, customer, and so on. This simple language may include domain-specific jargon.

There are a few terms in this definition that need further clarification. **Atomic** here means that a rule cannot be broken down without losing important information. **Declarative** is the opposite of procedural. In a procedural rule language the order of execution of the rules matters; in a declarative language the outcome is the same whatever execution order is selected (Graham, 2006). Date (2000) makes a similar point, stating that rules should convey 'what not how'. It should be clear that because of the declarative principle business rules do *not* describe business processes; instead they constrain what processes are possible.

CLASSIFYING BUSINESS RULES

Different types of statements can be considered as a business rule according to the definition presented above. Many authors have made distinctions between different types of rules. Graham (2006) distinguishes business rules on a high level, identifying two main categories: assertions and rules. Assertions (or facts) are statements about properties or value of entities and have the form: 'A is X' or 'P is TRUE'. Rules are directives or constraints and have the form: 'IF A THEN X', where X can be an assertion or action.

Other authors are more specific, they recognize more categories of business rules and designed complete classification schemes (Ross, 2001; Date, 2000; von Halle, 2001). Most of these schemes show overlap in categories, they mainly differ in their level of granularity and focus. The Business Rule Group (BRG) proposed a classification scheme somewhere in the middle with regard to granularity. They distinguish three types of business rules:

1. **Structural assertion,** a defined concept or a statement of a fact that expresses some aspect of the structure of an enterprise. This encompasses both terms and the facts assembled from these terms.
2. **Action assertion,** a statement of a constraint or condition that limits or controls the actions of the enterprise. It tests a certain condition.
3. **Derivation,** a statement of knowledge that is derived from other knowledge in the business.

A few examples of business rules of different types can be found in Table 1

Type	Business rule
Structural assertions	
Term	Car
Fact	Customer owns a car
Action assertions	
Integrity constraint	Customer's age must be at least 18
Authorization	Only insurance agents may sell insurances
Condition	If the stock level is below 50, then replenish
Derivations	
Calculation	Profit equals revenue minus costs
Inference	If the stock level is below 10, then the replenish need is "high"

Table 1 Examples of business rules

As can be seen in the table above, these three main categories can each be subdivided further. Structural assertions are either *terms* or *facts*. Terms are the core entities of a business. In an insurance company for example, terms could be customer, policy or car (for car insurances). Facts indicate relations between two or more terms.

Using a car insurance company as an example again, a fact could be ‘a customer owns a car’. Structural assertions are somewhat counterintuitive as a type of business rule as they do not really match the concept most people have about rules. They are, however, the core of a data model.

Whereas structural assertions describe possibilities, action assertions impose constraints. These can be divided into *integrity constraints*, *authorizations* and *conditions*. An integrity constraint is an assertion that must always be true to keep a valid state. It makes sure that any action which would result in a false value is not possible. An authorization defines a specific right or privilege with respect to one or more constructs. It has the form of: (Only) x may do y, where x is typically a user and y is an action. A condition is mainly an “if…then…” statement. It tests if something is true, and if so, another business rule will apply. Action assertions probably resemble the general idea people have when they think of rules the most.

Derivations are either *mathematical calculations* or *inferences*. A mathematical calculation produces new (derived) facts on the basis of mathematical algorithms. Inferences also produce new facts, but these are based on logic: induction or deduction. The example in Table 1 shows that for a fictional retailer the ‘replenish need’ is ‘high’ when the stock is below a certain threshold. The value of ‘replenish need’ could be used as input for other rules, especially conditions. A possible condition could be: if replenish need is high, then place a new order.

2.2. Why should we care?

Just as humans, businesses make decisions based on facts, and in order for these decisions to be of high quality; the quality of the reasoning behind them should also be high. High quality rules and facts are therefore a necessity for decision makers and systems. Only when business people and information systems have access to the right business rules at the right time they are able to make informed choices in a timely manner. This way business rules serve as the guidance system that influences the collective behaviour of an organization's people and information systems (von Halle, 2001). They also reflect the way in which a business implements its competitive strategy and complies to legislation (von Halle & Goldberg, 2006).

Too often, however, business rules are inaccessible, or even unknown by many people within an organisation. For many business critical applications, the business rules by which they operate are buried inside of the source code, for which there is little or no documentation. One can imagine that it is undesirable that business decisions are made based on rules hidden from the actual decision makers. This could result in people making assumptions about those rules, which could be incorrect or inconsistent. Such assumptions tend to lead to disorganized behaviour, which is not effectively focused on common objectives, and incapable of reacting to change on short notice, which is very important for a business today's competitive markets. After all, humans are very capable of assimilating new knowledge, correcting old knowledge, applying it to behaviour, and evaluating the results. Humans have a great affinity for learning and businesses today need to capitalize on that ability to remain healthy and competitive (von Halle, 2001).

A key statistic relevant to the failure of IT in the modern world is the cost of maintenance. It is widely estimated that well over 90% of IT costs are attributable to maintenance of existing systems rather than to their development (Graham, 2005). An approach to keep maintenance costs low is to have loose coupling, so that changes in one place do not propagate to other places. But this benefit does not apply to business rules when they are scattered across the application.

Because of the pace of change today, now, more than ever, businesses need to learn quickly and be able to redefine their automated procedures. According to von Halle (von Halle, 2001) this means businesses need systems in which the rules are: separated from other components so everyone knows that they exist; externalized so everyone knows what the rules are; traceable to their origins and their implementations so everyone knows where the rules come from; and deliberately positioned for change so everyone knows how to improve them.

2.3.Business Rule Approach

Awareness of the importance of the role of business rules has led to a paradigm shift in business-system design and development, which is often referred to as the business rule approach. The business rules approach is a development methodology where rules are in a form that is used by, but does not have to be embedded in, business process management systems (Ross, 2003).

Von Halle (von Halle, 2001) identifies four important principles of the business rule approach that were already briefly mentioned in the previous section, they are:

Separate rules, this means that you separate the rules from all other aspects of the requirements and in the system itself. You do this primarily so you can reuse rules and be able to revise, remove them, or add new ones in a timely manner. That is, if you manage them as an individual asset, you can apply techniques specific for optimizing them and you can change rules independently of other aspects of the system.

Trace rules, this means that you maintain a connection from each rule in two directions. The first direction is toward its origins. A rule has origins in aspects of the business's motivation, such as business missions, goals, objectives, strategies, tactics, and policies. The second direction to trace is the rule's implementation. You do this so that you can assess the impact of rule changes. That is, you record all of the places where the rule is executed.

Externalize rules, this means that you express a rule in a format understandable to nontechnical, business audiences and that you make the rule available to these audiences. You do this so that everyone can access the rules and optimize them if necessary. Doing this allows business leaders to inspect rules and consider challenging them or measuring their effectiveness from time to time.

Position rules for change, this means that you always position a rule for change precisely because you expect rules to change as a regular course of doing business. You do this so that rule changes happen easily and quickly. It means being able to conduct impact analysis when a rule needs to change, such that you know which business events, decisions, and organizations are impacted by the change. It also means that the data foundation supporting the rule is built in a flexible manner such that rule changes should not require expensive and time-consuming database changes.

If systems do not support these principles, businesses lose memory of its business rules and therefore lose control in effecting business change. In many cases rules remain lost, unknown, inconsistent, inadequate, or simply resistant to change. If these rules are crucial to business operations they form a hidden liability. Unmanaged rules result in lost time-to-market, regulation violations, and customer dissatisfaction (von Halle & Goldberg, 2006).

2.4.Concluding remarks

Although most people do have associations when they hear the term *business rule*, there is no one clear definition. This thesis uses a definition that states that business rules are atomic and declarative. Rules can be divided into different categories and examples of each one are given. Organizations are driven by their business rules, but often these rules are buried deep within their business applications. This causes rules to be hidden from the actual decision makers and leads to high IT maintenance costs. It is therefore beneficial to separate business rules from application logic. The business rule approach is a methodology with the following principles: separate rules, trace rules, externalize rules and position rules for change.

3. BUSINESS RULES MANAGEMENT SYSTEMS

In this chapter business rule management systems will be covered in detail. This entails their history, what parts they are made up of. Also, related concepts, as rule extraction and rule storage will be covered. After reading this chapter you should know what business rules management systems are and have clear idea of the aspects to consider when implementing a BRMS.

The previous chapter has shown the importance of carefully managing your business rules and how the business rule approach can help you achieve that. When the business rules approach results in the automation of rules in systems, often a BRMS is deployed. Applying the business rules approach with a BRMS builds better, changeable systems faster than any previous approach (von Halle & Goldberg, 2006).

Before diving directly into the topic of business rules management systems, this section starts off with an outline of technologies that can be regarded as predecessors of BRMS's.

3.1.History

The first talk of business rules management emerged from discussions in the database community as long ago as the late 1980s (Graham, 2006). However, its roots may go back even further. As early as 1972 a project called MYCIN started. MYCIN (Shortliffe, 1976) was an expert system that could, with some success, diagnose infectious diseases of the blood. You can in no way say that MYCIN was a BRMS; its rules were hard coded and it covered the fairly esoteric domain of medicine. A few years later a new system, based on MYCIN, was introduced that shows several significant commonalities with BRMS's. EMYCIN (van Melle, Shortliffe, & Buchanan, 1984) was an 'empty' MYCIN in the sense that the rules were taken out and it had two significant mechanisms. First, rules on any suitable domain (including business domains) could be typed in and run under the control of the same logic used by MYCIN. Secondly, an EMYCIN application could be asked to explain its conclusions when asked ‘How?’ or ‘Why?’ Although this system does not resemble a modern BRMS, it is important to notice that EMYCIN separated business rules from both data and the control logic that enabled conclusions to be reached, and these are key principles of modern business rules management systems. Also, the rules were written entirely declarative, another important principle of the business rule approach.

The evolution towards business rules management systems can be traced back to several sources. The most prominent are the following three: artificial intelligence, data modelling and business process engineering. Advances in business modelling created by artificial intelligence, data modelling and process management have made lasting contributions to the global digital infrastructure (Ould, 2005).

KNOWLEDGE SYSTEMS

In the field of artificial intelligence, a rich history has been built up on knowledge systems and expert systems over the last decades. In today’s so called information society, there is an emerging consensus that economies are increasingly driven by information and knowledge (Powell & Snellman, 2004; Huber, 2004). Despite the importance of knowledge, it is an abstract and intangible concept. The meaning of knowledge is often clarified by relating it to the concepts of data and information. These concepts are often used interchangeably in everyday discourse, but there are distinct differences. Boisot and Canals (2004) use encryption to make the distinction between data and information very clear. Encryption, by developing algorithms that bury information deeply in data, provides “the lock and keys” of the information age (Singh, 1999, p. 293). Thus while the data itself can be made “public” and hence freely available, only those in possession of the “key” are in a position to extract information from it (Singh, 1999). Information can be thought of as data with meaning added to it. There is also a distinction between information and knowledge. Consider the following example, decreasing sales indicate that a certain response is required. Here, the decreasing of sales is information, but applying this information in a specific context, with an attached purpose or action makes it knowledge.

Humans possess knowledge, but also computer systems are able store knowledge and reason with it, these systems are called knowledge systems. The term knowledge system is often used interchangeably with the term expert system. However, it should be noted that an expert system should be regarded as a special type of knowledge system. According to Schreiber et al. (1999) a knowledge system can be distinguished from a normal software system by the distinctive property of having some explicit representation of the knowledge that is included in the system.

An expert system, on the other hand, can be described as a computer program designed to model the problem-solving ability of a human expert (Durkin, 1994). Lucas and van der Gaag (1991) propose a similar, but slightly more detailed definition: a system capable of offering solutions to specific problems in a given domain or able to give advice, both in a way and at a level comparable to that of experts in the field. Referring to the earlier statement that an expert system is a special kind of a knowledge system, an expert system can be defined as a knowledge system that is capable of emulating a human expert in a specific domain.

The remainder of this section is used to discuss the characteristics of the early expert systems that were built in the 1970's and how these systems have developed over time. This will eventually show that business rules management systems are not a completely new phenomenon, but can in many ways be regarded as one of the results of the development of expert systems in the last decades.

The first expert systems were the result of an attempt to create systems that could solve complicated decision-making problems. Knowledge was gathered from experts and written resources and stored in the form of production rules in order to make this knowledge interpretable for a computer. The process of gathering knowledge in discussed in more detail in section 3.5. The use of production rules as the means to store knowledge led to the term rule-based expert system. The foundation of the modern day rule-based expert systems can be traced back to the early 1970's, when Newell and Simon from Carnegie-Mellon University proposed a production system model (Newell & Simon, 1972). The production system model is based on the idea that humans solve problems by applying their knowledge (expressed as production rules) to a given problem represented by problem-specific information. In the long-term memory, the production rules are stored, whereas the problem specific information (facts) is stored in the short-term memory. The system then looks for matches between the facts available in the short-term memory and conditions on production rules in long-term memory. Whenever a match is found, that production rule fires, modifying the contents of short-term memory. This is an on-going process. The production system model is shown in Figure 1.

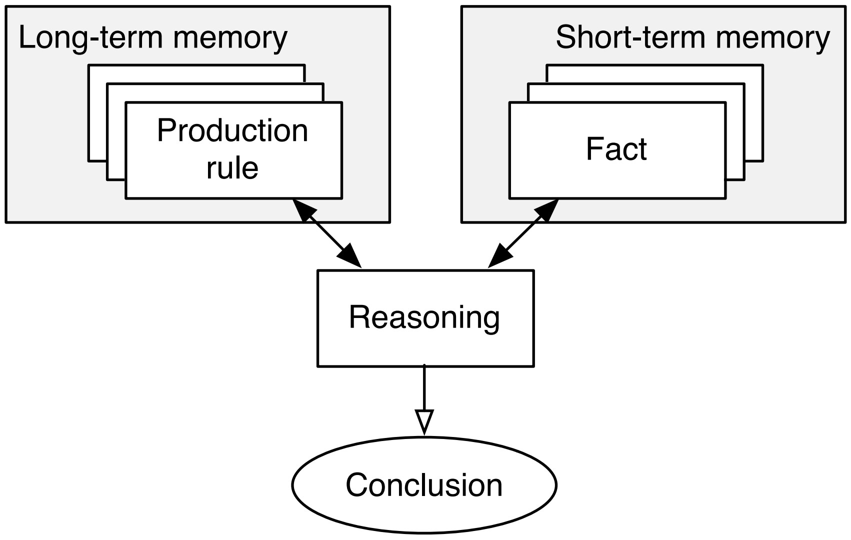


Figure 1 Production system

The first systems built on this principle required the user to interact with the system through a terminal, in which the system asks questions and the user provides the answers as to simulate a dialogue with an expert. Whenever the user supplies the system with an answer, the system's inference engine determines whether extra information is required from the user, or a conclusion can be drawn. A typical architecture of such a system can be found in Figure 2. This figure shows that a rule-based expert system consists of five components: the knowledge base, the database, the inference engine, the explanation facilities, and the user interface. Note that throughout the literature slight variations of this model can be found. Not all authors distinguish the knowledge base from the database, or the explanation facilities from the inference engine. Also, the user interface is sometimes not regarded as being part of the system.

The *knowledge base* is where the production rules are stored in the form of IF (condition) THEN (action) statements. When the condition part of the rule is found to be true, the rule is said to be fired and the action part is executed. The knowledge base corresponds to the long-term memory of the production model.

The *database* corresponds with the short-term memory of the production model and this is where the domain specific facts are stored which are matched against the condition parts of the rules in knowledge base.

The *inference engine* is responsible for the reasoning of the system by linking the facts in the database to the rules in the knowledge base.

The *explanation facilities* are there to provide the user the possibility to show how the conclusion was drawn (which rules were fired) and why specific input is needed.

The *user interface* provides the communication between the user and the system.

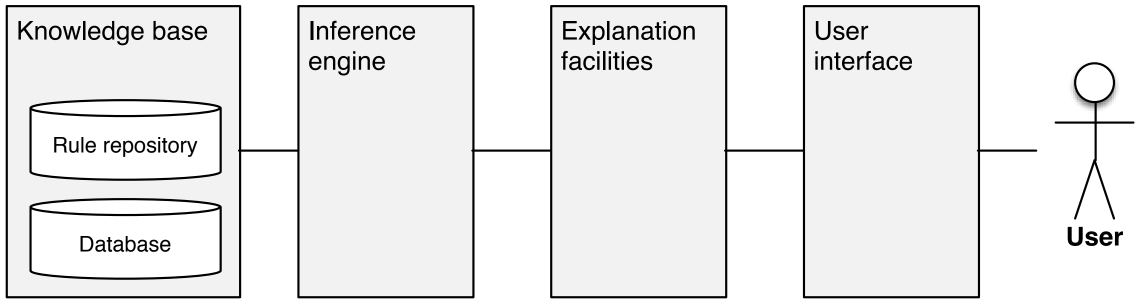


Figure 2 Typical architecture of a rule-based expert system

During the 1980's several companies were built to develop expert systems and the market quickly became saturated. This led to a shift towards the exploration of business rules problems. Some of the BRMS's of today started their life as an expert system.

After the initial boom of interest and some commercial successes, interest in the technology slowly faded and during a large part of the 1990's expert systems were widely regarded as not being able to deliver on their promise and the term was to be avoided by anyone in sales or marketing (Graham, 2006).

Despite this negative image, interest in the academic world remained the same and advancements were made, most notably expert system shells and hybrid expert systems.

An expert system shell is basically an expert system with the knowledge removed from it. Therefore the user is only responsible for adding the rules to system, i.e. the knowledge, and providing the relevant data, i.e. the facts. Because the user is in charge of the knowledge on which the system operates, the user decides in what domain the system operates. An expert system shell in itself is domain independent. The earlier mentioned system EMYCIN (van Melle, Shortliffe, & Buchanan, 1984) is widely considered to be the first expert system shell.

Hybrid expert systems differ from early expert systems by having multiple forms of knowledge representation. Hybrid systems were developed because it was found that knowledge representation by production rules is not always achievable or sufficient. Especially when the domain knowledge is ill-conditioned, poorly structured, or sparse, rule-based systems tend not to perform well (Dlodlo, Hunter, Cele, Metelerkamp, & Bot, 2007). Forms of knowledge representation often seen in hybrid systems are case-based and frame-based reasoning, neural networks and object-oriented models. The components in hybrid expert systems should be highly complementary. That is, the weaknesses of one method in a specific situation are compensated for by the other and vice versa. The use of hybrid systems has impact on the inference engine, as it must be able to connect the different forms of representation.

Another field within artificial intelligence, data mining, can also be regarded as an originating technology for BRMS, however far more indirectly. Data mining is the detection of interesting patterns in databases that are useful in decision making (Bose & Mahapatra, 2001). Data mining methods are used to (automatically) handle large numbers of objects. It typically involves the following types of tasks: clustering, classification, regression, and association rule learning (Fayyad, Piatetsky-Shapiro, & Smyth, 1996). There are also data mining methods that help to visualize the data. Some rule systems are able to provide a graphical representation of rules (Figure 3) and its developments can often be traced back to the work in this field (Debevoise, 2010).

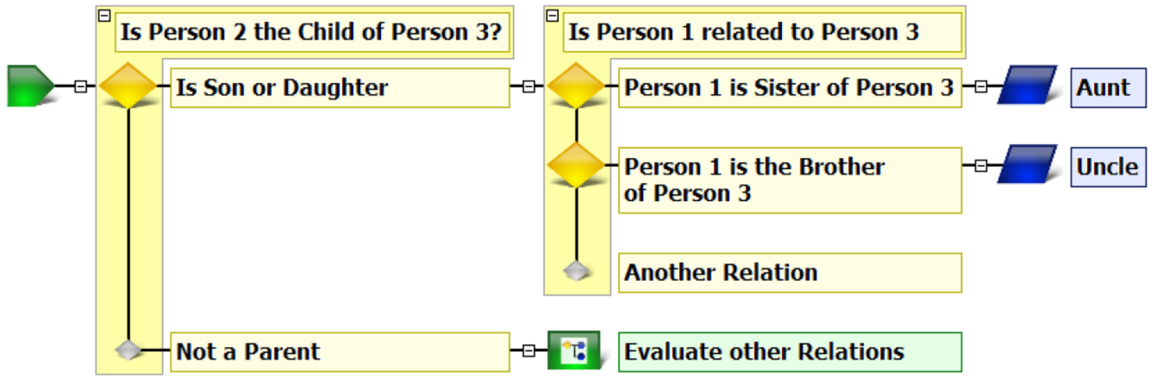

Is Person 2 the Child of Person 3?
Is Son or Daughter
Not a Parent
Is Person 1 related to Person 3
Person 1 is Sister of Person 3
Person 1 is the Brother of Person 3
Another Relation
Aunt
Uncle
Evaluate other Relations

Figure 3 Visual representation of rules in Visual Rules Modeler. The green icon in the top left corner defines the starting point. Yellow diamonds define a decision. And a blue parallelogram defines a conclusion or action.

DATA MODELLING

As the use of relational databases became more and more widespread during the 1980's en 1990's, interest in business rules grew within the data modeling community, especially with regard to requirements modeling and design. This ultimately led to the forming of the, in chapter 2 mentioned, Business Rules Group. Their goal was to provide a basis for engineering new systems based on formal definitions of business rules (von Halle, 2001). The group has been very influential in the growing awareness of the importance of business rules. One of the most compelling arguments for the importance of business rules from the data modeling world is incorruptible databases. Date (2000) argues that business rules should be used to ensure that data can not be corrupted by not allowing updates to the database that violate the business rules.

PROCESS MANAGEMENT

Business Process Management (BPM) is the third source that can be regarded as very influential to the growing need for of Business Rules Management Systems. Smith and Fingar (2002) were responsible for the recognition of the role of business rules in the field of BPM (Debevoise, 2010). Later, Debevoise (2007) presented methods for combining BPM and business rules with Business Intelligence. Analysts are predicting that this combination will become more prevalent and will have a significantly greater business impact when used together (Evelson, Teubner, & Rymer, 2008).

3.2.Business rules management systems

While the developments in the fields of data modeling and process management have been very influential on the adoption of business rules, from a technical point of view knowledge systems are definitely the predecessors. As these systems are very similar in architecture, the question 'in what way are they different?' may arise. First, consider the difference from an architectural point of view. Figure 4 shows the typical architecture of a business rules management system. The core of the architecture is very similar to that of an expert system. They both share the existence of a knowledge base, existing of long-term and short-term part and they both have an inference engine, which is called rule engine in BRMSs, but is basically the same component. In comparison with Figure 2, there are two important differences. First is the presence of a development environment and maintenance facilities; these can be the same. The maintenance facility is a key part of a BRMS, as it enables non-technical business people to update the rule base. The second difference is that the user interface is replaced by a communications interface. The end user typically does not interact with a BRMS itself, but with an application that communicates with the system, e.g. a website. Some BRMSs do provide a user interface, but this is mainly used for testing purposes. The communications interface can be implemented in several ways. One way of doing this is with an application programming interface (API). This allows the developer to directly communicate with the system from his programming environment. A far more common way, however, is the use of a web service. Web services are the building blocks of most service orient architectures and will be discussed in a later chapter.

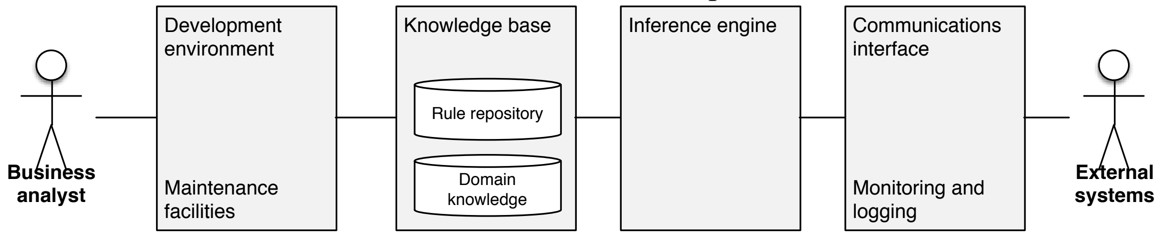


Figure 4 Architecture of a BRMS

Aside from the architectural differences, some authors also distinguish other differences. Von Halle (2001) lists a total of five characteristics of experts systems that distinguishes them from BRMSs. Expert systems

1. can handle uncertainty.
2. are used to address very complex decision-making activities in esoteric domains.
3. include an inference engine.
4. usually consist of inference rules. One decision may require the execution of hundreds or thousands of inference rules.
5. usually do not update the database during rule execution.

As can be seen from the third point, Halle claims a BRMS (typically) does not include an inference engine. Today, nearly all BRMSs include an inference (rules) engine and many even promote their engine’s speed as a selling point. Graham (2006) regards the third point not a valid difference and only agrees with the first point. He argues that the other points are often true in practice, but do not mark an inherent difference between the systems and points to examples from his own experience. Graham concludes that there are no significant technical difference between the core of both systems, aside from the uncertainty management facilities and application domain.

The abovementioned differences are only in regard to the more traditional expert systems. But when BRMSs are compared with hybrid systems, another important difference is apparent. As the name suggests, BRMSs work on rules and do not offer other means of knowledge representation. And as not all knowledge can be represented with rules (Collins, 1990), this limits the domains in which BRMSs are applicable.

The section above should provide a clear understanding of what a business rules management system is. However, we still have not given a clear definition for it. As few authors directly consider BRMSs, it is hard to find definitions in existing literature. In this thesis, the following definition is used, largely based on the definition of Thirumaran et al. (2010):

A Business Rule Management System is a knowledge system – consisting of at least a rule development environment, a rule repository, a rule engine and a communication interface – used to define, deploy, execute, monitor and maintain the variety and complexity of decision logic that is used by operational systems within an organization or enterprise. This logic, also referred to as business rules, includes policies, requirements, and conditional statements that are used to determine the tactical actions that take place in applications and systems.

3.3.Suitability

Now that it has been made clear what a BRMS exactly is, the question when to use one arises. Section 2.2 showed the problems that arise when business rules are buried inside application source code and discussed the advantages of separating of business logic and application. However, separating your business logic increases the complexity of your IT architecture and the implementation and maintenance of a BRMS is costly and requires specific knowledge. It is therefore important to be aware of the factors that indicate the need for a BRMS. The following situations signal that a BRMS is a suitable solution (Graham, 2006) (von Halle, 2001).

- The business logic within a system has become increasingly complex to the point where only a very few people understand what the system is actually doing, but the systems still needs to be updated on a regular basis.
- Current software systems are in continuous development and changes need to be released soon, but even relatively small changes take a long time.
- Many software processes use an extensive set of rules to reach a decision. And these processes should only reach the correct decision, but also need to be able to show how and why this decision was reached.
- Due to competitive pressure the policies and rules carried out by automated processes are susceptible to rapid change.
- The business is part of an industry that is heavily influenced by external imposed regulations, e.g. finance or public sector. Therefore rules will change outside the control of the business.
- Many applications across the whole enterprise make use of largely the same business rules.

A closer look at these situations reveals that there is small set of properties of which at least one of them applies:

- The amount of business rules is enormous, or the rules are complex.
- The business rules change often.
- Validity of business rules needs to be proven.
- The business rules should be shared across applications.

These properties have in common that they all lead to maintenance problems. So in general one can say that costly maintenance that is for an important part caused by business rules being decentralized is a great indicator that a BRMS is a suitable solution.

3.4.Rule engines

In rule-based expert systems and in BRMSs the knowledge is represented by a set of IF-THEN production rules stored in the rule repository. The short-term memory is represented by a set of facts about the current situation. The inference engine – or rule engine as we call it in regard to BRMSs – compares each rule in the rule repository to the facts in the database. Every time the IF part of a rule matches against the facts in the database, the rules is said to fire and the THEN part is executed, which could mean the database is updated, see Figure 5.

As mentioned earlier, a BRMS should be able to explain how it reached a conclusion. This is done by using the inference chain. Every time a rule fires it is added to the inference chain, which ultimately consists of an ordered list of the rules that have fired to reach a certain conclusion.

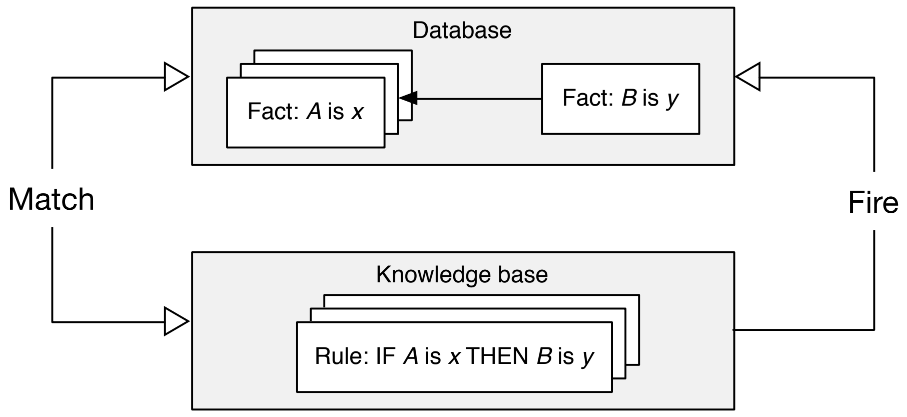


Figure 5 The cycle of a rule engine

As stated earlier, business rules must be formulated declaratively. This means the business rules do not require a specific order in which they are evaluated and chained together. It is up to the rule engine to determine when each rule should be evaluated.

There are two principle strategies to do this: forward chaining and backward chaining (Hayes-Roth, 1985).

FORWARD CHAINING

Forward chaining is a data-driven approach. The reasoning starts from the known facts and continues to check the facts as the database is updated by fired rules until none of the remaining rules fire. Consider the following rules:

Rule 1:	A & Z	->	Y
Rule 2:	B & C & X	->	Z
Rule 3:	D	->	X
Rule 4:	E	->	W
Rule 5:	W & V	->	U

The part before the **->** symbol represents the IF part of the rule, the part after represents the THEN part. The letters represent the conditions or facts. For example, rule 1 means: *If* A and Z are both true, *then* Y is true. Assume that these five rules are the entire rule base. Suppose the current facts known to the system are that A, B, C, D and E are true. The most basic way a rule engine could approach this is by cycling through the rules from top to bottom and fire the rules that match the facts in the database, each rule will only be fired once.

In the first cycle only rule 3 and rule 4 match the initial facts in the database and will fire. Rule 3 fires because D is in the database and after this has fired X is added to the database. When rule 4 fires, W is added to the database.

In the second cycle rule 2 is fired and Z is added to the database. Therefore, rule 1 will fire in the third cycle and Y is added to the database. At this moment the database consists of the following facts: A, B, C, D, E, W, X, Y, Z. This means no rules will fire in the fourth cycle and the process stops.

At the end of the process the inference chain looks as follows: rule 3, rule 4, rule 2 and rule 1. There are more tactics that can be used to determine when to stop the cycle. In this example we stopped after no more changes would be made to the database, i.e. no more rules fire. Other tactics would be to stop after a single rule fires, to stop after one cycle, or stop after a specific facts has been proven (Graham, 2006).

By using forward chaining, systems can draw new conclusions from existing data and add these conclusions to the database. This approach is useful when you know all the initial facts, but do not have a clear idea of what the conclusion might be or what you want to know (Cawsey, 1998). If you do have an idea of a possible conclusion or you want to infer a specific fact, the forward chaining inference mechanism can be very inefficient. Looking at our example, suppose you were only interested in the value of Z. Only rule 2 and rule 3 are necessary to fire in order to conclude Z is indeed true. But as we saw, rule 1 and rule 4 were also fire. When you have a very large rule base this could mean that hundreds or thousands of rules would be fired unnecessarily. In these cases a backward chaining mechanism is more appropriate.

BACKWARD CHAINING

Backward chaining is a goal-driven approach. Using this approach, the system starts with a specific fact in which it is interested in and the rule engine tries to prove or disprove only that fact, without evaluating unrelated rules. The rule engine starts with searching the rule base for rules that can potentially prove the specific fact we are interested in. These are the rules that have this fact in their THEN part. The system then evaluates the IF part of such a rule and fires the rule if it matches. If a match is found the goal is proven and the process can stop immediately. Often the goal is not proven immediately. In this case, the rule engine saves the rules it has found in the first iteration and sets a new goal, proving the IF part of these rules. The system explores the rules that can potentially prove this newly defined goal and this process continues until the main goal is proven, or no more rules can be found that can prove the current goal.

We saw that using forward chaining four rules would fire in the example above. What happens when we use backward chaining with the same rule base and the same initial values? Suppose again our goal is to know the value of Z. In the first cycle the rule base is searched for rule that can prove Z. Only rule 2 is able to do this as it has Z in its THEN part. This rule shall not fire because it requires B, C and X to be true. While B and C are already true, X is not and therefore X becomes the new goal. In the second cycle rule 3 is found as being able to prove X and this rule will fire because D is already true. As this happens X is added to the database, meaning that rule 2 can now also fire and Z is proven. The cycle stops.

Just as with forward chaining, there are several strategies to chose from when using backward chaining. In the first cycle of our example we saw that only rule 2 was able to prove Z. Suppose the following sixth rule would exist: E -> Z. After the system has found rule 2 as being able to prove Z it has two options. The first is to explore the IF part of this rule and try to prove it as was done in the example. The second option would be to look further for other rules that can prove Z. By using this strategy the sixth rule would be found and immediately fired as E is already true. These strategies are called depth-first and breadth-first respectively (Graham, 2006).

The best way to choose a chaining mechanism is to study how an expert would approach the problem. If an expert starts with gathering all the information and reasons with that information forward chaining is the best choice, if an expert starts with a specific hypotheses go for backward chaining (Negnevitsky, 2005). Forward chaining is mostly used for analysis and interpretation, while backward chaining is often found in diagnostic systems.

Opportunistic chaining

Forward and backward chaining are the two main forms of inference that are used by rule engines. In practice, most of the time a combination of these two strategies is used. Frequently an approach is used that starts with backward chaining. When a new fact is inferred forward chaining is used to maximize the use of this new data, this is called opportunistic chaining and is found in the better BRMSs (Negnevitsky, 2005; Graham, 2006).

As shown earlier, forward chaining uses a naïve algorithm that can become very slow as the rule base grows. Although backward chaining is more efficient, it is only applicable when you start out with a specific goal you are interested in. Rete is very efficient pattern matching algorithm that avoids iteration over the elements in the short-term memory, as well as over the items in the long-term memory (Forgy, 1982). The algorithm uses a tree-structured network of nodes. Each node, except the root, in the three corresponds to a pattern in the *IF*-part of a rule. The complete path from the root node to a leaf node describes the *IF*-part of one rule. The complete tree represents the entire rule bases, i.e. the long-term memory and thereby avoids the need to iterate over each rule. To avoid the need to iterate over the short-term memory (the facts) each node stores the facts that match the pattern. When facts change, matching nodes are computed and their memory is updated with the fact that matches the pattern. When a fact or combination of facts causes all of the patterns for a given rule to be satisfied, a leaf node is reached and the corresponding rule is triggered.

Although the Rete algorithm results in great speed increase, it uses a lot of memory to do so. As the rule-base grows the performance gain of Rete becomes larger, as Rete performance is nearly independent of the number of rules, as no iteration takes place. In very large rule-bases, the increased memory burden can become a problem. Many Rete-based algorithms, often proprietary, have been designed to address this problem.

CONFLICT RESOLUTION

An important aspect that rule-matching algorithms need to consider is how to handle contradicting rules, i.e. rules that are both eligible for firing based on their *IF*-parts, but have mutually exclusive *THEN*-parts. Consider the following two rules:

1. IF the traffic light is green
   THEN the action is go
2. IF a car is crossing
   THEN the action is stop

When your traffic light is green the condition part of the first rule is matched. But when another car is crossing because it runs a red light for example, also the second rule is matched. As the have mutually exclusive conclusions these rules cannot both fire. These rules represent a conflict set and the rule engine must make a decision. A method for choosing a rule to fire when more than one rule can be fired is called conflict resolution (Negnevitsky, 2005). When employing a simple forward chaining approach both rules will fire, but each in a different cycle, causing the last rule to fire to overwrite the action of the first rule. This illustrates the impact of rule order in forward chaining.

When you use a goal-driven approach and stop rule execution when the goal is reached using backward chaining only one rule will be fired and the conflict seems resolved. When the goal is to determine the value of the *action* object, this approach finds this can be achieved by evaluating both rules. When the first rule is evaluated the goal is reached and rule execution stops. Although only one rule is fired with this approach the rule order is still just as important.

If the rule base is relatively small and the inference strategy is constant considering the order of the rules in the rule base might be a sufficient approach to conflict resolution. But as the rule base grows or different methods of inference are employed this will not suffice.

Shirai and Tsuji (1982) and Giarratano and Riley (1998) describe several other approaches to conflict resolution:

- Fire the rule with the highest priority. This approach is popular in medical diagnostic systems (Durkin, 1994). Drawbacks are that rules must be prioritized and in theory it is still possible to have a conflict if two eligible rules have the same priority.
- Fire the most specific rule. This longest matching strategy, as it is often called, is based on the assumption that more specific rules process more information. When two conflicting rules are eligible for firing because a shared condition is met, but one rule also has a second condition, which is also met, that rule will fire.
- Fire the rule that uses the data most recent added to the working memory. This method requires that each fact added to the working memory must be time tagged. In a conflict rule set, the rule of which its condition uses the most recent added fact will fire. This technique is especially useful for real time expert systems in which the data changes constantly. Note that this method does not provide a solution for conflict sets that have the exact same *IF*-part, as with our traffic light example.

These approaches can be implemented relatively easy and can still be very effective. In some rule-based system a more advanced conflict resolution method is used, metarules. These are separate rules that describe the strategy to use in conflict resolution. An example of such a rule is: rules concerning human lives have higher priorities than rules concerning assets. This example shows how difficult it is to implement such metarules, as the system should know when a rule is concerned with human live. None of BRMS's in this thesis provide this kind of functionality. They do provide functionality to detect possible conflicting rules, so they can be addressed before they are used in a production system.

3.5.Extracting business rules

Business rules can be found everywhere throughout organisations, every process is governed by rules. These rules come in very different forms, some are documented clearly, some are implicit and with regard to software systems, they are often buried deep down inside the source code. Therefore, capturing the rules that are to be included is a vital part of the process of implementing a BRMS. In the field of knowledge engineering this process is referred to as knowledge acquisition. Because of the different forms in which rules appear and their decentralization, knowledge acquisition is a difficult process, which is often regarded as the bottleneck in knowledge system development (Wagner, 2006). Several attempts have been made to (partly) automate this process, often through data mining. Crerie et al. (2009) describe a method for discovering business rules from information systems event logs through the use of process and data mining techniques. They have shown that is possible to automatically extract authorization action assertions and condition action assertions. Their technique does require structured logs of all transactions from information systems.

Another important source of business rules is (legacy) source code. As the software that encompasses business rules grows large, the embedded rules become difficult to extract. Also, software is often changed without updating the corresponding documents, leading to a situation where the rules hidden in code are more reliable than any other document. Huang et al. (1996) describe an interactive approach to business rule extraction that combines variable classification and program slicing. Variable classification is based on the idea that a business rule is usually centered around data and thus variables. A large application contains thousands of variables and Huang et al. present some heuristics to automatically classify the variables that are concerned with business rules, these heuristics are mainly concerned with variables used as input and output parameters of functions and procedures. Once these variables have been identified, program-slicing techniques (Huang, Tsai, & Subramanian, 1996; Joiner, Tsai, Chen, Subramanian, Sun, & Gandamaneni, 1994) are used to extract relevant code fragments. These code fragments form the input for manual rule extraction. It is beyond the scope of this thesis to elaborate on the process of knowledge acquisition with regard to documented and human knowledge.

3.6.Expressing business rules

Once the business rules that are important to the system are clear, they have to be entered into the BRMS. The way in which rules are represented differs among the different BRMS's on the market and is an important differentiator. When it comes to expressing rules, the challenge is to express them in a way that is understandable for humans as well as for computers. Much research has been done throughout the years to address this challenge. Ross (2000) states expression of rules to users should be cleanly separated from the system internal representation. Most BRMS's offer multiple means of external representation and it depends on the type of rule what method is more applicable. Ross (2000) mentions the following types of external representation and when they could be applied:

- Decision tables for value thresholds and certain computations
- Point-and-click expression builders for limits and type consistency
- Structured language for complex restrictions and inferences
- State transition diagrams for state transition rules
- Data model for property rules

Next to the internal and external representation of rules Ross also distinguishes a third layer of representation, a single unified conceptual representation. This representation should be transparent to the rule specifiers, but visible to analysis tools and rule engineers. Ross states that while all three layers are important, the representation layer is the most powerful. Predicate logic, objects and production rules are well known representation formats of this level that have been used for many years in knowledge systems as the means of knowledge representation.

Predicate logic is a mathematical way of describing assertions, attributes and relations. The rule 'all men are mortal' would be expressed in predicate logic as follows:
$\forall_X$: man(x) $\Rightarrow$ mortal(x)

A detailed introduction to predicate logic syntax can be found in Lucas and Van Der Gaag (1991).

The main advantage of predicate logic is its ability to express domain knowledge completely declaratively and this enables systems to reason with them. They are, however not easy to understand for humans and not applicable to every kind of rule (Lucas & Van Der Gaag, 1991).

Thanks to the popularity of the object-oriented programming paradigm, *objects* have fast become a well-known form of knowledge representation. An object is described by its name, its properties, its methods and its parent. Objects are closely related to a much older form of knowledge representation, frames. Objects have the advantages that they are relatively easy to understand for humans. They lack, however, expressiveness and are therefore not suitable as the only form of knowledge representation.

Production rules were already covered extensively in earlier in this thesis. They are relatively easy to understand for humans, while still interpretable for computers.

Due to the increasing awareness of the importance of business rules, new forms of representation have evolved to specifically address business rule representation.

The progress in the field of natural language processing has led to the introduction of natural language as a means of representing rules in a BRMS. One of the systems evaluated later in this thesis uses this as the main form of rule representation. While rules presented in natural language are great for business people, the relatively immaturity of natural language processing poses some constraints.

Be it natural language or not, all BRMS products are able to represent rules as sentences, which relates to the structured language representation in the list of Ross (2000) mentioned earlier. Often, these sentence have structure consisting of IF and THEN. For example:

IF the customer's basic salary is above 30,000 and the loan is less than 5,000 and the customer does not have a low rating THEN approve the loan

Morgan (2002) recommends a better style aimed at resembling natural language while removing ambiguity, making relationships explicit, avoiding obscure terminology and so on. The above rule would become:

A loan may be approved IF the basic salary of the customer is above 30,000 and the loan is less than 5,000 unless the customer has a low rating

Other means of representation available in most BRMS's are decision trees and decision tables. Both provide a concise view of a set of business rules in the form of tree or spreadsheet. They help managing and navigating large sets of rules. Decision tables are simple rules composed of rows and columns, where each row represents a condition or an action and each column represents a rule, or the other way around. Table 2 shows an example of a decision table for a car rental company. The third column, for example, shows that a customer can rent a car in Idoha if he is between 16 and 25 years old and has had less than two accidents. Decision tables make it easy to identify overlap or gaps in the rules. Table 2 makes it clear that there is no rule for customers aged above 65 with less than five accidents in Maine. The main problem is that they can become unmanageably large when there are many conditions in the rule base. When organizations already hold their knowledge in this form, e.g. pricing charts or rate tables, decision tables are a really convenient choice.

Condition								
State	Idaho	Idaho	Idaho	Idaho	Idaho	Maine	Maine	Maine

Age	<16	16-25	25-65	-	>70	<18	18-65	-
# accidents	-	<2	<4	>3	-	-	<=5	>5
Action								
Rental approved	False	True	True	False	False	False	True	False

Table 2 A decision table for a car rental company

Decision trees provide the same function as decision tables, but are composed of branches where the root node represents a condition and the leaves represent actions. They are well suited for a large set of rules that have some conditions in common, but not all. Figure 6 shows a small decision tree for sending a brochure.

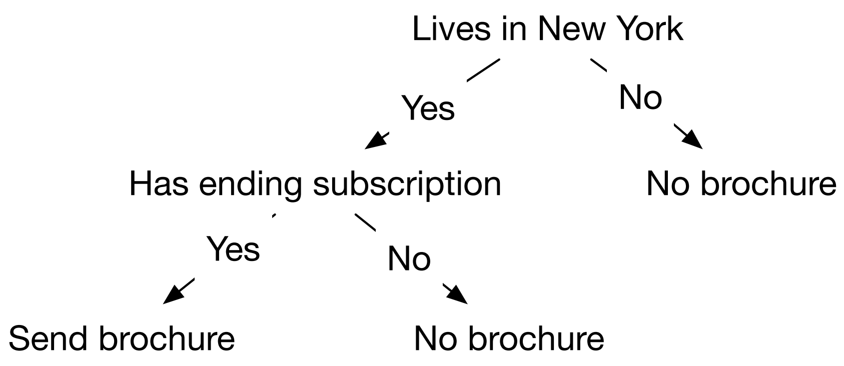


Figure 6 A decision tree

3.7.Closing words

This chapter presented a historic overview of BRMS's and showed they are essentially a new form of knowledge system. Architectural details were discussed with special regard to the rule engine, a core component. The process extracting business rules and expressing them, an important part of BRMS implementation, was also covered. This information leads to a list of requirements to which a BRMS should minimally adhere. A BRMS should:

- have a rule authoring environment
- not be domain specific
- be able to show the reasoning steps towards a conclusion

- have a separate rule engine
- provide forward and backward chaining capabilities
- be rete-based
- have conflict resolution capabilities
- provide different means of rule representation

This list will be supplemented in the next chapter which discusses related developments and will be used in the selection of BRMS's in which a use case will be implemented in chapter 6.

4. RELATED DEVELOPMENTS

The rising attention towards business rules and the emergence of business rules management systems is not a movement that comes on its own. There are many more or less related developments that caused the demand for these systems or helped their implementation and acceptance. This chapter covers the most important developments that are often associated with business rules management systems.

4.1. Service Oriented Architecture

Organizations rely more and more on a large number of complex software systems to support them in their everyday business. This can be regarded as a major factor in the emergence of business rules management systems. The complexity of having many advanced software systems interacting with each other has led to the development of a software architecture of multiple components in which each component has a clearly defined discrete task and there are standardized interfaces between these components. This decentralized architecture is often referred to as a service oriented architecture (SOA). As this section will make clear, many of the ideas and techniques of SOA carry directly over to the business rules approach that was covered in paragraph 2.3. This makes that BRSMs and SOA often go hand in hand. Graham (2006) even goes as far as stating that SOA without BRMS is like a runaway train with no wheels.

There exist some slight misunderstandings of what SOA actually entails. Many define it mentioning web services using specific standards such as SOAP (Simple Object Access Protocol) and WSDL (Web Services Description Language). However, SOA is merely a general description of a software architecture without any regard to techniques used to achieve this architecture. A SOA can be implemented using any service-based technology. The loosely coupled character of services in a SOA is critical, service interfaces should be completely independent of their implementation. A software developer should be able to build applications using multiple services without having to know the underlying technology of those services. So a service could be implemented in a language running on a J2EE or .NET environment, while the application consuming the service could be written in a completely different language, running on a different platform. This does, however, require a communication protocol to be used by the service provider and the service consumer. And while SOA does not dictate specific techniques to achieve this, different attempts have been made to standardize service communication.

Web services provide such a standardized way of interoperating amongst applications using standard internet protocols independent of platform or programming language. Web services are essentially a combination of HTTP and XML. HTTP (Hypertext Transfer Protocol) is an application protocol for distributed information systems and forms the foundation of communication for the World Wide Web. XML (Extensible Markup Language) is a language to structure data by marking individual items with tags hinting at what each item represents. XML can be used to exchange complex messages between different platforms and programming languages. While XML and HTTP present a way to represent and transfer a message, there is still need for a language to describe the format of the message and a mechanism to mange the interchange of messages. Also, since distributed systems are concerned, there is a need for a way to locate services. The following elements fulfil these requirements:

- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description Language)
- UDDI (Universal Description, Discovery and Integration)

Figure 7 shows an overview of a typical web services architecture and shows where the three elements mentioned above fit in.

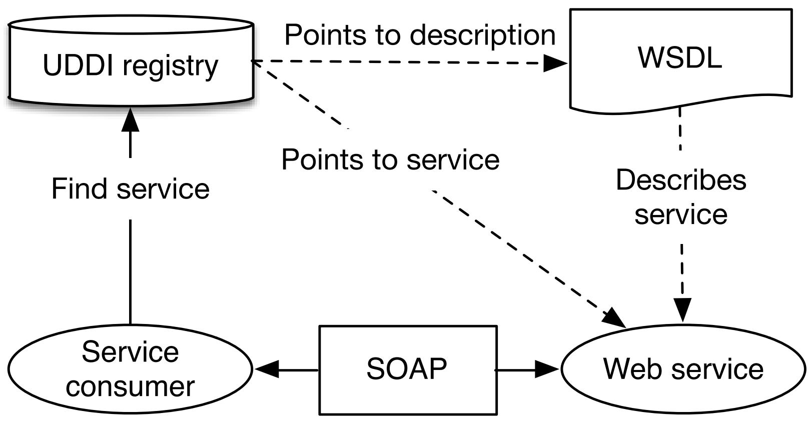


Figure 7 Web services architecture

SOAP

Most software platforms, like .NET or J2EE, provide ways to remotely call functions or procedures in a distributed environment. One major problem with these mechanisms is that they are not platform or even programming language independent, making them unsuitable for a SOA environment, as they violate the loose coupling of services. There is a need for a way of sending messages that are in itself platform independent and are also transferred independently. SOAP is communication protocol for exchanging structured information between applications. It wraps up a document in XML and uses other protocols to move it over the internet, usually HTTP. A SOAP method call consists of a request and response that conform to the SOAP standard. It is way to call a specific procedure of a service with the appropriate arguments and receiving its response in a predefined way.

A SOAP message is an ordinary XML document containing the following elements:

- A required Envelope element that identifies the XML document as a SOAP message.
- An optional Header element that contains application specific information about any bureaucracy surrounding the service.
- A required Body element that contains the actual message.

- An optional Fault element that provides information about errors that occurred while processing the message.

WSDL

A SOAP message must conform to a predefined structure in order to be understood. Such a structure can be defined using a WSDL. WSDL is an acronym for Web Services Description Language. Just like SOAP, a WSDL document is expressed in XML. This XML document describes the web service's capabilities, like the available operations that can be called, including their parameters and the form of their response. A WSDL also contains information about the internet addresses at which the operations can be invoked, so called end points. In short, a WSDL completely describes what a valid SOAP request and response should look like for a specific operation of a given web service. To put it in programming terms, a WSDL is a procedure declaration and SOAP is the procedure call mechanism. This characteristic of WSDL makes it possible to auto-generate a boilerplate SOAP request for which a programmer only needs to provide the appropriate input.

UDDI

UDDI stands for Universal Description, Discovery and Integration. It is an

XML based register that enables businesses to list themselves and their web services on the internet in order to be found by other business. The goal of UDDI is to simplify online transactions by making it easy for businesses to find each other and let their systems work together. UDDI consists of three categories, often referred to as the colored pages.

The white pages provide basic information about the companies providing a service. They allow businesses to discover services based upon the business name.

The yellow pages allow services to be found by business categories. A service might fall in multiple categories at the same time.

The green pages provide technical information on the behavior and use of the offered services, such as service location and specification.

The communication to a UDDI directory can be done with SOAP messages. These provide access to the WSDL's of the listed web services. While WSDL is the most popular method to describe web services, WSDL is not required by UDDI as the protocol to describe services.

More and more companies have adopted SOA, but only few really succeed in setting up an effective architecture. An often made mistake made when implementing SOA is to use many web services as wrappers around their existing legacy systems (Graham, 2006). This leads to many low-level interface, poor reuse and extensibility and poor maintainability. SOA done properly should be implemented with a top-down approach, where services represent business processes. A good SOA should be understood by IT as well as the business. Therefore BRMS's should play a vital part when implementing SOA. Some go as far as stating that BRMS's and SOA should be done either both together, or not all (Nelson, Peterson, Rariden, & Sen, 2010). As a consequence, the ability of a BRMS's to function as a component within a SOA is a requirement.

4.2.Ontologies

As mentioned earlier, business rules are specific kinds of statements. Statements always need a context to be understandable. This is called the vocabulary of the domain, or the domain ontology. Originally, ontology is the philosophical study of the nature of being, becoming, existence, or reality. But in computer and information science it is described as form of formal knowledge representation by defining a set of concepts within a domain and the relationship between them. It is often used as a domain model and some like to compare it with a database schema. With regard to business rules, the ontology specifies the terms that are allowed to be used when writing rules. Without a sound ontology the rules are meaningless.

Over the past years ontologies have drawn considerable attention in the field of knowledge engineering. Knowledge and information systems are generally regarded as representations of the real world. Ontologies provide a vocabulary to describe this world and its concepts and show how they relate. Essential to ontologies is that this vocabulary offers a common understanding and can be communicated between people and application systems (Gruber, 1995). Ontologies consist of sets (or concepts), attributes (or properties), and relationships. Ontologies are typically specified in languages that allow abstraction away from data structures and implementation strategies, for this reason ontologies are said to be at the semantic rather than at the physical level (Sangers, 2008). Although there are numerous languages available to represent ontologies, Graham (2005) concludes that with regard to a BRMS a type model in UML or a semantic network can be used to represent any ontology needed. A semantic network, as shown in Figure 8, represents concepts and the semantic relations between them within a specific domain. Most BRMS products provide some level of support to automate the process of creating such ontology. Ontologies are especially important for systems that attempt to state rules in natural language. Any attempt to achieve this will certainly fail if there is no well-constructed model of the concepts, with their attributes and relationships, in the domain.

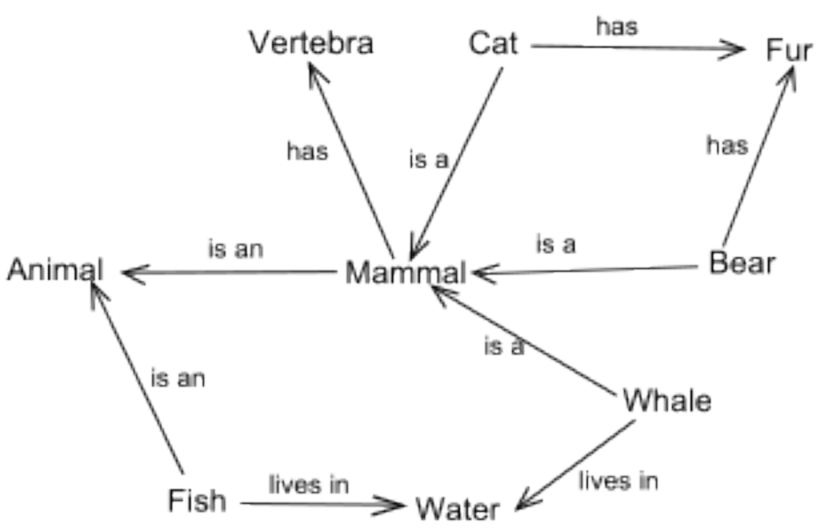


Figure 8 A semantic network (Wikipedia)

4.3.Compliance

Ontologies have been studied in the context of knowledge and software engineering for several decades now. A more recent study field within software engineering is compliance. Simply put, compliance means abiding by the laws and rules that apply to the specific type of organization (Sangers, 2008). The need to be compliant concerns all people and processes within an organization. And as these are almost always supported with software systems, they need to be compliant as well. As business policies and environments change constantly, there is a growing need for software systems to be compliant with these policies and other legislation.

In recent literature, different approaches toward achieving this have come by. One example is the ampersand method developed by Joosten (2007). This method proposes specifying business rules in a special language called ADL (A Description Language) and uses relation algebra to generate IT systems that are compliant with these business rules.

Compliance does not only entail making sure your systems are compliant, but also that it can be proved that they are. Compliance should therefore be traceable so that it could be verified that rules which were agreed upon are implemented correctly. This is easier said than done and at this moment, compliance projects take up a large part of the budget of IT departments and this is not likely to change in the foreseeable future (Mercury, 2005). A BRMS can play a very important role in the need to be compliant and proving that a system is. The externalized rule base used by a BRMS makes the business rules a system uses easily auditable (Weigand, van den Heuvel, & Hiel, 2011).

4.4.Concluding remarks

This chapter shows that BRMS's go hand in hand with other developments. SOA is the definitely the best example. Two of the core motivations for implementing a BRMS are separating business logic from application code and making the rules available to multiple business applications. It is therefore obvious that a BRMS should not be tightly coupled with application code. SOA provides an excellent solution to prevent tight coupling and facilitate reuse.

Ontologies are essential for business rules to make sense. For a computer system to make sense of business rules it is necessary to know concepts within the business domain and how these concepts relate to each other.

A newer field of study is compliance. This involves making sure business applications abide laws and regulations. By externalizing business rules, it becomes much easier to audit a system in terms of compliance. The conclusions from this chapter add two new requirements to list started in section 3.7, a BRMS should also:

- be able to function as a service within a SOA (using SOAP)
- provide means to build an ontology

5. BRMS'S IN PRACTICE

The previous chapters presented a theoretical discussion on business rules and BRMS's. Many benefits of separating rules from application logic are mentioned, but how do these promises hold up in practice? Within CGI there is little experience with the implementation and usage of BRMS's. This was shown by a small study conducted by one of its Dutch employees on CGI's capacities with regard to BRMS's (Appendix A). It also showed that this was especially the case in the Netherlands. In this study a scan of the resumes of all employees on all known business rules products was performed. This scan made clear that of the market leading BRMS's there was only real experience with ILOG Jrules, but mainly in the United Kingdom. The full notes of this study can be found in appendix A.

In order to get more insight in how BRMS's perform in practice interviews were held with CGI employees who had some direct or indirect experience with BRMS's. Also, internal discussion boards were searched for discussions covering the topic of BRMS's. Because of the earlier mentioned shortage of many experienced people in the Netherlands the people interviewed had no in-depth experience with BRMS's themselves, but were involved in projects in which a BRMS's was used. Analysis of internal discussion boards did show many interesting comments of people how were directly involved in the implementation of BRMS's. The interviews showed mainly mixed feelings towards BRMS's. It was generally found a good approach to separate the rules from the source code. But especially the programmers found it difficult to find an effective way of working with the externalized rule base, causing projects to seemingly take off slower than similar projects without a BRMS. But this could very well be caused by a lack of experience. Another point of critique was the effort of authoring the rule base, this was found to be a rather steep learning curve, as it involved learning a new rule language and getting familiar with the authoring software. In general, the interviews were not very concluding as to whether a BRMS was a successful choice.

The experiences shared on the discussion boards, however, where more outspoken. The general attitude towards BRMS's varied from very negative, to mildly positive. Many of the negative remarks revolved around the process of rule authoring. One person notes that the use of a BRMS was a bad choice in their project, because implementing the rules in a BRMS is far more complicated than in the traditional way in source code. Even for developers experienced with the BRMS it takes more effort to implement rules in a BRMS. The following reasons are mentioned: a specific data model must be defined inside the BRMS and mapped to the model you use in the rest of the application; and the rule syntax forces the rule author to develop a single very simple rule with a basic *if then else* structure at a time, so for more complex business rules you actually have to develop many rules.

Another point of criticism that showed up multiple times was targeting the often-made claim by vendors that BRMS's enable business experts to create and update rules themselves without resorting to the IT department. This was found to be too positively stated or even utterly untrue. The rule authoring environments require rules to be structured according to a specific syntax and conform to a specific data model, making it non-trivial for non-technical users to write new rules, or even change existing ones. Also, change control and release mechanisms mean that it is not possible that users would be allowed to directly update a live system. One person, however, did mention positive experience with business experts updating the rules, but stated that this requires another layer of abstraction by providing your own user interface which constraints the users from what they can do.

The last negative points mentioned are painful integration with traditional software and complexity of the rule base. This last point is illustrated by the fact that forward-chaining rules that introduce new assertions into working memory can become hard to understand and debug as the application complexity increases.

On the other end of the spectrum successful projects using a BRMS were mentioned in which the BRMS proofed to be of value. Good performance was mentioned, which becomes increasingly important as the rule base grows. And, as stated earlier, it is possible to have the rules maintained by business experts, but it does require some extra effort. Rules were found to be quite expressive, making them easier to understand by business experts that rules expressed in a traditional programming language. Also, success stories of BRMS's in other projects were mentioned.

As were to be expected by the results of internal study mentioned in the first paragraph of this chapter, the remarks in these discussions were mainly based on experience with ILOG JRules. And while more outspoken, many experiences mentioned in these discussions concur with those that came forth out of the interviews, which were held with people who were only familiar with Oracle Policy Automation. This makes it appealing to think these experiences can, at least to some extend, be applied to BRMS's in general.

Overall, the opinions on BRMS's of people that have had experience with them on large projects tend to be mostly negative. On the other hand, there are projects that successfully use a BRMS, where it was found to be of added value. So why is negative sentiment towards BRMS's predominant? A general pattern can be observed in the cases in which people were largely negative. In general, expectations of the ease of the rule authoring process were too high, developers had little to no experience with BRMS's, and the decisions to use a BRMS seemed to be driven by marketing, rather than by the applicability to problem at hand. These findings stress the need for a better understanding of their suitability and guidelines for the implementation of a BRMS.

6. USE CASE

So far, BRMS's have been discussed theoretically and the previous chapter examined how they were experienced in practice. These experiences stressed the need for a better understanding of their suitability and implementation guidelines. In order to address these needs a realistic use case will be implemented in several market leading BRMS's.

6.1.Method

BRMS's SELECTION

The first important criterion in the selection of these systems is that each of them should fulfill all requirements listed in section 3.7 and 4.4. The complete list is as follows, a BRMS should:

- have a rule authoring environment
- not be domain specific
- be able to show the reasoning steps towards a conclusion
- have a separate rule engine
- provide forward and backward chaining capabilities
- be rete-based
- have conflict resolution capabilities
- provide different means of rule representation
- be able to function as a service within a SOA (using SOAP)
- provide means to build an ontology

An important choice in the process was to decide how many BRMS's to include in the evaluation. Getting a good sense of different possible approaches by which these requirements can be fulfilled is critical to produce general guidelines. In order to compromise between getting a good feel of different approaches and being able to evaluate with a satisfying level of depth, three systems will be evaulated.

The initial shortlist of candidate BRMS's was based on a Forrester Wave edition on business rules platforms (Rymer & Gualtieri, 2008). The Forrester Wave is an independent and transparent evaluation of vendors in a software, hardware, or services market. Forrester (Rymer & Gualtieri, 2008) included 13 systems in the assessment. Based on criteria as features, vendor strategy and market presence Forrester marked five as the general leading systems. These five are Policy Automation from Oracle, ILOG JRules from IBM, Blaze Advisor from FICO Business Rules Management System from Corticon, and PegaRULES from Pegasystems. PegaRULES lacked chaining capabilities and is not RETE-based. The other systems ticked all boxes on the requirements list, but Corticon's system was not chosen because its market presence lacked behind and is therefore the least interesting from the perspective of CGI. The three systems in which a use case will be implemented are thus: Policy Automation from Oracle, ILOG JRules from IBM, and Blaze Advisor from FICO. The first two were already mentioned in chapter 5, as having been used by CGI. There was no experience with Blaze Advisor found within CGI.

Focus

The goal of implementing a use case in three BRMS's is twofold: 1) To extend the requirements list mentioned above; 2) To gain more insight in the implementation process. To address these goals evaluation of the BRMS's will focus on multiple aspects.

The previous chapter made clear that rule management by non-technical business people often does not work well. Therefore the usability of the rule authoring facilities and ontology creation is a focus point. Specifically in how it appeals to non-technical people.

To function as a service within a SOA is mentioned as a must for a BRMS, but chapter 5 showed that integration with existing systems can be cumbersome. The use of a BRMS as a service is therefore considered.

Although some remarks will be made on feature set and usability the evaluation is not meant as a review or general comparison of the three systems.

USE CASE

There are several demands on the use case. With regard to the rule authoring environment it is interesting to have some rules that can be represented in an alternative way such as a decision table. As stated in section 3.6 value thresholds are well suited to be represented in decision tables. With the same regard it also interesting to see how different types of rules are handled, like calculations. In order to concern aspects as conflict resolution and chaining methods there need to be rules to reflect this. Meaning there must be conflicting rules, and rules that require other rules to be fired first in order to fire.

In general, these demands can be served by including rules in the use case that fall into the different rule types mentioned in Table 1 as these can be mapped very well to these demands. Different types of rule representation can be achieved by using different *action assertions*. While calculations and insight in chaining methods can be achieved by using *derivations*. *Structural assertions* are used automatically as the other rule types build on top of them.

Policies, laws and legislation are the main examples of rules that are well suited for a BRMS (von Halle & Goldberg, 2006). To have a highly realistic use case the choice was made to base it on an existing government policy and adapt if necessary to abide to the demands mentioned before. Also CGI has multiple projects in its portfolio concerning government policy.

Together with CGI several possible use cases were discussed based on experiences from previous CGI projects. The decision was made to use an Australian policy on determining income support allowance as the basis. CGI was already familiar with this policy, but not in combination with a BRMS.

In short, the use case is this: A person with no or a low income wants to know if he/she is eligible for an income support supplement, and if so, for what amount of supplement. A more detailed overview of this scenario and the rules involved will follow later that in this chapter.

The Australian income support policy (Australian Government, Department of Human Services) was found to be a good choice because it covers all demands mentioned in this section. The rules for determining eligibility consist of *constraints* like *age* and *residency status*. There are many *conditions* like *if taxable income is less than $18,000 then tax requirement is fulfilled*. To determine taxable income *calculations* are required and *inference* is needed for requirements based on multiple rules, like *residency status*.

6.2.Scenario

The scenario used in this use case is this. A person with a low or no income uses a web application to determine eligibility for an income support allowance. The web application guides the user through the appropriate questions to determine eligibility and is able to explain the conclusion. The web application is part of the scenario to represent a system that uses the business rules in the BRMS as a service. The choice for a web application instead of other possibilities, like a desktop or mobile application, was made arbitrarily.

As stated earlier, the rules used in this scenario are based the Australian income support policy. The complete set of rules for eligibility is stated on the web site of the Australian Department of Human Services (Australian Government, Department of Human Services). To make the rule base not unnecessary large, some rules were excluded because they fall into the same rule classification as other rules and therefore do not serve any extra purpose. The complete rule base used is listed in appendix C. In summary eligibility is determined by age, marital status, income, residency, tax offsets, and some specific requirements regarding mental state and imprisonment.

The first step is to express the rules in a structured format. Using the style proposed by Morgan (2002) as mentioned in section 3.6 rules will look like this:

Candidate meets the residence requirement for the Low Income Supplement
IF candidate is in Australia on the date of claim
AND in the previous financial year, candidate was in Australia for at least 46 weeks

Rules formatted this way are used as the bases for implementing them in the business rules management systems. The complete rule base listed in appendix C is formatted this way.

6.3.Evaluation

FICO Blaze Advisor

Blaze Advisor is a complete solution for rules management by FICO. Blaze Advisor includes an architectural framework for building and creating business rule applications and tools for deploying and maintaining rule services. FICO defines a rule service as a software component that is called by an enterprise business application to provide a decision. In other words, it is the combination of a rule repository, a rule engine and an interface. Blaze Advisor comprises of the following components:

- Integrated Development Environment: A powerful GUI to write, edit and test rules, with direct access to a powerful set of error checking, logic validation, visual comparison and decision simulation tools.
- Rule repository: A repository to store rules and manage rule changes.
- Rule engine and server: Here the rules are deployed to and executed from.

- Rule Maintenance Application (RMA): A web-based application to preview, verify and test rules. It also provides configurable rule widgets that can be used to change the rules on the fly.

All of these components are reflected in the overview of Blaze Advisor's architecture in Figure 9.

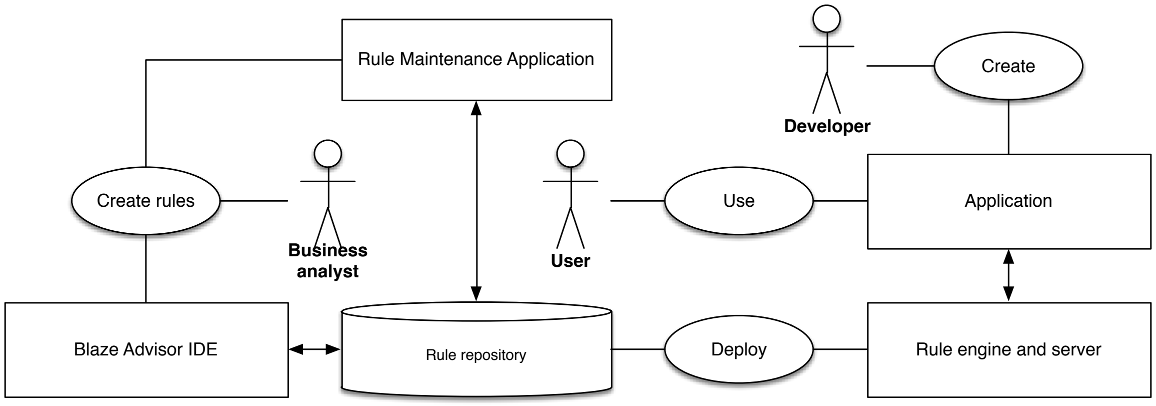
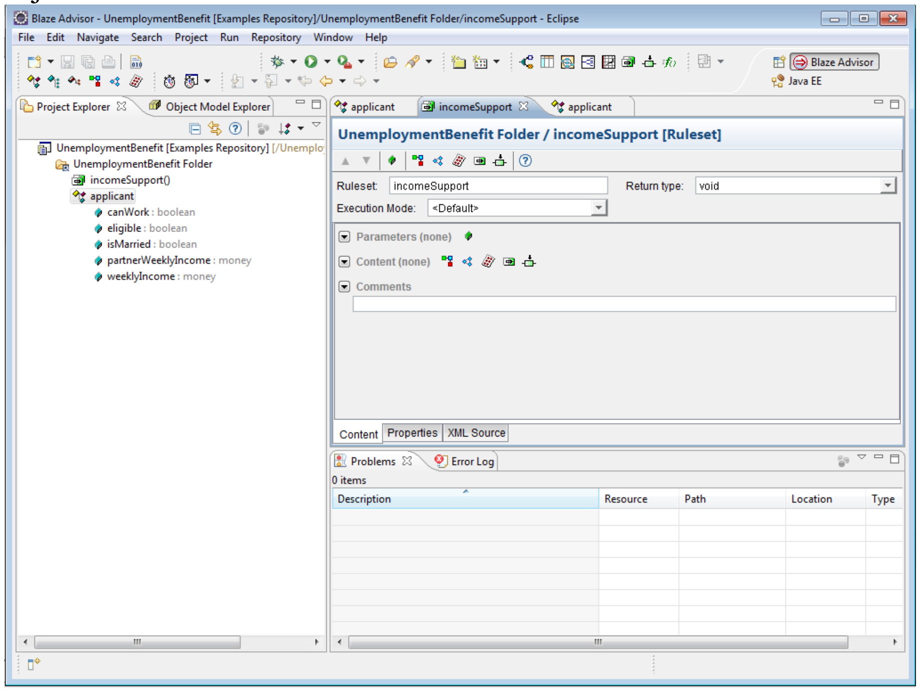


Figure 9 Blaze advisor's architecture

Rule authoring

In Blaze Advisor rules are written in an Integrated Development Environment. FICO did not create its own IDE, but instead provides a plug-in for Eclipse. Eclipse is an open source platform written in Java for software development environments by the Eclipse foundation. Its most familiar use is as IDE for writing Java code. Because it is written in Java it runs on all popular platforms. Eclipse has an open structure and – except for its core – is completely made up of plug-ins. The installation of the Blaze Advisor rule authoring environment requires a compatible JVM and Eclipse version to be installed. When these requirements are met the Blaze Advisor plug-in can be installed into Eclipse. Using an Eclipse based user interface (Figure 10) for rule authoring will make it feel like a familiar environment to programmers.

When designing a rule service in Blaze Advisor, the first step is to design a repository, and then create a project, which is stored in the repository. Next, data definitions (business object models) are created or imported into the project, making ontology creation a separate step. Rules are written against the objects that will be available at runtime. Finally, rules are organized into decision tasks that control the flow of the project.


Blaze Advisor - UnemploymentBenefit [Examples Repository]/UnemploymentBenefit Folder/incomeSupport - Eclipse
File Edit Navigate Search Project Run Repository Window Help
Blaze Advisor
Java EE
Project Explorer
Object Model Explorer
applicant
incomeSupport
applicant
Content
Properties
XML Source
Problems
Error Log

UnemploymentBenefit [Examples Repository] [/Unemplo
UnemploymentBenefit Folder
incomeSupport()
applicant
canWork : boolean
eligible : boolean
isMarried : boolean
partnerWeeklyIncome : money
weeklyIncome : money

UnemploymentBenefit Folder / incomeSupport [Ruleset]

Ruleset: incomeSupport
Return type: void
Execution Mode: <Default>

Parameters (none)
Content (none)
Comments

0 items

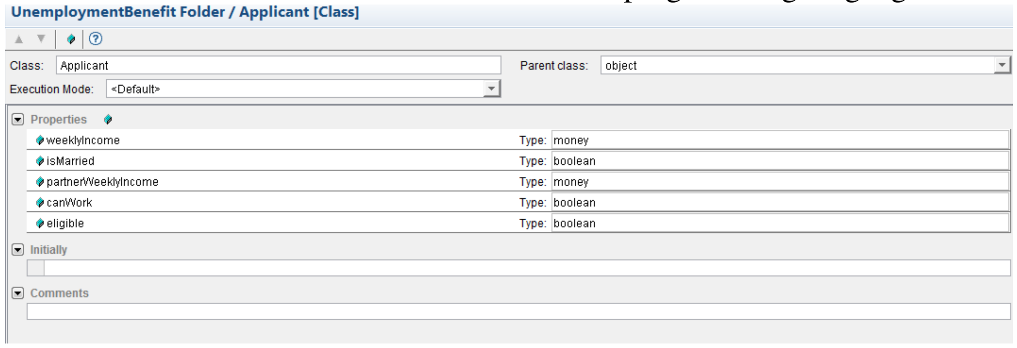
Description	Resource	Path	Location	Type

Figure 10 Blaze Advisor's user interface

Before creating a rule project, a repository must be created. A repository can be implemented as a file-based, database (JDBC), or Lightweight Directory Access Protocol (LDAP) repository and can use different services, e.g. versioning, authentication and authorization. Blaze Advisor comes with a default repository that can be used to experiment. Before creating the first business rules, the business objects that are used by the rules must be created beforehand. Business objects are represented as classes and can be created from scratch or imported by mapping Java classes, XML files, or database records via a wizard. A business object resembles objects as used in object-oriented programming languages. It contains properties that have a specific type, e.g. boolean, string, or date. Figure 11 shows a class of an applicant with several properties in Blaze Advisor.

Once the business objects to be used in the rules are defined the rules can be written. Rules are expressed as a conditional statement followed by an action. A simple rule has the following form:

if condition(s)
then action(s)
else alternate action(s)
This structure resembles how conditions are handled in programming languages.


UnemploymentBenefit Folder / Applicant [Class]

Class: Applicant
Parent class: object
Execution Mode: <Default>

Properties

weeklyIncome	Type:	money
isMarried	Type:	boolean
partnerWeeklyIncome	Type:	money
canWork	Type:	boolean
eligible	Type:	boolean

Initially

Comments

Figure 11 A business model class in Blaze Advisor

Rules in Blaze Advisor are written in FICO's own rule language called Structured Rule Language (SLR). SRL is an object-oriented programming language intended to provide a syntax for authoring rules that is easily understandable by people with little or no programming background. This is done by having language constructs that resemble natural language as is shown in the following example:

if applicant's canWork and applicant's income is greater than 30.000
then applicant's eligibility is false.

Anybody reading this rule will understand what is meant. So, there is an obvious advantage in expressing rules in a natural language like syntax, business people can browse through the rule base and understand it. However, this example also shows a problem with using an English-like syntax: it leads to rules that are partly correct natural sentences and partly incorrect or unusual. This is especially so for rules considering boolean properties. This leads to an inconsistent syntax that makes it challenging to write new rules. Blaze Advisor does offer a real time syntax checker to help identify problems immediately. This is invaluable, especially when using an inconsistent syntax and will be added to the list of requirements.

SLR also supports another way of representing rules. Unlike using an English like syntax, the rule shown above can also be expressed in a more programmatic way:

if applicant.canWork = true and applicant.income > 30.000
then applicant.eligibility = false.

Having both a technical way and an English-like way of defining rules has the advantage of being both appealing to people with and without a programming background respectively. Choosing between a natural language like syntax and a formal syntax is tradeoff that should be determined by the intended rule authors. It is wise to choose one way of writing rule and keep that consistent within your whole repository.

Blaze Advisor also supports representing rules by decision tables and decision trees. Complex rule sets can graphically formatted using decision graphs. The different graphing formats make it possible to select a view that isolates or identifies a specific aspect of rules without changing the logic.

There is also a way to author rules without using the Blaze Advisor IDE. Blaze Advisor offers the possibility to create a rule maintenance application (RMA) to allow non-technical users to review, edit, delete, or create rules in a web browser without having to know SRL. A rule maintenance application can be created by writing all or a portion of the business rules in templates and then using a wizard to generate the RMA. Rules edited in the RMA can be saved and become immediately effective. Editing existing rules in RMA is made extremely easy by using standard html form elements. This is well suited in the scenario where business people are not the ones doing all the rule authoring, but should be by able to make small changes to the rules. In this case having separate environments for rule creation and rule maintenance is a good solution.

Blaze Advisor provides several ways to test rule sets. One way is to write a function which initializes business objects and then trigger rule execution and inspect the rules. This is all done by writing code and can therefore typically only by done by someone with programming experience. Blaze Advisor also contains a test framework called the brUnit. Test cases can be written and organized into suites that can be stored in the same repository as the rules. Tests in brUnit are XML-based, but can also be created in Java using an API. Both ways are non-trivial to inexperienced users.

Rule deployment

Blaze Advisor Rule Server is the server-side component of the Blaze Advisor BRMS. The rule projects created in Eclipse can be deployed with Rule Server. Blaze advisor supports three types of deployments. Rule services can be deployed as plain Java components, as Java EJB's, or as web services. The first two options are only suitable if the business rules are only used by applications written in Java. Creating a deployment requires the configuration of the following deployment entities:

- Rule Service Definition (RSD): which is used to configure the platform-independent settings, such as the entry point and the rule service interfaces.
- Deployment Definition (DD): which is used to configure the system-independent deployment settings, such as the deployment manager or remote debugging.
- System Definition (SD): which is used to configure the system-dependent deployment settings, such as the connection details for the deployment manager and remote debugger.

These definitions are all created from within Eclipse and can be saved and reused. When these definitions are created a wizard can be used that generates a server configuration and the source code for the deployment. When the definitions are setup to deploy as a web service, the wizard will also generate the wsdl file and build scripts to generate a war file that can be deployed in a JavaEE application server. This process is rather technical and requires knowledge of JavaEE.

ORACLE POLICY AUTOMATION

Oracle Policy Automation is a suite of software components for modeling and deploying business rules within enterprise applications. The product was designed to transform legislation and policy documents into executable business rules. It was initially developed for and sold to the public sector, but is now used in other industries (Garlick, 2008).

The Policy Automation suite consists at least of Policy Modeling and the Policy Automation runtime. Other components for specific purposes, like mobile, do exist but are not discussed in this thesis. Policy Modeling is a Windows desktop application that serves as the rule development environment. It helps transforming policy documents into executable rule models, using Word and Excel. The Policy Automation runtime is responsible for the actual rule execution and consists of the following three components:

- Web Determinations: An interview application that uses screens, rules and flows defined in Policy Modeling to deliver web-based interactive assessments. Data entered is used in combination with backward chaining to determine which screens need to be shown to the user in order to reach a decision.
- Determinations Server: A high performance SOAP-based web service for fully auditable determinations and calculations. By passing data to Oracle Determinations Server, and receiving responses in return, enterprises can integrate rule-based decision-making with other applications.
- Determinations Engine: A Java and .NET API for low-level control over integration of decision-making into applications. It is used by both Web Determinations as Determinations Server.

Rule authoring

From a business user's perspective Policy Modeling is the most important part of Policy Automation, it is the part where the rules are actually written. As mentioned above, Policy Modeling is a standalone Windows desktop application with a regular installation process that is familiar to every Windows user.

The user interface of Policy Modeling (Figure 12) is designed for people with a business or legislation background, who do not have technical or programming skills. The rules themselves are not created in Policy Modeling, but in Microsoft Word and Excel. The rational behind this choice is to have a rule authoring environment that feels familiar to the intended end user, a business specialist.

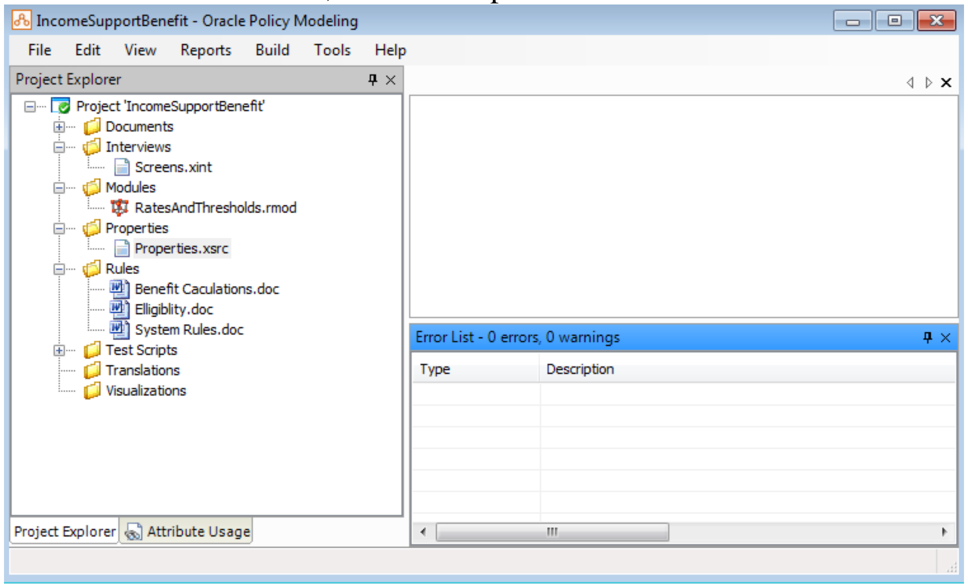


Figure 12 Oracle Policy Modeling's user interface

The typical flow of a project using Oracle Policy Automation is as follows. Policy modeling is used to create the rule base. The rule base is then deployed to the Policy Automation runtime, using either Web Determinations or Determinations Server. The application using the business rules communicates via the runtime engine, using either SOAP or the Java or .NET API. This flow is reflected in Policy Automation’s architecture shown in Figure 13.

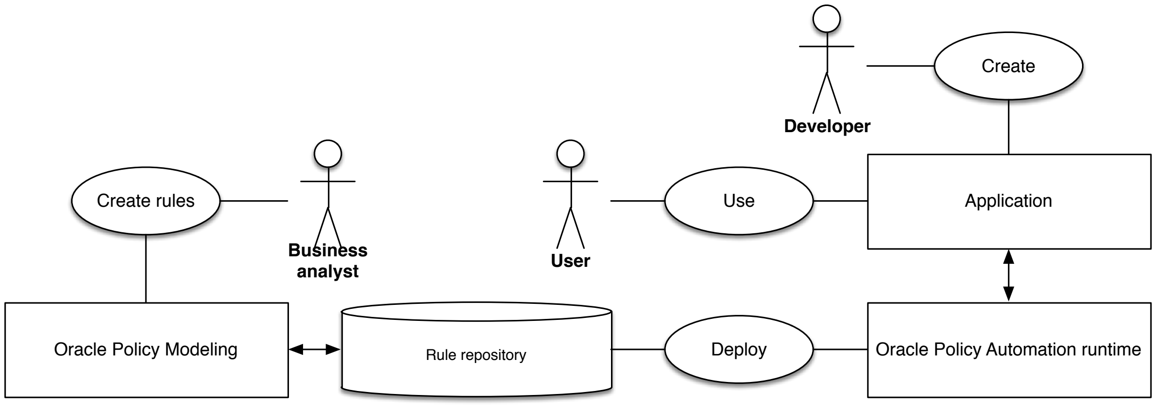


Figure 13 Oracle Policy Automation architecture

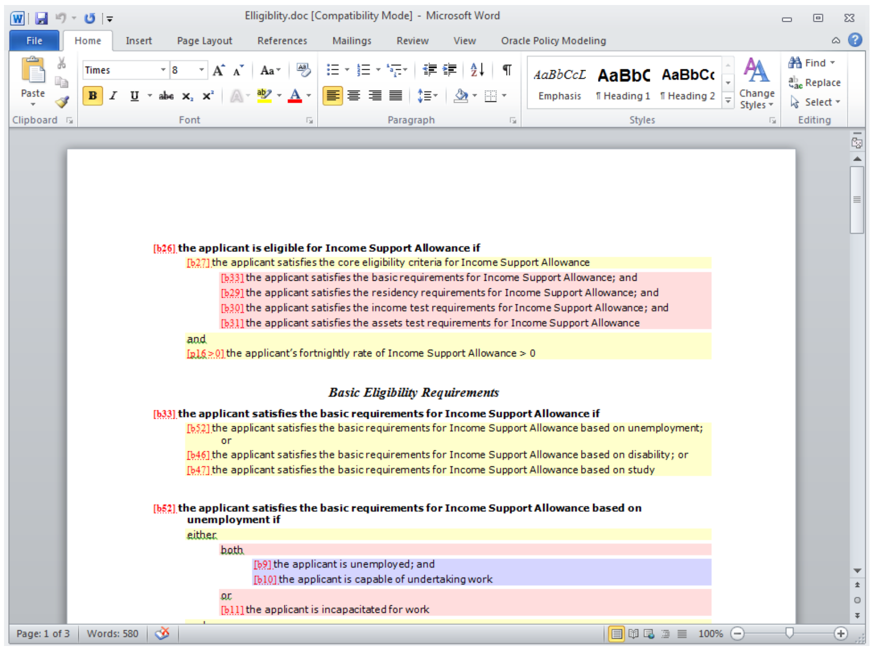
Unlike other BRMS's that require the user to write rules in a structured syntax, Policy Automation allows the user to write rules in plain English. At this moment Policy Automation is the only BRMS that is able to generate executable logic from business rules expressed in natural language, although Blaze Advisor has a syntax resembling natural language.

Policy Automation requires rules to be written starting with the conclusion and then the conditions. This style adheres perfectly with the style proposed by Morgan (2002). The example rule stated at the end of 6.2 will therefore look almost identical in Policy Automation:

Someone is eligible for Income Support Allowance if
Someone satisfies the basic requirements for Income Support Allowance; and
Someone satisfies the residency requirements for Income Support Allowance

After a rule is typed, the rule needs to be compiled. Policy Automation then parses the rule and automatically generates an ontology and data model. There is no need to define an object model beforehand and it does not require any programming. This can all be done from within Microsoft Word (Figure 14). Writing just one rule in Word would be enough to deploy the rule base. As the number of rules grows or the rules become more complex relying on automatic generation of the data model is not sufficient. By default everything becomes a boolean. Consider the following rule: *the applicant is an adult if the applicant is aged 18 years or over*. Policy Automation would create a model containing a boolean called *the applicant is aged 18 years or over*. The concept of age does not exist and must be created manually. The process of adding new properties to the ontology is simple and non-technical, only a name and type are required. Once the attribute is defined it is still necessary to explicitly describe the relation between the boolean *the applicant is aged 18 years or over* and the attribute *age*. This is done by the following rule: *The applicant is aged 18 years or over if the applicant's age >= 18*. This rule would be unnecessary if the first rule were rewritten as *the applicant is an adult if the applicant's age >= 18*. This is, however, somewhat less readable. It is probably preferable to have date of birth as input instead of an age. This is a similar process. A data of birth attribute has to be created and the relationship between date of birth and age can be defined with the following rule: *the applicant's age = YearDifference(the applicant's date of birth, CurrentDate())*. This rule shows that not everything can be written in plain English, this is especially the case when dealing with calculations. Although dealing with calculations is very unintuitive, it is clearly and thoroughly documented, as is the complete product. Even though a BRMS can be designed with business people in mind, to be able to use to its fullest is a steep learning curve. It is therefore a good idea to invest in training future users.

When dealing with a large number rules that deal with numbers, it is often clearer to represent them as a rule table. Consider for example a risk assessment for a car insurance company. To calculate a risk score multiple combinations of factors could result in a specific risk score. A possible rule could be *the risk score is 10 if the person is male and is aged below 25 and lives in a city*. If there are many rules like this it is much clearer to represent them in a rule table. In Policy Automation rule tables can be created in Excel. The process of compiling rules from the rule tables is similar to writing rules in Word. Simple rule tables can also be written in Word.



[b26] the applicant is eligible for Income Support Allowance if
[b27] the applicant satisfies the core eligibility criteria for Income Support Allowance
[b33] the applicant satisfies the basic requirements for Income Support Allowance; and
[b29] the applicant satisfies the residency requirements for Income Support Allowance; and
[b30] the applicant satisfies the income test requirements for Income Support Allowance; and
[b31] the applicant satisfies the assets test requirements for Income Support Allowance
and
[p16 > 0] the applicant's fortnightly rate of Income Support Allowance > 0

Basic Eligibility Requirements

[b33] the applicant satisfies the basic requirements for Income Support Allowance if
[b52] the applicant satisfies the basic requirements for Income Support Allowance based on unemployment; or
[b46] the applicant satisfies the basic requirements for Income Support Allowance based on disability; or
[b47] the applicant satisfies the basic requirements for Income Support Allowance based on study

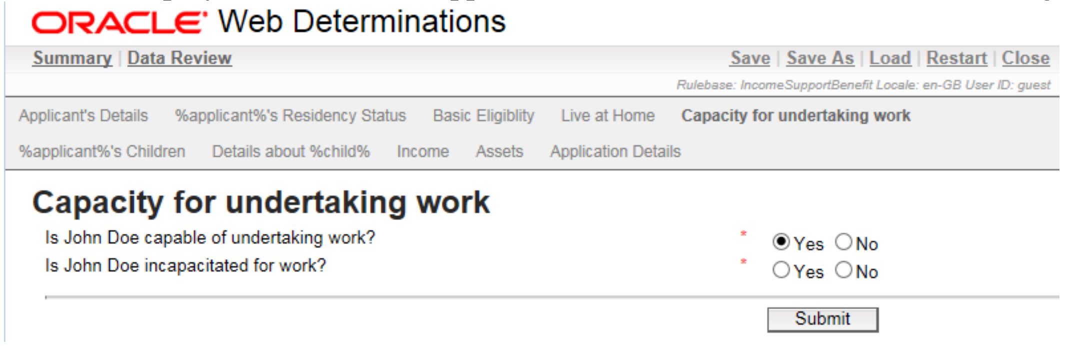
[b52] the applicant satisfies the basic requirements for Income Support Allowance based on unemployment if
either
both
[b9] the applicant is unemployed; and
[b10] the applicant is capable of undertaking work
or
[b11] the applicant is incapacitated for work

Figure 14 Rule authoring in Word

To test the rules created, Policy Modeling offers the possibility to create test scripts. This is also done is a very non-technical way. When you create a new test case, Policy Modeling shows all the attributes that be entered and all the attributes that can be inferred. By simple setting values to the entered attributes and setting expected values to the inferred attributes you create a test case that shows whether your expected values match the actual inferred values.

Rule deployment

Deploying your rules to be used in an actual application can be done from within Policy Modeling. Rules can be deployed in two ways, using Web Determinations or Determinations Server. As mentioned earlier, Web Determinations is an interview application that uses screens, rules and flows defined in Policy Modeling to deliver web-based interactive assessments. When you choose to deploy using Web Determinations a war-file is generated containing the complete interview application. WAR stands for Web Application Archive and is a standard used to package web applications written in Java. This war-file can be deployed in any standard Java application server, e.g. Oracle WebLogic Server, IBM WebSphere AS and Apache Tomcat. Policy Automation comes with an embedded Tomcat that can be used directly from within Policy Modeling. After the war-file is deployed the interview application can be started in a web browser (Figure 15).


ORACLE Web Determinations

Summary | Data Review — Save | Save As | Load | Restart | Close

Rulebase: IncomeSupportBenefit Locale: en-GB User ID: guest

Applicant's Details | %applicant%'s Residency Status | Basic Eligiblity | Live at Home | Capacity for undertaking work | %applicant%'s Children | Details about %child% | Income | Assets | Application Details

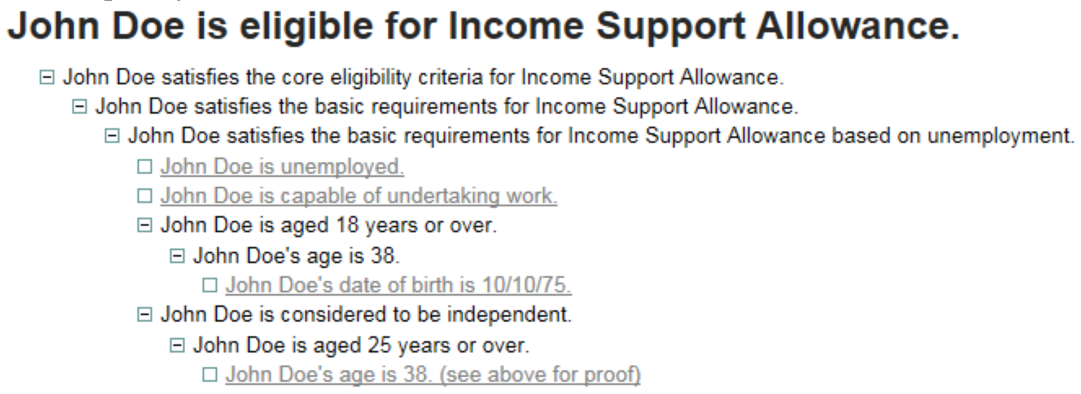
Capacity for undertaking work

Is John Doe capable of undertaking work?	*	[✓ Yes] No
Is John Doe incapacitated for work?	*	Yes No

Submit

Figure 15 Web Determinations interview

This application asks only the questions it needs to have answers for to reach a conclusion using backward chaining. When a conclusion is reached, a screen is able to show exactly how that conclusion was reached (Figure 16). Using web standards, it is possible to completely customize the look and feel.


John Doe is eligible for Income Support Allowance.

- John Doe satisfies the core eligibility criteria for Income Support Allowance.
  - John Doe satisfies the basic requirements for Income Support Allowance.
    - John Doe satisfies the basic requirements for Income Support Allowance based on unemployment.
      - John Doe is unemployed.
      - John Doe is capable of undertaking work.
      - John Doe is aged 18 years or over.
        - John Doe's age is 38.
          - John Doe's date of birth is 10/10/75.
      - John Doe is considered to be independent.
        - John Doe is aged 25 years or over.
          - John Doe's age is 38. (see above for proof)

Figure 16 Web Determinations explanation

The other way to deploy the rule base is using Determinations Server. This option will also generate a war-file, not containing a interview application, but a SOAP-based web service. This is the preferred method when integrating applications with Policy Automation as their BRMS. The WSDL's describing the SOAP services are automatically generated, making it straightforward to send and receive SOAP calls. There different SOAP calls available. There is an operation to assess a specific attribute by sending it known values, an operation to show all inferable attributes, operations that give interview like responses like Web Determination and several others with specific functions. Using the assess operation it was easy to receive inferred attributes as response by supplying known values in the request.

IBM ILOG JRules

ILOG JRules is a BRMS by IBM. ILOG refers to the name of the company that originally developed the BRMS. The J in JRules refers to Java, this shows that this system is bound to be running in a Java Virtual Machine. ILOG JRules provides separate environments for (Java) developers and business analysts, called Rule Studio and Rule Team Server respectively. These two environments are part of the total of four modules that ILOG JRules offers:

- Rule Studio: an integrated development environment (IDE) for rule applications. It provides development tools with which one can write, test and deploy business rules, and also create and manage the rule vocabulary, business and execution object models and the mappings between them. Rule Studio supports deploying and debugging rulesets to the Rule Execution Server and enables collaboration with business rule authors through its integration with Rule Team Server.
- Rule Team Studio (RTS): a web environment for authoring, managing, validating, and deploying business rules.
- Rule Execution Server (RES): a J2SE and J2EE-compliant execution environment for deploying business rule SOA services to web and application.
- Rule Scenario Manager (RSM): a testing environment that provides ruleset testing and business simulation capabilities. It provides an integrated environment in which the correctness of rules can be verified and wherein changes in business policy can be simulated.

These components are very similar to the components that make up FICO Blaze Advisor. This similarity is also apparent when you compare the two system's architecture. Figure 17 shows JRules' architecture.

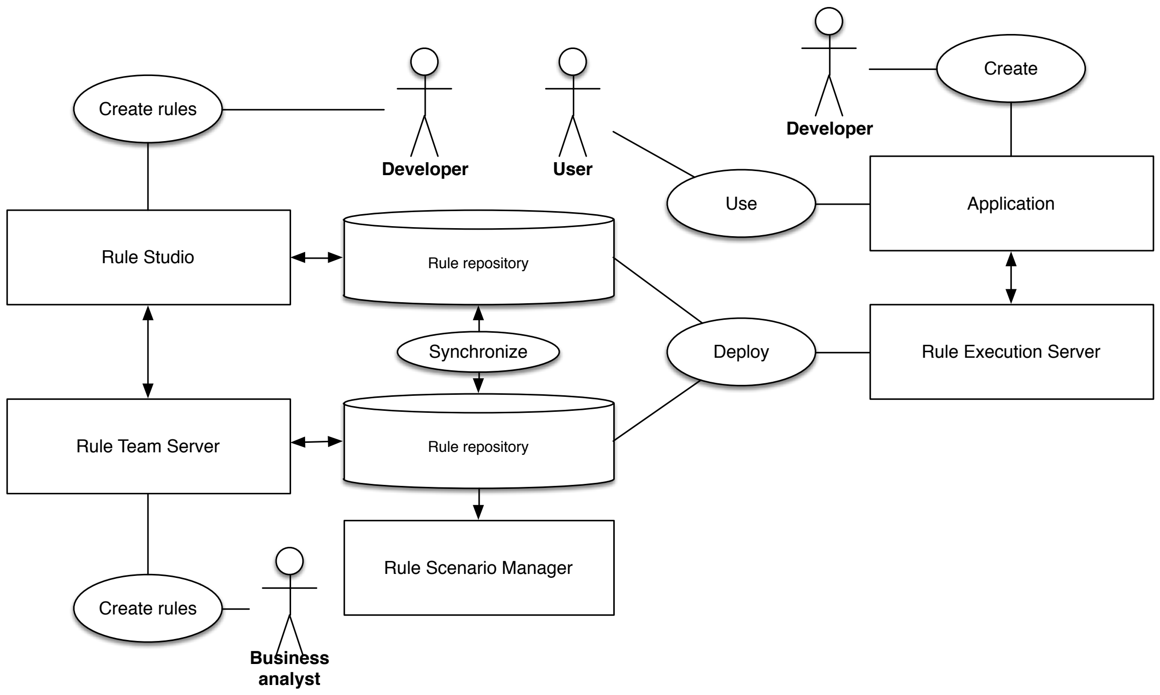
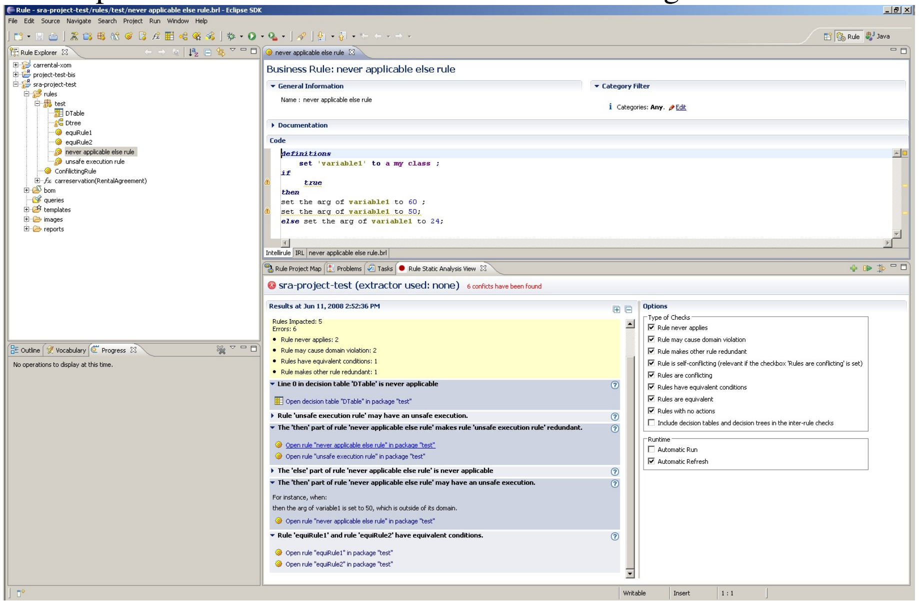


Figure 17 IBM ILOG JRules architecture

Rule authoring

Just as Blaze Advisor, ILOG JRules uses an Eclipse-based rule authoring environment called Rule Studio. Rule Studio runs as an Eclipse plug-in so enabling the developer to create standard Java and business rules within a single Integrated

Development Environment and stored in a single source code control repository.


Rule - sra-project-test/rules/test/never applicable else rule.brl - Eclipse SDK
File Edit Source Navigate Search Project Run Window Help
Rule Explorer
carrental-xom
project-test-bis
sra-project-test
rules
test
DTable
Dtree
equiRule1
equiRule2
never applicable else rule
unsafe execution rule
ConflictingRule
carreservation(RentalAgreement)
bom
queries
templates
images
reports
Outline
Vocabulary
Progress
No operations to display at this time.
Rule
Java
never applicable else rule
Intellirule
IRL
never applicable else rule.brl
Rule Project Map
Problems
Tasks
Rule Static Analysis View
Writable
Insert
1 : 1

Business Rule: never applicable else rule

General Information

Name : never applicable else rule

Category Filter

Categories: **Any**. Edit

Documentation

Code

```
definitions
    set 'variable1' to a my class ;
if
     true
then
set the arg of variable1 to 60 ;
set the arg of variable1 to 50;
else set the arg of variable1 to 24;
```

sra-project-test (extractor used: none) 6 conficts have been found

Results at Jun 11, 2008 2:52:36 PM

Rules Impacted: 5
Errors: 6

- Rule never applies: 2
- Rule may cause domain violation: 2
- Rules have equivalent conditions: 1
- Rule makes other rule redundant: 1

Line 0 in decision table 'DTable' is never applicable

Open decision table "DTable" in package "test"

Rule 'unsafe execution rule' may have an unsafe execution.

The 'then' part of rule 'never applicable else rule' makes rule 'unsafe execution rule' redundant.

Open rule "never applicable else rule" in package "test"

Open rule "unsafe execution rule" in package "test"

The 'else' part of rule 'never applicable else rule' is never applicable

The 'then' part of rule 'never applicable else rule' may have an unsafe execution.

For instance, when:
then the arg of variable1 is set to 50, which is outside of its domain.

Open rule "never applicable else rule" in package "test"

Rule 'equiRule1' and rule 'equiRule2' have equivalent conditions.

Open rule "equiRule1" in package "test"

Open rule "equiRule2" in package "test"

Options

Type of Checks
- ✓ Rule never applies
- ✓ Rule may cause domain violation
- ✓ Rule makes other rule redundant
- ✓ Rule is self-conflicting (relevant if the checkbox 'Rules are conflicting' is set)
- ✓ Rules are conflicting
- ✓ Rules have equivalent conditions
- ✓ Rules are equivalent
- ✓ Rules with no actions
- ☐ Include decision tables and decision trees in the inter-rule checks

Runtime
- ☐ Automatic Run
- ✓ Automatic Refresh

Figure 18 ILOG JRules Rule Studio

Before writing the first rule an object model needs to be created first. This can be done by defining a business object model (BOM) or by importing Java objects. When choosing to import existing Java objects, Rule Studio lets you select verbalizations for the classes' properties. This verbalization step enables BOM classes to map natural language syntax to the underlying application objects, which are in Java or XML. For example, when a class Applicant is imported as a BOM, verbalization enables the user to define an template syntax for setting and reading its properties so that this syntax can be used later when writing rules. For the Applicant's age property the 'setter' could be verbalized as 'set the age of {this} to {value}', where '{this}' is a placeholder for an instance of a Applicant object. When the necessary BOM's are created rule sets can be created. A rule set is an executable package that includes rule artifacts and non-rule artifacts. It contains a set of rules that can be executed by the rule engine. A rule set has parameters that need to be declared. Rule set parameters are part of the design of the project because they define the data that is sent to the application and the type of information that can be retrieved. They are equivalent to Java method parameters, they describe what can be used when writing rules. There are three types of parameters: IN, OUT, and IN_OUT. In the sample application used in this thesis applicant is an example of an IN_OUT parameter and support allowance of an OUT parameter. Rules can then use these parameters to manipulate objects passed to the rule engine. In ILOG JRules, rules are organized into rule packages that contain related rules. A ruleflow diagram defines the flow through these rule packages.

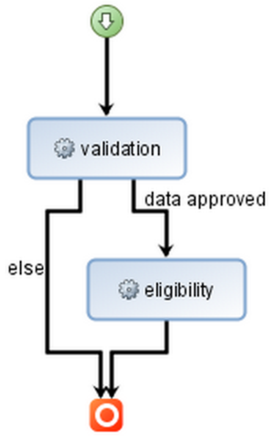


Figure 19 A ruleflow diagram

All steps described here are required before writing even a single rule. The skills needed the go through these steps are probably beyond those of a typical business analyst. Also, the actual rule authoring is not straightforward for non-technical users. ILOG JRules offers several rules languages to choose from when writing rules:

- Business Action Language (BAL): a general-purpose business rule language with syntax close to natural language. BAL is designed to be the standard language when writing business rules.
- Decision tables: these are rules composed of rows and columns used to lay out all possible situations in tabular form.
- Decision graphs: a graphical representation of decision tables.
- ILOG Rule Language (IRL): this is the language that can be directly executed by the rule engine. The syntax is similar to Java and is aimed towards developers. Other rule languages are translated to IRL.

Rules written in BAL are in the form of *if <condition> then <action>*. Because of the verbalization this results in easy readable sentences. Also, the editor has a great autocomplete feature that suggests all possibilities.

Although business users can probably use some parts of Rule Studio, it is mainly intended for developers. ILOG JRules does include RTS, which is more geared towards business users. RTS serves as a designated web-based workspace where business users can work collaboratively to author, edit, organize, and search for business rules. RTS has its own DBMS-based repository. New projects are published to RTS from Rule Studio through the RuleSync feature. Then RuleSync keeps the two versions of the project synchronized on demand. RTS serves as a designated workspace where business users can work collaboratively to author, edit, organize, and search for business rules. With RTS you can access all facets of the rule project, but in a more non-technical environment. This is environment is especially suitable for adjusting existing rules and making new rules in existing rule packages using existing BOM's.

Rule deployment

A complete ruleset is deployed to the Rule Execution Server (RSE). In the same way that Java classes are packaged in a JAR file, a ruleset is packaged in a JAR file and contains all that is necessary for execution. Deployment is done from within Rule Studio using a wizard. As most technical work is already done during the creation of the rule project, this wizard is very basic. After wizard is completed the rule project is deployed to the RSE. The RSE can be accessed using either SOAP, or pure Java. A WSDL is automatically generated, making it simple to create an application that uses the rule project via SOAP.

SUMMARY

This section described the observations while evaluating the use case in three different BRMS's. These systems have different approaches to rule implementation and differ in the type of user they appeal to.

Blaze Advisor has a wide feature range and integrates well with other FICO products. The Blaze Advisor IDE is very developer oriented. Rule sets are created in a with a Blaze Advisor plug-in extended version of Eclipse, an open source Integrated Developer Environment, which is very popular among Java programmers. Before creating a new rule set an object model must be created first. This process consists of defining entities with properties of a specific type. The rules are defined in a structured rule language, all rules are written in the following form: *if condition(s) is true then perform action(s)*. All this makes that creating a new rule set in Blaze Advisor will not come easy to someone without a developer background. However, the structured rule language is very readable and easy to learn, making rule authoring by business user possible to some extend. An aspect of Blaze Advisor that appeals more to business people is its ability to create web-based rule editing environments, in which rules can adapted and published directly into a live environment.

Policy Automation clearly stands out by its natural language approach to rule authoring. Rule sets are created in Microsoft Word and rules are written in a controlled natural language. A domain model does not need to be defined beforehand, but can be generated automatically based on a rule document. This makes it very easy for non-technical users to create rule sets. Although the natural language processing works surprisingly well in most cases, the freedom of natural language also comes with a downside. With more specific rules it can be challenging to formulate them in a way that Policy Automation interprets them as intended. So a strict syntax does have its benefits. Another strong point of Policy Automation is its so-called web determinations feature. This is an interview module that presents an end user a web-based questionnaire based on the rule base. After the user enters data, backward chaining is used to determine the next questions. After all the questions needed to reach a conclusion are answered, this conclusion is shown. Also the rules triggered to come to this conclusion are shown.

Just as Blaze Advisor, JRules uses an Eclipse based rule authoring environment. The approach to rule authoring is also very similar to Blaze Advisor. First an object model must be created, then parameters of the rule set must be declared. This is a very technical process geared towards Java developers. Although the business rule themselves are understandable by non-technical people, overall JRules is very developer-oriented, even more so than Blaze Advisor.

CONCLUDING REMARKS

In this chapter a realistic use case was implemented in three BRMS's with two goals in mind: To extend the requirements list mentioned above and to gain more insight in the implementation process.

Based on literature study in the earlier chapters of this this thesis section 3.7 and 4.4 listed a set of requirements a BRMS has to fulfill. Evaluation in this section led to some extra requirements and guidelines that will be discussed in detail in the next chapter.

The steps in this evaluation showed that implementation consists of several steps. It starts with making clear which rules need to be included in the system, this means externalizing them and expressing them in a uniform way. The actual implementation of the rules in the BRMS consists of defining a model, insert the rules into the system and deploy the rule base as a service.

7. RESULTS

This thesis discussed BRMS's in terms of how they fit in the context of historic and related developments, in terms of suitability, in terms of architecture, and in terms of practical use. This was done to gain insight in the requirements to these systems and achieve a set of guidelines for implementation. This chapter presents the results with regard to these two topics.

7.1.Requirements

As shown in section 3.2, to qualify as a BRMS a system must have at least the following components: A rule base, a rule engine, a rule authoring environment and a communication interface.

But to have a chance at being successful in solving the problems a BRMS is supposed to solve these components each have to fulfill their own set of requirements. Section 6.1 included a list of general requirements based on findings from literature study. Implementing a use case in multiple BRMS also led to some more specific requirements.

It was found that the rule language used by the BRMS's differed in to what extend they are usable by non-technical people. Some rule syntaxes tried to use a middle ground between being usable by business people and technical people. This ended up in being less usable for both types users. It is therefore recommended to have a rule syntax geared towards a specific user type.

Another big difference is the way in which an ontology is created. This is a very explicit and technical process in Blaze Advisor and JRules, while in Policy Automation the ontology is generated automatically when writing rules. The latter approach is far more suitable for business users.

When authoring rules in Blaze Advisor and JRules it was found that live syntax checking was an invaluable feature. This was lacking in Policy Automation, because of the non-formal syntax.

All three BRMS's use a specific way to store rules internally. JRules use IRL for this purpose, which can also be used by the end user. Blaze Advisor and Policy Automation hide their internal representation from the end user. Whether hidden or not, it is important to use a separate internal rule representation to provide different means of rule authoring and to integrate with other systems.

The complete list of requirements that follows from this and section 6.1 is listed below. The requirements are separated by the BRMS component to which they apply.

The rule base must

- store rules in a representation that is cleanly separated from how the rules are expressed to users.

The rule engine must

- provide forward and backward chaining capabilities.
- be rete-based.
- have conflict resolution capabilities.
- be able to show the reasoning steps towards a conclusion.

The rule authoring environment must

- provide means to build an ontology.
- provide different means of rule representation.
- have a rule syntax geared towards a specific user type.
- Include live syntax checking.

When the rule author is a business user

- be designed to be useable by non-technical people.
- use a rule language resembling natural language.
- generate an ontology automatically.

The communication interface must

- be able to function as a service within a service oriented architecture

When the BRSM's from section 6.3 are held against these requirements all three systems fulfill most requirements, but differ in the way they implement them. The main differentiation lies in the rule authoring environment. Oracle Policy Automation clearly focuses on business people as the end user by having rules expressed in natural language using familiar tools like Word and Excel. On the other end ILOG JRules is more developer-oriented. It has a strict rule language and configuration and setup requires in-depth technical knowledge, specifically Java knowledge. FICO Blaze Advisor is somewhere in the middle of this spectrum. The setup, configuration and deployment of rules are quite technical, but the rules maintenance application feature makes simple changes incredibly easy. The choice between BRMS's mainly boils down to the type of organization. Oracle Policy Automation provides the best solution for organizations that wish to use business users in rule creation and maintenance. ILOG JRules is only suitable for organizations with a developer-centric culture that do not wish to include business users in rule authoring. FICO Blaze Advisor is suitable for organizations that are somewhat developer-centric, but that do place rule maintenance in the hands of business users.

7.2.Guidelines

The process of selecting and implementing the appropriate system is a complex one, with many aspects to consider. Throughout this thesis several positive aspects of BRMS's are mentioned, like faster response to change, lower cost of maintenance, better compliance with policies and regulations, the business is control of the business rules. But as is shown in chapter 5, these benefits do not always hold up in practice. This suggests a need for a set of guidelines when selecting and implementing a BRMS. The findings in this thesis resulted in the following guidelines:

- Do not expect business users to specify rules all by themselves.

  Experiences within CGI (chapter 5) and working with three BRMS's (section 6.3) showed that specifying rule bases is technically complex, even in systems more geared towards business users. Simple rule changes are possible, but it is too positive to think that business users are able or willing to set up and maintain the entire rule base.

- The type of BRMS that is suitable depends on the type of organization.

  Different BRMS's have different degrees of technical complexity for authoring rules (section 6.3). ILOG JRules is more suited in a developer centric organization, Policy Automation in a business centric organization and Blaze Advisor somewhere in between.

- Think about architectural consequences beforehand. What rules go in the BRMS's and how do you integrate with custom build software?

  It is mentioned that the only right way to implement a BRMS is as a service (section 4.1). But this has consequences for your existing and future system's architecture.

- Invest in training developers and business users, preferably separately.

  BRMS's are mostly unfamiliar to the intended end users, business people and developers alike (chapter 5).

- Include a specialized rule author in your team, with in-depth knowledge of the BRMS authoring feature.

  The previous chapter showed the complexity of implementing a complete rule base. This should be done by someone who is familiar with the system.

- Always write rules in a completely unambiguous way.

  Section 2.1 and 3.6 showed that rules must stated unambiguously.

- When the rule language of the used BRMS supports multiple styles of representation, make a choice and use it consistently.

Choose a style that conforms to type of organization and rule author. Switching between styles is confusing (section 6.3).

- Be aware that the flow of rule firing is implicit and therefore hard to understand and debug.

  Because of the declarative nature of business rules there is no predefined order in which the rules are fired (section 3.2).
- Pay sufficient attention to testing your rule base

  Because of the implicit flow testing is extra important (section 6.3).
- Make sure the rules that are supposed to be in the BRMS are known before implementation.

  Section 3.5 discusses the process of extracting business rules and shows that this in itself is a complex operation. This is therefore a separate phase that should not be combined with implementation of a BRMS.

Together, these guidelines help to make the process of implementing run more smoothly.

8. CONCLUSION

In this research business rules management systems were covered based on a literature study, experiences on actual usage in projects and a use case was implemented in three systems. This chapter discusses the findings in this thesis and provides recommendations for future research.

8.1.Findings

In order to answer the main research question, this section provides answers to the research questions posed in section 1.2.

WHAT COMPRISES A BUSINESS RULE?

Although everybody has a sense of what a business rule is, the exact meaning slightly differs throughout literature. The definition used in this thesis is the one proposed by Graham (2006) mentioned in section 2.1. There are different types of business rules and the classification scheme proposed by the Business Rule Group distinguishes *structural assertions, action assertions, and derivations*.

HOW DO BUSINESS RULES MANAGEMENT SYSTEMS RELATE TO TRADITIONAL KNOWLEDGE SYSTEMS?

Section 3.1 showed that the roots of business rules management systems can be traced back to the 1970's. Some of the expert systems of those days were already based on the same core principles as BRMSs: rules were separated from control logic and rules were written entirely declarative. In this sense, BRMSs can be regarded as a product of developments in the field of expert systems. What sets these systems apart from traditional software is the ability to of having some explicit representation of the knowledge in the system. In the case of a BRMS, this representation is formed by the business rules.

How do you recognize and extract (externalize) the business rules to be implemented in the business rules management system?

An important part of implementing a BRMS is capturing the rules that should be included in the system. This process of knowledge acquisition is not trivial as rules may be found throughout the organization and are often buried somewhere in the source code of existing systems. Section 3.5 discusses attempts that have been made to assist in extracting business rules from existing software by partly automating this process. This can be done by interpreting transaction logs, or actually scan the source code itself using a set of heuristics to detect business rules. Helpful as these automated techniques may be, it is impossible to capture all business rules this way. There will always be need for manual code interpretation. And clearly rules can also be elicited from documents and people.

When is a business rules management system suitable?

The possible benefits of BRMS's are mentioned early on in this thesis. They include faster adaption to change, change rules by business users and having consistent rules throughout the complete business application landscape. However, a BRMS is not a great solution per se. They are especially suitable when the amount of business rules in an organization is enormous, rules change often, validity of rules needs to be proven, or the same rules are shared between multiple applications. Section 3.3 includes a complete list of indicators of suitability of a BRMS.

How do business rules management systems fit in the context of other related developments?

BRMSs do not live in a vacuum; they are closely related to other developments which are covered in chapter 4. One of these developments is service oriented architecture, in which systems are composed of multiple independent components with their own function that communicate in a standardized way. Because these components have a specific purpose and a standard interface, they are reusable by definition. Ideally, business rules should also be reusable and independent of software platforms and should therefor be one of those components within a system.

The business rules in a BRMS need to be understood by it. This means the concepts used in the rules and relationships between them must be specified. This is called an ontology and this is what gives rules meaning.

The last related technology mentioned in this thesis is compliance. This means most of all abiding by the laws and rules that apply to the organisation, but also being to proof that you do. As software systems are more and more crucial to organizations, there is a growing need for systems to be compliant. Because of the externalized rule base within a BRMS, it can play an important role in achieving compliance, as the rules are easily auditable.

WHAT ARE THE REQUIREMENTS OF A BUSINESS RULES MANAGEMENT SYSTEM IN A GIVEN SITUATION?

A BRMS must consist of a rule base, a rule engine, a rule authoring environment and a communication interface. Each these components must fulfill a set of requirements that are listed in 7.1. For requirements that concern the rule authoring environment a distinction must be made based on the technological skills of the end user.

WHAT DOES THE PROCESS OF IMPLEMENTING A BUSINESS RULES MANAGEMENT SYSTEM TYPICALLY LOOK LIKE?

As stated in 6.3, implementation consists of several steps. It starts with making clear which rules need to be included in the system, this means externalizing them and expressing them in a uniform way. The actual implementation of the rules in the BRMS consists of defining a model, insert the rules into the system and deploy the rule base as a service.

In conclusion, BRMS's show many benefits and when done right is a massive improvement over the traditional approach towards business rules in software. However these benefits do not always hold in practice. Mainly because the promise that business users can write their own business rules is often unrealistic. The other problem is difficult integration with existing systems. These problems show the need for implementation guidelines which are offered in this thesis in section 7.2.

8.2.Recommendations

As mentioned, the argument that BRMS's will allow business users to specify and maintain the business rules themselves is often not completely valid. Only Oracle Policy Automation provides a rule authoring environment that requires very little technical skills to set up the required models and write business rules completely from scratch. FICO Blaze Advisor and especially IBM ILOG JRules provide too little separation from their underlying technology to be usable and appreciated by business people.

With regard to business rules syntax, Oracle Policy Automation's is the most impressive. But its completely unrestricted syntax does have its limitations. Too much freedom in a rule language can be counter-productive. As long as natural language understanding is imperfect, a stricter natural language based language is probably preferable.

It needs to be clear that that setting up a complete rule base is inherently a process that involves numerous technical tasks. The focus should therefore lie on separating the inherently technical tasks from the non-technical task and creating completely separated environments.

8.3.Future work

One of the drivers for the development of BRMS's is the assumption that business users want to write their own business rules. It is interesting to investigate to what extent business users want full control of the business rules. It could be that in practice most business users do not want to write business rules any more than they want to write code. It that case the value of a BRMS comes only from the separation of business rules and application logic and the fact that it allows business users and IT to look at the same thing and both understand it, which on its own could be of immense value.

This thesis showed that the type of rule languages is an important decision. Business users are served better by a syntax resembling natural language, whereas technical users prefer a more formal syntax. It would therefore be interesting to research the feasibility of a rule language that is translatable to both forms.

Section 3.5 showed that extracting business rules is a complex process in itself, but mentioned multiple methods for automatically discovering business rules. Future should focus on transforming these extracted rules to the format used by a BRMS.

9. REFERENCES

Appleton, D. (1984). Business rules: The missing link. *Datamation , 30* (16), 145-150.

Arsanjani, A. (2001). Rule object 2001: A pattern language for adaptive and scalable business rule construction. *Proc. of the 8th Conference on Pattern Languages of Programs (PLoP)*, (pp. 370-402).

Australian Government, Department of Human Services. (n.d.). *Low income supplement*. Retrieved 5 15, 2015, from Human Services: http://www.humanservices.gov.au/customer/services/centrelink/low-income-supplement

Boisot, M., & Canals, A. (2004). Data, information and knowledge: have we got it right? *Journal of Evolutionary Economics , 14* (1), 43-67.

Bose, I., & Mahapatra, R. (2001). Business data mining—a machine learning perspective. *Information & management , 39* (3), 211-225.

Cawsey, A. (1998). *The essence of artificial intelligence*. Printice Hall.

Chisholm, M., & Ross, R. (2007). *How to build a business rules engine: Extending application functionality through metadata engineering*. San Francisco,: Morgan Kaufmann Publishers.

Collins, H. (1990). *Atrificial experts: Social knowledge and intelligent machines*. Cambridge: MIT Press.

Crerie, R., Baiâo, F., & Santoro, F. (2009). Discovering business rules through process mining. In *Enterprise, Business-Process and Information Systems Modeling* (pp. 136-148). Springer.

Date, C. (2000). *What not how: The business rules approach to application development*. Addison-Wesley Professional.

Debevoise, T. (2007). *Business process management with a business rules approach: implementing the service oriented architecture*. Exeter: BookSurge Publishing.

Debevoise, T. (2010). *The past, present, and future of business rules*. Tech. rep., Innovations Software Technology Corporation.

Dlodlo, N., Hunter, L., Cele, .., Metelerkamp, C., & Bot, R. (2007). A hybrid expert systems architecture for yarn fault diagnosis. *Fibres and textiles in Eastern Europe , 15* (2), 43-49.

Durkin, J. (1994). *Expert systems: Design and development*. Prentice Hall PTR Upper Saddle River, NJ, USA.

Evelson, B., Teubner, C., & Rymer, J. (2008). *How the convergence of business rules, bpm, and bi will drive business optimization*. Forrester.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). Knowledge discovery and data mining: Towards a unifying framework. *KDD*, *96*, pp. 82-88.

Forgy, C. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence , 19* (1), 17-37.

Garlick, S. (2008, November 4). *Oracle hunts for bargains in down economy*. Retrieved April 19, 2014, from IT Knowledge Exchange: http://itknowledgeexchange.techtarget.com/

Giarratano, J., & Riley, G. (1998). *Expert systems: Principles and programming* (3rd ed.). Brooks/Cole Publishing Co. Pacific Grove, CA, USA.

Graham, I. (2006). *Business rules management and service oriented architecture: A pattern language*. John Wiley.

Graham, I. (2005). Service Oriented Business Rules Management Systems. *Whitepaper, Trireme*.

Gruber, T. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies , 43* (4-5), 907-928.

Hay, D., & Healy, K. (2000). Defining business rules, what are they really. *Defining business rules, what are they really , 3*. Business Rules Group.

Hayes-Roth, F. (1985). Rule-based systems. *Communications of the ACM , 28* (9), 921-932.

Huang, H., Tsai, W., & Subramanian, S. (1996). Generalized program slicing for software maintenance. *Proceedings of SEKE*, (pp. 261-268).

Huang, H., Tsai, W., Bhattacharya, S., Chen, X., Wang, Y., & Sun, J. (1996). Business rule extraction from legacy code. *Computer Software and Applications Conference, 1996. COMPSAC'96., Proceedings of 20th International* , (pp. 162-167).

Huber, G. (2004). *The necessary nature of future firms*. London: Stage Publications.

Joiner, K., Tsai, W., Chen, X., Subramanian, S., Sun, J., & Gandamaneni, H. (1994). Data-centered program understanding. *Software Maintenance, 1994. Proceedings., International Conference on*, (pp. 272-281).

Joosten, S. (2007). *Deriving functional specification from business requirements with ampersand*. Open University of the Netherlands. Citeseer.

Lucas, P., & Van Der Gaag, L. (1991). *Principles of expert systems*. Addison-Wesley.

Mahmoud, Q. (2006). Getting Started With the Java Rule Engine API (JSR 94): Toward rule-based applications. *Getting Started With the Java Rule Engine API (JSR 94): Toward Rule-Based Applications*.

Mercury. (2005). *Sustainable compliance, industry regulation and the role of IT-governance*. Retrieved from http://www.mercury.com

Morgan, T. (2002). *Business rules and information systems: Aligning IT with business goals*. Addison-Wesley Professional.

Negnevitsky, M. (2005). *Artificial intelligence: A guide to intelligent systems*. Addison-Wesley Longman.

Nelson, M., Peterson, J., Rariden, R., & Sen, R. (2010). Transitioning to a business rule management service model: Case studies from the property and casualty insurance industry. *Information \& management , 47* (1), 30-41.

Newell, A., & Simon, H. (1972). *Human problem solving*. Prentice-Hall Englewood Cliffs, NJ.

Ould, M. (2005). *Business process management, a rigorous approach*. Meghan-Kiffer Press.

Powell, W., & Snellman, K. (2004). The knowledge economy. *Annual Review of Sociology , 30*, 199-221.

Ross, R. (2000). Expressing business rules. *ACM SIGMOD Record , 29* (2), 516.

Ross, R. (2003). *Principles of the business rule approach*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Ross, R. (2001). The BRS rule classification scheme. *The BRS rule classification scheme*.

Rymer, J., & Gualtieri, M. (2008). *The Forrester Wave™ : Business rules platforms, Q2 2008*. Tech. rep., Forrester Research, Inc.

Sangers, H. (2008). *Achieving traceable compliance using the ampersand method*. Master's thesis, Open University of the Netherlands.

Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., van de Velde, W., et al. (1999). *Knowledge engineering and management: The CommonKADS methodology*. Cambridge: The MIT Press.

Shirai, Y., & Tsuji, J. (1982). *Artificial intelligence: Concepts, technologies and applications*. John Wiley.

Shortliffe, E. (1976). *Computer-based medical consultations: MYCIN*. Elsevier.

Singh, S. (1999). *The Code Book: The science of secrecy from ancient egypt to quantum cryptography*. New York: Anchor Books.

Smith, H., & Fingar, P. (2002). *Business process management: The third wave*. Tampa: Meghan Kiffer Press.

Taylor, J. (2005). Achieving decision consistency across the SOA-based enterprise using business rules management systems. In *Web Information Systems Engineering--WISE 2005* (pp. 750-761). Springer.

Thirumaran, M., Ilavarasan, E., Thanigaivel, K., & Abarna, S. (2010). Business rule management framework for enterprise web services. *International Journal on Web Service Computing , 1* (2), 15-29.

van Melle, W., Shortliffe, E., & Buchanan, B. (1984). EMYCIN: A knowledge engineer's tool for constructing rule-based expert systems. Addision-Wesley Reading, MA.

von Halle, B. (2001). *Business rules applied: building better systems using the business rules approach*. John Wiley \& Sons, Inc. New York, NY, USA.

von Halle, B., & Goldberg, L. (2006). *The business rule revolution: Running business the right way*. Cupertino: Happy About.

Wagner, C. (2006). Breaking the knowledge acquisition bottleneck through conversational knowledge management. *Information Resources Management Journal , 19* (1), 70-83.

Weigand, H., van den Heuvel, W., & Hiel, M. (2011). Business policy compliance in service-oriented systems. *Information Systems , 36* (4), 791-807.

Wikipedia. (n.d.). *Semantic network*. Retrieved 8 10, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Semantic_network

APPENDIX A: INTERNAL STUDY CAPACITIES CGI BRMS

Een scan op de CV's in Lara (global) op aanwezige kennis van de belangrijkste rules engines (pakketten) leverde het volgende op:

BRM-tool	Aantal CV's, waarin de tool genoemd wordt (opgeleid of een project ermee uitgevoerd)	Opmerking
Be Informed	1 (Niet volledig zeker door grote aantal overeenkomsten op informed)	Maarten Koskamp heeft bij Be-informed gewerkt Due Dilligenge uit te voeren / uitgevoerd door Joop Trouwee in Augustus 2008
Blaze	5	ABN-Amro mogelijk referentie. Lijkt er echter op dat de implementatie door de leverancier zelf is gedaan.
Cordys	8	Lijkt geen BRMS
Corticon BRMS	1	Mogelijk referentie bij Aegon in combinatie met G360 voor WFM
Easyrules	0	
Global 360 BPM Suite	2	Geen BRMS
Haley (voorheen RuleBurst)	2	Alleen in het kader van IND bid onderzocht
ILOG Jrules (Martkleider?)	14	Vooral projecten in UK, mogelijk wel referenties: • Companies House CHIPS • Orange – BSAT Revenue Protection Proof of Concept • FORD - New Generation Marketing Programme System • Automotive J2EE Marketing Application
RuleManagement	0	
RuleArts	0	
Staffware iProcess Suite TIBCO Business Studio	42 in nederland (verder nog heel veel staffware rest van de landen)	Inzet veelal niet als business rules engine, maar als WFM Vooral EUT en Telecom en Rabo ICT Ook nog veel Masterclasser en cursisten zonder projectervaring Inzet van Logica consultants vooral bij Eneco en KPN. Lijkt echter in competence centers van KPN en Eneco zelf. Refenties mogelijk: • Eneco (lijkt geen resultaat verantwoordelijke opdracht) • KPN (lijkt geen resultaat verantwoordelijke opdracht) • Rabo ICT • Informatie Beheergroep
Valens	0	
Pegasystems	1	

Belangrijkste punten:

- Kennis over BRM-tools is in Nederland beperkt.
- Redelijk wat kennis over Jrules in UK.
- ILOG Jrules, Fair Isaac – Blaze Advisor en Pegasystems zijn volgens Forrester marktleiders op het gebied van BRMS. We hebben redelijk wat kennis en mogelijk referenties van jRules, maar niet in nederland. Jrules is ook één van de Marktleiders.
- De positie van bé informed als rules engine is nog niet duidelijk. Zij timmeren wel veel aan de weg in Nederland.

APPENDIX B: DISCUSSION BOARDS

Hi everybody,
A while ago I asked about predictive analysis and later on about rule engines, all related to a pre-bid. Thanks to your helpful comments we have made great progress and right now we are writing an actual bid.

For risk analysis we have chosen JRules from ILOG for its superior management tooling, matching exactly the customer needs. I know from LARA that we have some consultants with experience with the product, but I suspect that there are many more. I don't think we ever did projects with it. I need names of people who have worked with JRules. What I want to know is the degree to which we can confidently say that we, Logica, have good experiences with JRules. (It is not about spotting consultants to engage when we get selected: that is a very different question.) Thanks in advance, -Jan

---

Hello,

I've worked with JRules in an SOA project for a french phamaceutical distribution company (OCP). I was part of the team that designed the services. Logica wasn't in charge of the project, we just had some consultants in the functional team and a couple more in the design team.

JRules was used to implement all of the business rules in the project (the services called JRules). I'm sorry to say that it was a terrible architectural choice, because implementing the rules in JRules is far more complicated than in plain old java or .net. It takes muche more effort even for an experienced developer, if you can find such a developer (ILOG consultants are very expensive). Some of the reasons are the following :
- a specific data model must be defined inside JRules and mapped to the model you use in the rest of the application
- the rule syntax is makes you develop a single very simple rule at a time (if <something> then <something else>), so for more complex business rules you actually have to develop many JRules rules

It is not true that business experts can update rules on their own as ILOG suggests in their marketing material.

This is my personal opinion and I've heard people give good feedback about JRules, so other projects might've had more success.

Rules engines need structured data in order to be able to pull attributes which the rules work on. Where the engine is well integrated it is possible to navigate objects and attributes in the designer (Hayley engine integration with Siebel has wide access to Siebel Objects for example), however where you are integrating a general rules engine such as JRules into an application, then you need to copy data/attributes into objects that Rules Engine can access. When the rules change consideration needs to be given to any new attributes needed (and have a programmer ready at hand, followed by a test and relapse cycle).

JRules has an extensive workbench and testing environment, but change control and release mechanisms mean that it is not possible that users would be allowed to directly update a live system - but it should be a relatively slick operation to deploy an new rules set as it comes out of test. The case often quoted is eBay which is a JRules user and is able to make business decisions today which are globally deployed tomorrow - enabling them to quickly respond to bad press with a statement of policy change which has been made effective world wide.

I worked on knowledge-based (rule based) systems back in the late 80's and early 90's, so I have no objection in principle to the approach, but as Andrei says the integration with conventional software is painful, and there is a tendency for people to believe vendors' claims that using rules will make everything easier and understandable by (non technical) business experts – and to use rules engines in areas where declarative coding in a normal language would be a better answer. At the other extreme, forward-chaining rules that introduce new assertions into working memory become hard to understand and debug as the application complexity increases.

A few years ago when I was working for a debt management sw product company, we used ILog JRules for building a module which is responsible to perform scoring, risk assessment, decisions on treatment pathways etc. We used JRules factory API to provide a UI in our system which allows the business experts to change their business rules. This is done by providing a layer of DSL that is well understood by the business users and that is transformed into rules and deployed. I was a key developer in this. It was quite successful after so many iterations and took us at least 1 year to get it right with help from ILog consultants.

It's been many years and I don't remember JRules that well now, but this is what I can remember about what we tried to do (some information might be outdated):

1. JRules has a very fast execution context, good performance
2. The best way to implement it is to be very specific on the data model that rules will access and modify.
3. Design everything in a way where only those that should be rule driven are exposed to rules, usually ppl think that with a rule engine one can convert all business logic to be rules driven. This is not the case. This is more specific to rules design, no JRules per se.
4. Be very domain specific and build constraints around what can the rules do and not do. Rules should only determine decisions, not manipulated and process data.
5. Although there are testing mechanisms provided by JRules, testing is still a nightmare.
6. Business experts can update the rules themselves, but you have abstract it another layer by providing your own UI which constraints the users from what they can do. Some limitations must exist.

ILog has many success stories (example from Keith about eBay) and I know that the product is good. It is more so the question of to go with rules or not to go with rules and the question of whether the implementation team is experienced in rules design or not?

Thanks for your insightful comments and the reference! I understand your objections, let me briefly discuss. It is true that the rule-engine movement in the 80's and 90's, with their knowledge based systems built with software packages like ION DS and languages like PROLOG, has come to a grinding halt. However, companies like Haley and ILOG have a different, less ambitious and far more practical approach.

What doesn't work is to put "all business rules" in one rule repository and let these execute by the rule engine. That is guaranteed to lead to a spaghetti-architecture that flies in the face of principles of modern OO-based thinking, and will lead to a host of problems. However, it is perfectly possible and even advisable to put complex rules about a specific, separate part of the business logic in a separate component, provided that one observes rules of low coupling and high coherence.

In the case of our customer, the rules are about assessing whether an object poses a risk, based on an enormous amount of information about attributes and relations. The rules are interrelated and volatile in the sense that they change often. We built a PoC in JRules, with very encouraging results. The rule-engine is only a small part of the total solution, which mainly relies on ESB and BPM-technology for workflow and the like --- I suppose that is what you mean by declarative language.

In summary, I agree with your objections but I think these apply to wrong, overly ambitious applications of the rule-engine concept. That is what in my opinion killed ION DS and can be fixed in the new approach advocated by Haley, ILOG and the like. Only use them where they add real value. Thanks again!
Just to give you a bit more background: the projects I've been involved in over the last few years where I felt the use of a business rules engine was inappropriate were (i) a project where the customer had insisted on the use of ILOG JRules for specific business logic, and (ii) a project where a technical architect had evaluated Drools (which became JBoss Rules) and found it performed well and was quite expressive, and then the developers started using rule sets for a variety of purposes. In the first project the designer who had the job of integrating the use of rules into the rest of the application found it a large amount of work for little or no benefit compared with writing the same rules in Java, and in the second project I was always sceptical about the appropriateness of the technology (but never had time to do a hands-on evaluation), and on the few occasions when I dug around in the codebase to see what kinds of rules people had come up with, I was unimpressed. Here's an example:

```
rule "Invalidate Underwriting Decision" salience 9

   when
      $ae:InvalidateUnderwritingDecisionAmendmentEvent()

   then
```

```
        /*
         *  Invalidate the underwriting decision on the product.
         */

        AmendmentHelper.logMessage("Invalidate underwriting decision for: " +
                $ae.getClient().getFirstForename() + " " + $ae.getClient().getSurname());
        AmendmentHelper.invalidateUnderwritingDecision($ae.getClient(),metaDataManag
er);

end
```

...and this is from a file containing six rules of similar form, all given different salience.

Elsewhere in the same project there was an application using rules for automated underwriting, which I would expect to be a good application of a rules engine.

Oh dear, I'm looking at this old code again and here's a horrible mixture of condition/action and procedural programming:

```
rule "AM083 / AM084 - Add / Remove CIC Rider Amend Dependencies"

    when
        $contract:ProposedTermContract()
        eval(AmendmentHelper.logMessage("Add / Remove CIC Rider Dependencies -
Remove"))
        eval(!AmendmentHelper.isFPIP($contract))
        eval(((DirtyDescriptor) $contract).isAttributeDirty("CICRiderEnabled"))

    then

    AmendmentHelper.logMessage("Firing AM083 / AM084 - Add / Removed CIC Rider
Amend Rules");

    // Clean up DirtyDescriptor to prevent re-firing during Redraws processing
    AmendmentHelper.cleanUp((ModelBaseImpl)$contract, "CICRiderEnabled");

    ProposedTermContract contract = AmendmentHelper.getGhostContract($contract,
        applicationContext, sessionContext, true, false);
    /**
     * Invalidate the quote only for the product which has had CIC changed
     */
    InvalidateProtectionQuoteAmendmentEvent ae = new
InvalidateProtectionQuoteAmendmentEvent();
    ae.setProposedContract(contract);
    ae.setIsMandatory(false);
    drools.assertObject(ae);
```

```
   /**
    * Invalidate application
    */
   InvalidateApplicationAmendmentEvent aae = new
InvalidateApplicationAmendmentEvent();
   drools.assertObject(aae);

   /**
    * Retrieve the client from the product
    */

   PersonalClient[] client =
ProductAmendmentHelper.getClient((ProposedTermContractImpl)contract);

   for (int i = 0; i < client.length; i++) {

      /**
       * Invalidate underwriting decisions for this client
       */
      InvalidateUnderwritingDecisionAmendmentEvent uwae = new
InvalidateUnderwritingDecisionAmendmentEvent();
      uwae.setClient((PersonalClientImpl)client[i]);
      uwae.setIsMandatory(false);
      drools.assertObject(uwae);

      /**
   * Invalidate UW Q&A's
   */
   InvalidateUnderwritingSessionAmendmentEvent usae = new
InvalidateUnderwritingSessionAmendmentEvent();
   usae.setClient((PersonalClientImpl)client[i]);
   drools.assertObject(usae);

      /**
       * Invalidate Customer Declaration
       * Will only happen if the item was agent amended
       */
      InvalidateCustomerDeclarationAmendmentEvent cdae = new
InvalidateCustomerDeclarationAmendmentEvent();
      cdae.setClient((PersonalClientImpl)client[i]);
      drools.assertObject(cdae);
   }

   /**
   * Check Sum Assured still valid for type of contract
```

```
    */

    AmendmentHelper.doSumAssuredCheck(contract, errors);

    /*
    *  Check TPD setting for non-MPI contracts. Ensure we only change contracts for
    *  this client and only contracts that are not already set to FATS
    */
    AmendmentHelper.doTPDChecks(contract);

     /**
     * Validate contractTerm - depends on CIC rider
     * Error msgs E87, E88, E126, E127, E127, E129, E133, E134 may be issued
     * - see helper method called.
     */
     errors.clearModelError(contract, "Policy Term");
     ProposedTermContractImpl contractImpl = (ProposedTermContractImpl)contract;
     ProductAmendmentHelper.validateTerm(contractImpl, errors);

end
```

That’s one of many in a file 1,682 lines long L

Note how the second rule is asserting events that will cause rules like the first one to fire. This is a very roundabout and computationally expensive way of making changes on a complex data structure (an application for life assurance, for multiple clients and products, decorated with pricing and underwriting decisions).

Early last year we had a customer asking how, in a proposed new development, we intended to externalise business rules. I responded with the attached slides (part of a larger set covering our technical approach). In presenting them, I stressed that I did not believe that the application we were looking would have any need for a rules engine (but our application was supposed to set the technical approach to be adopted by others, which might need one)

Some years ago a development of mine included what I still think was a good example - though surprisingly bespoke.

The application was concerned with the management of aircraft and had to implement OPDEF Rules, often known as "minimum equipment list". These are documented in a large book of regulations that tells you what combinations of faults and conditions on a given type of aircraft are sufficient to prevent it from being used on a passenger flight. For example, a large jet can be flown with faults affecting its reverse thrust and brakes - but only if it will land on a runway longer than a specified length. (That's one to encourage the nervous fliers out there.)

ILOG JRules was not available then, only its predecessor ILOG Rules. That was expensive and the rules had to be written in something like Prolog so I rejected it. What we did in the end was to write our own parser (using an offshoot of YACC and Lex) and rules engine and had the customer's staff transcribe the regulations into rules held within a spreadsheet. The bespoke part cost a lot less than an ILOG Rules licence and the thing worked well, though the application as a whole never, at it were, got off the ground. These days I would probably go for JRules if the budget allowed.

The application evaluated the relevant rules and then proceeded to a much more compute-intensive phase using these results. The efficiency of the rules evaluation was not much of a consideration. That's just as well because we had a similar experience to Justin's that the rules set up by non-programmers were pretty inefficient. (IIRC our DIY engine didn't have and didn't need rule chaining via assertions so the opportunities for inefficiency were not as rich as with a more capable engine - less can be more.)

APPENDIX C: USE CASE RULE BASE

- Candidate is eligible for the Low Income Supplement
    - IF candidate is 18 years or older
    - AND in the previous financial year, the income was below:
        - $30,000 for singles
        - OR $45,000 combined for couples
        - OR $60,000 combined for singles or couples with a dependent child
    - AND candidate did not receive Family Tax Benefit for 39 weeks or more
    - AND candidate meets the Australian residence, tax and other requirements for the Low Income Supplement
- Candidate meets the residence requirement for the Low Income Supplement
    - IF candidate is in Australia on the date of claim
    - AND in the previous financial year, candidate was in Australia for at least 46 weeks
- Candidate meets the tax requirement for the Low Income Supplement
    - IF in the previous financial year candidate had a taxable income less than $18,000
    - OR in the previous financial year candidate had a taxable income more than $18,000, but less than the Low Income Supplement threshold amount
- The Low Income Supplement threshold amount is: person's eligible tax offsets for the income year ÷ 0.15 + $18,000
- Candidate meets the other requirements for the Low Income Supplement
    - IF candidate was not in prison or a psychiatric institution for more than 25 weeks in the previous financial year
    - AND candidate or and their partner (if applicable) have not received the Low Income Family Supplement in the previous financial year