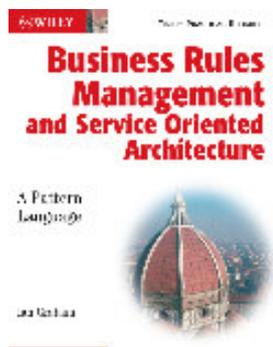# Service Oriented
Business Rules Management Systems

by

Ian Graham

Chief Technical Officer and Principal Consultant,
TriReme International Ltd.

[www.trireme.com]

Business Rules Management and Service Oriented Architecture

A Pattern Language

Published later in 2006

VERSION 2
Dated 2005-06-06

TriReme
www.trireme.com

Table of contents

1 EXECUTIVE SUMMARY

This report is a detailed business and technical comparison of three business rules management systems (BRMSs): Blaze Advisor from Fair Isaac Inc., JRules from ILOG SA and HaleyAuthority from Haley Systems Inc. The foci of the report are the business benefits that potential users may obtain, the ease of use by business analysts and users and the level of integration with the commercial and technical environment. Eight other products, eliminated from the shortlist, are covered briefly.

Businesses continue to strive for shorter time to market and to lower the cost of developing and maintaining computer applications to support their operations. Business rules management technologies can play an important rôle in this.

In the report, we examine the features and responsibilities of a BRMS and then the benefits of and business drivers for adoption of the technology. We list typical applications and indicators of the need for a BRMS. Next we cover the key technical features of a BRMS. A detailed technical appendix covering inter alia knowledge representation and inference techniques supports this. Next we compare the products in some detail. Further appendices cover the issue of methods appropriate for the development of such systems and our study method and competence.

Analysis reveals that each product is suitable for use in quite different scenarios. We therefore evaluate all three products under three different business scenarios. We conclude that HaleyAuthority provides the best solution for organizations that wish to use business analysts and users in rule creation and maintenance. This applies to a wide range of rule-based application

We were most impressed by HaleyAuthority's genuine natural language syntax for the rules and its totally transparent code generation. It also scores well in terms of price, performance and memory utilization

The strengths of Blaze Advisor were its range of features, integration with other Fair Isaac decision support products and its ability to create rule maintenance applications. Blaze Advisor also scored well for ease of learning by competent knowledge engineers and on applications such as credit scoring, with JRules looking more suitable for environments where skilled Java developers constituted the workforce (rather than business analysts or users) and beating Blaze Advisor on price and performance.

JRules we find suitable only for organizations with a developer-centric culture that will not involve users much in development or rule maintenance. We point out that such cultures are not ideal but have to admit that they do exist.

Blaze Advisor occupies the middle ground between the technical JRules and the user-focused HaleyAuthority. It is best suited for environments which are not developer-centric but where, perhaps, users are too busy to get heavily involved in rule maintenance. Again, this is an all-to-common situation that has to be recognized.

Scenario 1

Scenario 1 looks at BRMS products from a viewpoint that emphasizes the business user's and business analyst's perspective. From this point of view, the principles that such users care about are:

  # Ease of understanding.
  # Not requiring programming expertise.
  # Ease of use and expression of rules.

A multi-attribute feature analysis gave HaleyAuthority the highest score: 85% of the maximum possible score in Scenario 1. Blaze Advisor came second with 77%. JRules scored 68%.

Scenario 2

Scenario 2 maintains the viewpoint of Scenario 1 but lays greater stress on the level of integration with the commercial and technical environment. Furthermore, in this scenario, rule input by users

and non-technical business analysts is not required, because users are too busy. They will maintain rules via custom applications, where appropriate.

The multi-attribute feature analysis gave Blaze Advisor the highest score: 79% of the maximum possible score in Scenario 2. HaleyAuthority came second with 76%. JRules scored 67% in this scenario.

Scenario 3

Scenario 3 abandons the emphasis on rule creation by end users and assumes a strong commitment to a technical architecture, such as J2EE. This scenario is not explored in this version (Version 2) of this report, although *a priori* we would expect JRules to do better in this scenario.

This report does *not* conclude that any one product is better than any other. It merely sets out the relative strengths and weakness of the three products and considers their suitability for different types of application. As a consultant, I will be able happily to recommend all three products in the future – if they match the business and technical problems to be addressed and are to be deployed in an appropriate organizational culture.

1.1 Version notes

This is Version 2 of *Service Oriented Business Rules Management Systems*. It updates Version 1 by considering the recently released Version 6 of Blaze Advisor and including two new development scenarios that show off the strengths and weaknesses of the three products in different circumstances. Scenario 3 is still under development and is only sketched in this version of the report.

Version 3 of this report will complete Scenario 3 and include consideration of the latest version of JRules (5.0).

> **WATCH THIS SPACE FOR VERSION 3.**

2 INTRODUCTION

This report makes a detailed comparison of three leading business rules management systems (BRMSs): Blaze Advisor from Fair Isaac Inc., JRules from ILOG SA and HaleyAuthority from Haley Systems Inc. I compare them from both a business and technical perspective, although the focus is on the business benefits that potential users may obtain from their use.

BRMSs have the following features and responsibilities.

Storing and maintaining a **repository** of business rules that represent the policies and procedures of an enterprise.
Keeping these rules (the business logic) separate from the 'plumbing' needed to implement modern distributed computer systems.
Integrating with enterprise applications, so that the rules can be used for all business decision making, using ordinary business data.
Forming rules into independent but chainable **rulesets** and performing **inferences** within and over such rulesets.
Allowing business analysts and even users to create, understand and maintain the rules and policies of the business with the minimum of learning required.
Automating and facilitating business processes
Creating intelligent applications that interact with users through natural, understandable and logical dialogues.

The business drivers for the adoption of BRMSs are as follows.

Current software development practice inhibits the rapid delivery of new solutions and even modest changes to existing systems can take too long.
Accelerating competitive pressure means that policy and the rules governing automated processes have to be amenable to rapid change. This can be driven by new product development, the need to offer customization and the need to apply business process improvements rapidly to multiple customer groups.
Personalizing services, content and interaction styles, based on process types and customer characteristics, can add considerable value to an organization's business processes, however complex. Natural dialogues and clearly expressed rules clarify the purpose of and dependencies among rules and policies.
In regulated industries, such as pharmaceuticals or finance, the rules for governance and regulation will change outside the control of the organization. Separating them from the application code and making them easy to change is essential, especially when the environment is multi-currency, multi-national and multi-cultural.
Business rules and processes can be shared by many applications across the whole enterprise using multiple channels such as voice, web, and batch applications, thereby encouraging consistent practices.

Using BRMSs should decrease development costs and dramatically shorten development and maintenance cycles.

Typical applications of BRMS technology include these.

l Automating procedures for
  ☞ claims processing
  ☞ customer service management
  ☞ credit approval and limit management
  ☞ problem resolution
  ☞ sales
Advice giving and decision support in such fields as
  ☞ benefits eligibility

- ☞ sales promotions and cross selling
- ☞ credit collection strategy
- ☞ marketing strategy
- \# Compliance with
  - ☞ external and legal regulations
  - ☞ company policy
- | Planning and scheduling of
  - ☞ advertising
  - ☞ timetables and meetings
  - ☞ budgets
  - ☞ product design and assembly
- \# Diagnosis and detection of
  - ☞ medical conditions
  - ☞ underwriting referrals
  - ☞ fraud (e.g. telephone or credit card fraud)
  - ☞ faults in machinery
  - ☞ invalid and valid data
- \# Classification of
  - ☞ customers
  - ☞ products and services
  - ☞ risks
- \# Matching and recommending
  - ☞ suitable products to clients
  - ☞ strategies to investors

In what is probably the best and most sensible and practical book yet on business rules management, Morgan (2002) defines a business rule as 'a compact statement about an aspect of a business [that] *can be expressed in terms that can be directly related to the business, using simple, unambiguous language that's accessible to all interested parties*: business owner, business analyst, technical architect, and so on' (emphasis added). Our focus in this report will be on the ease of expression of rules and the suitability of the available products for business owners, business analysts, as well as technical features.

Business rules arise from the objects that one encounters in a business and their interrelationships. These 'business objects' may be found in documentation, procedure manuals, automation systems, business records or even in the tacit know-how of staff. Morgan identifies the following indicators of the need for a business rules management system.

Policies defined by external agencies.
- ☞ Government, professional associations, standards bodies, codes of practice, etc.

Variations amongst organizational units.
- ☞ Geography, business function, hierarchy, etc.

Objects that take on multiple states
- ☞ Order status, customer query stage, etc.

Specializations of business objects
- ☞ Customer types, business events, products, etc.

Automation systems
- ☞ Business logic embedded and hidden within existing computer systems

Defined ranges and boundaries of policy
- ☞ Age ranges, eligibility criteria, safety checks, etc.

Conditions linked to time
- ☞ Business hours, start dates, holidays, etc.

The quality manual
- ☞ Who does what, authorization levels, mandatory records, etc.

Significant discriminators

☞ Branch points in processes, recurring behaviour patterns, etc.

Information constraints

☞ Permitted ranges of values, objects and decisions that must be combined or exclude each other.

Definitions, derivations or calculations

☞ Transient specialization of business objects, proprietary algorithms, definitions of relationships.

l Activities related to particular circumstances or events

☞ Year-end, triggering events, conditional procedures, etc.

If any of these concerns are familiar, then your organization may well be a candidate for a BRMS.

The benefits of adopting a business rules management system may be summarized as follows.

\# Faster development.
\# Faster maintenance, which particularly relevant in service oriented architectures, where the maintenance of a rules component is addressed outside of the wider IT maintenance context.
\# Clearer auditability.
\# More reusable business logic.
\# Greater consistency across the enterprise.
\# Better alignment and understanding between business and IT.

We now delve deeper into the reasons for adopting BRMSs and take a hard look at the technology. This analysis will allow us to examine the products on the market and arrive a clear conclusions and recommendations.

3 WHY USE A BUSINESS RULES MANAGEMENT SYSTEM?

Repeated biennial surveys by the Standish Group since 1995 have shown that nearly two-thirds of US software development projects fail, either through cancellation, overrunning their budgets or delivering software that is never put into production. It is not incredible to extrapolate these – frankly scandalous – figures to other parts of the world. What is harder to believe is that our industry, and the people in it, can tolerate such a situation. We should be too ashamed of ourselves ever to bid for work again. The Standish surveys also looked into the reasons why people involved in the sample projects thought such projects fail so often. The reasons given – in descending order of importance – were:

- # lack of user involvement
- # no clear statement of requirements
- # no project ownership
- # no clear vision and objectives
- # lack of planning

The first four items on this list may point to the developer-centric culture of many IT development organizations, a culture highlighted by Alan Cooper (1999) and others and familiar to those of us who have worked in or with corporate IT over a long period. Too often, developers expect users to learn *their* language – often in the form of UML diagrams. In today's fast-moving competitive environment this will not work. Project teams must develop languages that can be understood by users and develops alike: languages based on simple conceptual models of the domain written in easily understood terms. Business process modelling approaches of the sort pioneered by Graham (2001) and business rules management systems both have a rôle to play in this critical challenge for IT in the 21st century.

Another 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. This is one of the reasons that object-oriented and component based development is so attractive: when the implementation of a data structure or function changes, these changes do not propagate to other objects. Thus maintenance is localized to the changed object(s). However, this benefit does not extend to changes to the business rules if they are scattered around the application or tightly bound to interface definitions. If the interface changes – as well as the implementation – the changes *will* propagate and maintenance will be very costly.

To overcome this we need to separate the definition of policy from implementation and code detail. BRMSs facilitate this. Ideally, the rules are subdivided into modules that are encapsulated in individual objects, including so-called 'blackboard' objects, which are visible to all objects that have registered an interest in them. Such blackboards encapsulate global or organizational *policy*, while rulesets that pertain to specific classes (such as clients or products) are stored within those objects.

The separated rulesets need to be maintained and kept under version control. This implies that a good BRMS will store rulesets in a repository.

We think that a good BRMS should allow applications to be deployed in a service oriented architecture (SOA). The rule engine should therefore present itself as a service to applications and applications should be deployable themselves as services (e.g. as web services).

Returning to the linguistic gulf that too often separates developers from their customers, we need ways of writing the rules that are understandable to users. Ideally, this would be pure natural language but unfortunately it is impossible (in principle, I believe) for computers to understand unstructured human discourse. Our speech is too larded with cultural referents and ellipsis. There are four possible solutions to this problem:

1. Make business people learn computer-understandable languages like Java or UML. The language can be textual or graphical but it must be computer executable.
2. Invent a computer language that *looks like* natural language.
3. Provide user-friendly interfaces that generate rules in a way that is natural to business people.
4. Restrict usage to the subset of natural language needed to discuss a particular domain.

In our opinion, the first strategy is both arrogant and doomed. But it is currently the norm. The last three strategies all require the construction of a **vocabulary** or **domain ontology**: a model of the things and concepts under discussion and the connexions among them. It turns out that this is much the same idea as that of an object model in UML. However, there are more or less user-friendly flavours of UML, ranging from approaches that use UML like a language describing a Java program to really quite language-independent styles. We will these variation clearly when we look at the products under review. For now, suffice it to say that most people's conceptual model of their subject area does *not* fit comfortably into the object model of any programming language. UML can be used to describe the former but it may also be used to describe more natural conceptual models based on, say, semantic networks (see Section 8.1.1.3 for an explanation).

Thus, modern corporations will need to adopt development styles that fit their development culture. This will substantially affect the type of BRMS product that they choose.

4 FEATURES OF BUSINESS RULES MANAGEMENT SYSTEMS?

We have already quoted Tony Morgan's succinct definition of a business rule. It is worth repeating: 'A compact statement about an aspect of a business [that] can be expressed in terms that can be directly related to the business, using simple, unambiguous language that's accessible to all interested parties: business owner, business analyst, technical architect, and so on'. Morgan, because he has a long track record in artificial intelligence, assumes that these rules are embedded in a régime that can link them together; a key component of any BRMS.

In much other work, the implicit definition is much more fuzzy. Many writers, coming – as they usually do – from a database background, see business rules as little more than database cardinality constraints or even simple formulae.

The formula margin = revenue – direct.costs is *not* a rule. It is a statement of identity. It could even be regarded as a procedure for computing margin, as can many algebraic equations of this sort. Rules in BRMSs are characterized by being non-procedural; they state what is true, not how to compute it.

The mavens of UML mostly see business rules as coextensive with the idea of OCL-type statements and, of course, they are not completely wrong in this. Indeed, I would agree that a business rule is exactly a logically valid statement concerning the objects in the domain that must always be true[1]. However, what this view misses is the idea that rules interact. Even simple facts can interact. If I, for example, tell you that the writer of this report has grey eyes and that Ian Graham is the author of this report, now you know these two facts. But what if I were to ask you 'What colour are Ian Graham's eyes?'

Of course, you know the answer. But I haven't told it to you. You *inferred* it!

Similarly, from two rules that say 'if you overeat then you are likely to become obese,' and 'obese people often die young,' you may infer the obvious, dismal, if reassuringly probabilistic conclusion.

So, a BRMS has to support automatic inferencing, as well as rules and facts. Rules and facts are the easiest to formalize.

A rule is a statement that has, or can be transformed into, the form IF x THEN y, where x and y can be of the form A is P and|or B is Q and|or ... but y can also be an instruction to do something: an action. An example might help.

If an applicant's socioeconomic group is A and the applicant is not married then send the luxury dating brochure

This is written in fairly plain English. In a conventional rule language it is likely to be a little more opaque of expression – something like

If Applicant.SEG is "A" and Applicant.married is FALSE then "Send luxury dating brochure" is indicated.

As we have already seen, business rules do not always follow the if/then form, but may be specified in different formats that are not as closely linked to the underlying rule-engine implementation or syntax. Business rules often tend to use the deontic (must or must not) form to expression constraints or inferences. For example:

An order must not be invoiced before dispatch.

---

[1] The Object Management Group is, at the time of writing, in the process of extending UML to address business rules, with standards such as Business Semantics for Business Rules and the Production Rule Representation (the latter being co- developed by IBM, Fair Isaac and ILOG amongst others).

#	The luxury dating brochure should be sent to an unmarried applicant who's
	socioeconomic group is A
#	An applicant for credit must be at least 18 years of age.

Morgan (2002) gives a great deal of useful guidance on how to phrase rules, preferring the above form to the 'if … then …' form imposed by many products.

In this section, we discuss the features of business rules management systems and the technical terms and issues that will enable our later product analysis. These features include:

#	architecture.
#	integrating with enterprise applications.
#	knowledge representation.
#	inference strategies.
#	forming rules into independent but chainable rulesets.
#	the status of features found in current BRMSs.
#	creating intelligent applications that interact with users through natural, understandable
	and logical dialogues.
#	allowing business analysts and even users to create, understand and maintain the rules
	and policies of the business.
#	rule repositories, versioning, etc.

4.1 The components and technical features of a BRMS

BRMSs are related (both intellectually and commercially) to the expert systems products of the 1980s. To understand them it helps to know a little about their origins, although the products we review have come a long way since then.

Rule-based or 'expert' systems, sometimes called knowledge-based systems, are computer systems that can give advice or make decisions in a narrowly defined area at or near the level of a human expert. There are two kinds of such systems: systems that take decisions, which are chiefly process controllers and applications such as financial program trading systems, and systems that act as decision support systems, giving advice but not making decisions. This definition is couched in terms of what expert systems do. More importantly, rule-based systems are defined by how they do it: by their architecture. The most important architectural feature is that knowledge about a problem is stored separately from the code that applies the knowledge to the problem in hand. This applies equally to BRMSs. Some early expert systems jumbled up facts, data, procedures and rules in the knowledge base whereas modern BRMSs usually maintain a cleaner demarcation between business rules and business data. The rulebase is seen as acting upon the database (including metadata).

The repository of chunks of knowledge in a BRMS is referred to as the **rule base** or **knowledge base** and the mechanisms which apply the knowledge to the data presented to it as the **rule engine** or **inference engine**. This characteristic architecture is illustrated in Figure 4.1.
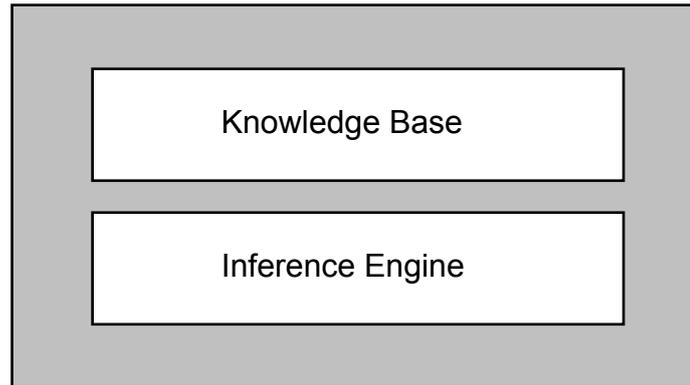
Figure 4.1 The architecture of a business rules management system

It is now widely accepted that there are essentially four components of a business rules management system. Firstly, the underlying environment of symbol and value manipulation which all computer systems share and which can be thought of as the programming languages and support environment; editors, floating point processors, data structures, compilers, etc. The grey area in Figure 4.1 represents this. Secondly, we have the structure of the knowledge base itself including methods of representation and access, and lastly there must be some techniques for applying the knowledge in a rational manner to the problem at hand. This third element is the inference engine, which chains the rules together to reach valid conclusions. Usually this is done non-procedurally but some BRMS also provide other methods whereby, for example, ruleset execution can be handled by a faster approach such as procedural rule firing. The fourth element is the repository, in which the rules are stored and from which they may be manipulated, versioned, shared, managed and so on.

The knowledge base and the inference engine are separated from one another to facilitate maintenance. After all, in most cases rules and policies will change over time and one does not want to rewrite the inference engine (the program code) whenever a new rule is added.

The knowledge base usually contains different kinds of knowledge; typically these include knowledge about objects, procedures and causal relationships. Knowledge about objects is usually stored in the form of an object model, XML schema, data model or semantic network. Procedural knowledge may be represented as Java methods, Excel macros and so on. Some business procedures can also be represented with rules.

4.1.1 Rules

Knowledge about causal relationships is usually stored in the form of **rules** of the form 'IF A THEN X'. Unlike the if/then statements found in conventional languages like Java, COBOL or C++, rule languages in are typically **declarative** or equivalently **non-procedural**; that is, the order in which the rules are written is not important. These rules work on knowledge about entities or objects. As we have said, another important way to represent knowledge is as **procedures**, as found in conventional languages. There are various other ways to represent knowledge, but rules, procedures and objects are the main ones used in business rules management systems at present.

In all BRMS products, rules are represented as sentences, usually containing the words IF and THEN. Morgan (2002) recommends a better style aimed at removing ambiguity, making relationships explicit, avoiding obscure terminology, removing wordiness, and so on. His style is remarkably close to natural language. He ends up preferring forms such as

```
A loan may be approved if
     the status of the customer is high
     and the loan is less than 2000
```

```
        unless the customer has a low rating
```

to

```
if the customer status is high and the loan is less than 2000
      and the customer does not have a low rating
      then approve the loan
if the customer status is high and the loan is less than 2000
      and the customer has a low rating
      then don't approve the loan
```

All three products reviewed support the second style of rule writing; only HaleyAuthority provides natural support for the first. Blaze Advisor, as we shall see, offers a completely different approach in the form of their Rule Maintenance Application. This allows the creation of custom rule maintenance forms that allow users to interact using any format of rule presentation considered appropriate to the business situation.

4.1.2 Rule templates

Rule templates are design patterns for rules. In many circumstances, a rule might be applicable to several data. In such cases, rule templates allow for the creation of rules with empty slots to be filled in later. A business rule template represents a partially defined business rule that contains placeholder slots for missing information. Templates can be used to create multiple rules with a similar structure, where only the value filled in the slots varies.

4.1.3 Rule syntax checking

A good BRMS will offer facilities for checking the rule syntax in real time, as the rules are entered. With structured rule languages, it is useful if the syntax checker highlights keywords, variable and values using different colours. There should be clear links between the object model and the rules.

4.1.4 Procedures and algorithms

Some knowledge is distinctly procedural. For example, we cannot compute our tax liability unless we first know our income and expenditure.

Rule representation can be very cumbersome when the knowledge to be stored is procedural. Examples include mathematical and financial computations. A good BRMS will offer the ability for rulesets to invoke procedures and for procedures to call upon rulesets to execute and return values.

4.1.5 Ruleflows

Ruleflow mechanisms with BRMSs let the designer specify that knowledge modules or tasks be carried out in a particular order. These tasks may be rulesets, functions or entire ruleflow modules. Such a feature is essential for a good BRMS.

4.1.6 Decision tables and decision trees

In some products, there are alternative representations to rules for if/then knowledge. We consider two of these: decision trees and decision tables. Decision trees represent the rules pictorially as a tree structure. This may be a useful aid to debugging or communication between users and developers or analysts but is not usually how business users visualize their knowledge. Decision tables represent the same knowledge and rules as decision trees in a tabular format.

The main problem with decision tables is that they grow unmanageably large when there is a large number of conditions in the rulebase. The approach gives a larger number of rules – one for each row – and the rules will be hard to read and understand. We characterize the approach as **row-oriented decision tables**. Rule subsumption checks may allow the author to tidy up the resultant rulesets but we think a rule induction approach is far sounder. It is better to use a data mining system to extract rules from decision tables and feed them into a BRMS.

The main advantage of decision tables arises when the organization already holds the knowledge in this form: pricing charts, rate tables, etc. However, this advantage largely evaporates when the rules can access the same data in the form of lookup tables. See Appendix 8 for more details.

4.1.7 Inference

An inference engine offers one or more means of applying knowledge to data. The most common strategies are known as **backward chaining** and **forward chaining**. Backward chaining or **goal-directed** reasoning is typical of product selection, diagnostic or advice giving systems. It involves deriving a plausible reason for some given fact. For example, given the fact 'the patient has spots' a medical expert system might reason that the patient could have been among young children recently since young children often have measles and measles causes spots. Forward chaining or **data-directed** inference takes all data present and attempts to discover as much as possible by applying as many rules as possible to them, or filling as many frame slots as possible. This is typical of process control and scheduling applications. It is also typical of many 'form filling' applications, such as tax credit or loan approval. Most rule-based systems involve a mixture of backward and forward chaining and other strategies to reduce blind search.

Implementing forward chaining efficiently is hard since, when a rulebase becomes large, naïve algorithms for forward chaining become very slow because few changes are made to the facts in working memory at each cycle. **Rete** is a very efficient mechanism for solving this problem. Rete is much more efficient at determining the relevance of rules, given particular data, than the equivalent nested if/then/else or select/case constructs. The rete network modifies itself after each rule firing, so that unneeded rules do not fire. The greater the number of rules, the greater rete's advantage over procedural code. This applies to rule execution. Of course, writing the rules is also far more efficient in a BRMS. See Appendix 8 for more details on rete and inference strategies.

All three products studied in this report offer support for backward, forward and mixed chaining using the rete algorithm. Each product has a proprietary improvement on the basic rete algorithm. These improvements are largely responsible for the variation in performance of the three rule engines. The three engines all also modify basic rete to permit backward and mixed chaining.

4.1.8 Uncertainty and explanation

Two other features, which separate rule-based systems from other computer systems, are that they can often:

provide an explanation of their reasoning.
incorporate qualitative or judgemental reasoning and manage uncertainty.

If the last two features are both present, rule-based systems can offer multiple conclusions ranked by a measure of confidence. Both features, if required, tend to increase the cost of system building and may, in some circumstances, imply additional complexity in defining the business rules. Built-in explanation facilities are useful debugging aids but are rarely suitable for user enquiries. Useful facilities for explanation of the system's reasoning to users usually must be hand crafted.

There are several techniques for managing uncertainty, the most common being:

- \# Reasoning explicitly using verbal labels for uncertain terms
- \# What-if facilities
- \# Truth maintenance systems
- \# Certainty factors
- \# Bayesian probability
- \# Fuzzy sets

Reasoning about uncertainty adds to the complexity of a system, and the knowledge acquisition associated with specifying it, but permits it to tackle more complex problems.

None of our shortlisted products offered sophisticated uncertainty management or the last three uncertainty management techniques. Blaze Advisor offers a scoring system based on the idea of scorecards, which can be regarded as a certainty factor variant. {36}

HaleyAuthority relies on uncertain linguistic constructs, such as 'may be' and 'could'. With the other two products, it is a matter of choosing attribute names that imply uncertain value ranges; e.g. terms such as 'risk averse'. What-if is handled in the testing environments of all three products or could be coded into any application that relies on a rule engine.

All three products supported truth maintenance well. A truth maintenance system keeps track of dependencies among sentences and allows the rule engine to retract assertions in a consistent way. This takes account of the kind of uncertainty we face when, over time, things we once believed true become false (*cf* Russell and Norvig, 1995). Truth maintenance can help improve the explanation facilities offered by a BRMS.

Appendix 8 discusses knowledge representation, inference, and uncertainty management in much more detail. It also includes more detailed material on such topics as decision tables, rule induction and data mining, explanation facilities and semantic networks. We finish this section with a summary of some other features that we have used in the evaluation.

4.1.8.1 Explanation and help facilities

Imagine a conversation between a life assurance salesman and a potential client. The rep takes the customer's personal and financial details and enters them in to the BRMS application on her laptop. She then asks a few well-chosen questions. At the end of this, she announces 'Thank you for your frankness, Mr Suzuki. I think the best product that we can offer you is a life policy linked to an investment in gilt-edged government bonds. That will provide you with adequate death benefit to cover your wife's needs and provide for your son's education and marriage costs.'

'Thank you, Diana, but I don't understand why.'

'Well, you told me that you are only 27.'

'That's right.'

'That is quite young in this context and we have a rule that says— No, look, I can show you.'

She swivels her knees so that he can see the display on the laptop. 'See? This rule here.'

Mr Suzuki slides his spectacles up his nose and reads.

> A bond linked policy is recommended for a client
> if the client is averse to risk and young

'OK; but what made you think I'm averse to risk? I never said anything about that.'

'We think that people with young children are usually averse to risk; because they want to protect their interest in as safe a way as possible. Look, here's the rule.' Diana presses a function key.

> A client is averse to risk if the client has children

'Ah! So, I understand now. What will the growth projection look like?'

The sale is nearly closed and both Diana and Mr Suzuki are pretty sure that the recommendation is a good one.

On a technical level the BRMS has fired the 'best product' ruleset, given the recommendation (bond) and printed an elaboration of the benefits (stored as explanation text perhaps). When asked, it unwinds the rule stack and shows which rules have fired.

All the products we looked at in this study offer this kind of rule trace in test mode. The same information is available to the applications using the rules but, in each case, it requires some programming to create an interface such as the one used by Diana's company. This is a shame but it is really quite hard to design completely general-purpose user-friendly explanation and help facilities.

4.2 The rôle of object modelling and natural language processing

Business rules management systems cannot be built without paying attention to rules and the objects that these rules refer to (the object model or domain ontology). The object model provides the business **vocabulary** that the rules can use to talk about a problem.

In particular, any attempt to represent rules in natural language will fail unless there is a well-constructed model of the objects and concept in the domain, their attributes and relationships. Furthermore, there must be a link between the object model and the rule language.

All the products considered in the sequel require that you build or reuse such an object model. The only questions are the order in which you create the models (rules first or objects first) and the degree of integration of the models.

One approach, used in early expert systems shells, is to create the objects automatically by parsing or compiling the rules. This leads to an impoverished, flat object model that often can't distinguish object from their attributes, makes it difficult to attach methods or rulesets to objects, and is complex to interface to existing business data. A better approach is to create a good packaged and layered component model separately from the rules. This is usually not a trivial task and most organizations will benefit from help in enhancing their component modelling skills. Indeed, one of TriReme's specialities is mentoring in this area and we have developed advanced methods for component based design and SOA, including Catalysis™.

If natural language processing is to be attempted we need to do two additional things. The model needs to reflect the way people think about objects and the way they construct sentences to talk about them.

The way normal people (as opposed to the Javarattzi) think about objects is not quite the same as the semantics of a C++ or Java object model. For example, in rule-based applications, rôles are important concepts and, in real life, instances often change rôles or adopt multiple rôles; I can be a student *and* an employee. For this reason a modelling approach based on semantic networks is more appropriate than on based on the semantics of programming languages. Such a model, however, must be translatable into code.

To capture the way people construct sentences to talk about objects is rich and varied. We have a choice: either restrict the syntax or teach the machine how to understand a wide range of phrasing. Both approaches have advantages. A well-design formal syntax can look like English and is quick and easy to type once you know it. A parser can warn you if you have violated the syntax or referred to an object that doesn't exist. However, the rules may look strange to the untutored eyes and it is impossible to just pick up rules written by business experts and just drop then into the application. On the other hand, natural language phrasings need to be made explicit by the knowledge base creator and this takes time and effort. The reward for the extra effort is that practically anyone can now add or change rules with the domain.

That last caveat is important; the system has to *know* about a particular domain. The object model and the phrasings constitute the limits of the system's knowledge.

5 PRODUCT PROFILES

There are many products that allow users to develop rule-based systems. Not all of these may be classified as BRMS because some, such as the expert system 'shells' of the 1980s and their descendents, do not usually offer repository-based rule management. I also excluded a number of popular open source solutions because they are not rule *management* systems. Examples include Jess and Drools. I have not considered such products in this comparison. But they are discussed briefly in Section 5.4.

From the long list of candidate systems we had to make a shortlist. The criteria we applied were that the shortlisted products should

allow business analysts to create and modify the rules;
use a fully-featured repository;
be rete-based;
support backward chaining;
allow the rule engine to be a component or service within larger applications;
allow applications to be deployed in a service oriented architecture;
focus on business rules management (as opposed to just workflow) problems;
provide evidence of successful commercial applications;
offer commercial-standard professional support (thereby we eliminate the open source products).

These considerations immediately eliminate many products, including the remainder of the ones listed in Section 5.4; because they are not rete-based, lack a full repository, were not compatible with a component-based or service-oriented architecture, lacked commercial support or perhaps more than one of these. Based on these considerations, we concentrate here, then, on only three rete-based business rule management component systems products: JRules from ILOG, Blaze Advisor from Fair Isaac and HaleyAuthority from Haley Systems. We will consider them in alphabetical order of their abbreviated names in the comparative evaluations of Section 6.2.

HaleyAuthority offers a natural language approach to rule authoring and sports a very fast and fully-featured rule engine. [www.haley.com]

Fair Isaac's **Blaze Advisor** (devised from experiences with the Nexpert Object expert system shell, which is still available as Blaze Expert) helps define, execute, and manage rules-based processes within business applications. It automates deployment and data integration for use in any Java, .NET or COBOL environment. [www.fairisaac.com/rules]

ILOG's **JRules** is a complete environment for creating and deploying business rules management systems in a Java environment. [www.ilog.com/products/rules/]

In what follows, numbers in curly brackets at the end of a paragraph are cross references to the evaluation criteria in the decision table in Section 6.2, when these are referred to in the text.

5.1 HaleyAuthority

HaleyAuthority from Haley Systems was originally called Authorete (but always pronounced Authority) to emphasize its support for rete. It is a rule authoring and knowledge management system that generates code for its sister rule engine HaleyRules. HaleyRules uses a rule language called Eclipse, a sophisticated descendant of the CLIPS language, which is in turn a descendant of OPS-5. There is no connexion whatsoever with IBM's Eclipse IDE.

Installation was trouble free and no JVM is required for the version we used (Version 5.0). The documentation was among the best and clearest IT product documentation that I have ever had to read. Creating my test application was a dream compared to the other products. {3,17,18}

HaleyAuthority is a Windows application, but it does have a Web Services component as well. HaleyRules is available both as a pure Java implementation and a C-based implementation

with out-of-the-box interfaces for integration with Java, .NET, C++, and Visual Basic applications. The availability of native and Java implementations enables Haley to support a very wide range of platforms and application scenarios. {3,4,46,60}

The files that are deployed for testing are the same files that would be used within an application. Typically, HaleyRules is embedded in an application or server side process and API calls are used to initialize the engine and give it the HaleyAuthority generated files to load. HaleyAuthority does not produce applications; it generates the logic (rules) that can be accessed by applications through the use of the HaleyRules engine. HaleyRules has a clever knowledge base loader that only loads what has changed thus reducing the impact of changes on users. {16,64}
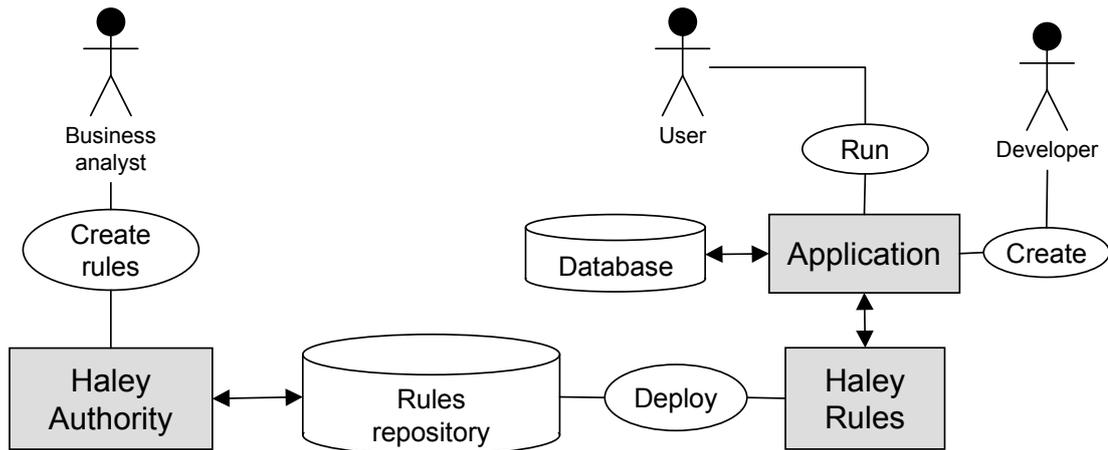

Figure 5.1 Haley architecture.

Another powerful feature of HaleyAuthority is its automatic generation of integration or glue code. It also has code import functionality that allows you to import an XSD or, say, a Java object model and map the elements of the XSD or the object model to its concept model which is defined in business terms. HaleyAuthority then generates the glue code that enables the rule engine to invoke the object model at runtime. In the case of the XSD, XML requests conforming to the XSD can then be directly asserted into HaleyRules, which does the processing. This means that there is no need to write code to parse the XML – the engine does it. This facilitates the integration of existing applications with little programming. Another use of this capability is that you can write rules that orchestrate interactions among existing implementation objects.

Unlike products that use multiple structured syntaxes, HaleyAuthority enables you to enter your rules in plain English. HaleyAuthority is the only candidate tool that generates executable logic from business requirements and policies expressed in natural language (currently only English). This is very helpful in helping you avoid errors. I could not get away with entering ambiguous rules: rules that the other products had tolerated at the syntactic level. {25} What you do is type a rule such as

> An annuity is recommended for a client if the client is averse to risk and the client is retired

or
> An application should be referred if the applicant is a smoker

Then you need to teach HaleyAuthority what the statement means by defining the concepts and relationships (vocabulary) used in it. This is done in a quite non-technical way compared to the other products considered; it does not require programming knowledge or a prior

implementation of objects in Java or .NET as is the case with JRules and, although to a lesser extent, Blaze Advisor. The disadvantage of this approach is that it may make linking to an existing database more complex. Of course, if there are multiple sources of business data, this complexity is ineluctable, as any data warehousing project will testify. When HaleyAuthority 'understands' a term, that part of the statement appears in bold. Starting with the concept of an application, I create an entity with that name and another called **applicant** that is a specialization of a **person**. I also create **smoker** as a kind of **person**. The feel is much more like a semantic net than a Java object model, although a Java object model is generated automatically behind the scenes.

Now I have to define relationships for being referred and smoking. Simple pop-up dialogues guide me to create a **referred** relation with application as a 'rôle'. I notice that I do have to understand basic entity modelling concepts, like the difference between many-to-one and one-to-one relationships, but this isn't too much to expect of a business analyst I think. Next, I define any phrasings for the relation that I want to be able to use, in this case: **an application should be referred**.

To do this I specify that the phrasing uses the modal verb SHOULD and the past participle of TO REFER with the auxiliary verb BE. Not too hard, providing you have some inkling about English grammar. Now to populate my world with smokers I create a relation using IS as the verb and giving smoker the 'syntactic grammatical rôle' of direct object and applicant that of subject. And we're done. HaleyAuthority understands my rule.

The more rules I create, the quicker this process gets as my semantic model becomes more complete. Furthermore, it's much more natural than having to start with an object model, say in Java, and any attempt at sloppy thinking or ambiguity is soon detected by this tool. HaleyAuthority uses an internal expert system for parsing and semantic processing of statements in real time on a word-by-word basis. HaleyAuthority also suggests valid choices as a sentence is being defined. It feeds back the words it understands in a sentence by showing these in bold. {32}

As an example of the clarity of thought that HaleyAuthority insists on from its user, when I wanted to say 'if a client has any children' I had to decide if child was a concept related to client (a client has a child) or having children was a property of a person. This indicates that some training and practice would have benefited me somewhat.

A downside for me was the terminology that HaleyAuthority uses: 'grammatical rôle' seems to stand for case (in the grammatical sense) and (in other places) HaleyAuthority talks about the '–ing verb form' rather than what I have always called the present participle. On the other hand, you soon get used to it and the user interface is superb; I loved being able to drag one concept over another to create a relation between them. Like most native speakers of English, my only non-superficial knowledge of grammar was acquired by trying to learn *foreign* languages. HaleyAuthority assumes at least a basic skill level in this respect. On the other hand, it assumes no technical skills beyond elementary entity modelling. {4}

The product supports different policy and rule expression formats: if/then, constraints, declarative definitions. In addition, HaleyAuthority lets you write general statements directly in natural language. HaleyAuthority supports the expression of archetypes, templates, overrides and specializations and exclusions, including the ability to specify the conditions under which an override or exclusion should apply using applicability conditions, rule templates, and designation of statements as overrides for other statements {25,27}.

Although HaleyAuthority allows you to enter business rules in natural language, it does not allow the use of any kind of punctuation, like commas and even humble concluding full stops. If you add a stop, HaleyAuthority can't compile the rule. It does, however, understand the Saxon genitive apostrophe as in: '**A person's mother's niece is the person's second cousin**'. The fact that I really could (and did) type in this last sentence and get it understood impressed me in some ways far more than the more focused tests I did with this product. This definition of second cousin is hard to cope with for a human, never mind a machine. And of course the question of punctuation didn't even *arise* with the other two products, neither of which go anywhere near this level of natural language processing.

Test cases can be defined using XML-based test data that represent incoming transactions. As HaleyAuthority contains embedded within it a copy of HaleyRules and can thus simulate the execution of the knowledgebase – especially as it allows the use of XML patterns representing external transactions for these simulations. It also allows non-technical rule authors to compare test results so that they can carry out regression tests themselves. HaleyAuthority allows these test cases to be grouped so that automatic regression tests can be run as the rules evolve. To test my rules, I created a set of instances of clients with different attributes. {40,41,44,50}

Debugging is supported both within HaleyAuthority at the rule level, and within the development environment for HaleyRules, where detailed racing of rule execution and working memory changes can be traced. HaleyRules also provides an API for passing knowledgebase execution information to external applications at runtime. {50}

HaleyAuthority has the best effective date mechanisms we have seen, with the ability to create deployment polices with future effective and expiration dates. We can reason with concepts like TOMORROW and YESTERDAY. The product has a repository which enables changes to be managed for multiple concurrent users. This supports workflow in development with rôle-based permissions for change management. The repository supports workflow features that can be used to implement an approval process for rule maintenance, tagging rules as 'proposed', 'reviewed' or 'approved'. Ruleflows are implemented by setting module (i.e. ruleset) priorities and writing rules to control branching among the modules. {5,39}

With regard to the tabular representation of rules, the current version of HaleyAuthority supports lookup tables rather than decision tables, Look-up tables allow information to be presented to rulesets in tabular form and reasoned with. A look-up table relates up to two sets of variable ranges (such age ranges) to a set of actions (such as the medical tests required of an applicant for life assurance). This has the same effect as support for decision tables in that rule data can be captured in tabular form. It suffers from the same disadvantage of decision tables; i.e. a large table may be equivalent to only a few rules. However, it is a useful tool to have, and is easy to use. {29}

HaleyAuthority includes a useful library of dates, units and quantities and phrasings for reasoning about them. These offer improved productivity and shorten the rule development cycle. For example, you can write rules that mention dates and temporal concepts such as before and after, without having to write any code to define such concepts. You can create polices with future effective and expiration dates – deployment parameters can be directly associated with modules. Many basic concepts are predefined and the concept sets are extensible. For example, DOLLAR and TON are predefined. Adding YEN and TONNE is easy because they still behave as money and weight. So we can write 'If the unladen weight of a vehicle is more than two tonnes then the vehicle is a heavy goods vehicle'. {34,65}

There is automatic, real-time, multiple cross-referencing of rules and concepts (ontology). {31}

Incremental development and selective deployment is helped by the ability to make incremental additions to modules and sentences. {33}

HaleyAuthority supports multiple evaluation and ruleflow control strategies. The product lets you define, maintain and organize rules into rulesets or modules. Applicability conditions can be shared across rulesets easily by defining these at the module level. Priorities and applicability conditions may be applied at both the module and statement levels. It also supports forward, backward, and mixed chaining with full support for truth maintenance. {28,35,36,43,49}

HaleyAuthority provides a simple deployment mechanism whereby an authorized person can deploy the rules directly into test or production environments, without any downtime. As HaleyRules can be configured or instructed by an application to check for any updates to the rules, it can then examine the deployed files and only load the changes like data; that is, without needing the current process to be restarted. Code generation capabilities include integration code for invoking external object models and direct processing of XML requests. {42,51}

There is built-in support for importing data and mapping it to the business vocabulary, usage, orchestration, and integration with external data representations and methods (XML, .NET, Java).

HaleyAuthority has import wizards to support this process. Support for XML input is direct with no need for additional programming. It generates the integration code automatically. {12,38,51,52}

There is support for dynamic or 'hot' deployment. The HaleyAuthority deployment button can invoke an external script. Support for multiple concurrent knowledge bases is good too. A single instance of HaleyRules can load multiple knowledgebases, and for each knowledgebase it can maintain an unlimited number of working memories subject to memory and CPU constraints of the host platform. {61,63}

There is not currently an operation-level 'undo' (e.g. undoing the addition of a concept). However, rollback is supported on the knowledgebase, which enables 'undo' of all of the operations performed during a session.

We consider the report generation capabilities of all three products as adequate. Haley has multiple levels for the knowledge base, as well as change management and test results. {6,45}

HaleyAuthority's nested logic syntax conforms to Morgan's guidelines (see above) and can dramatically reduce the size of rulesets. In a recent IDC report, the following single HaleyAuthority nested statement replaces a conventional if/then equivalent with 12 complex rules.

```
A child meets the relationship test
    if the child is the taxpayer's child
    if the child is a descendant of a child of the taxpayer
    if the child is a relative of the taxpayer who cared for the child
    if the child is an eligible foster child
        unless the child is married
                unless the child is a dependent of the taxpayer
                        if the taxpayer can claim the child as an exemption
                        if the taxpayer gave away the right to claim the
                        child as an exemption
```

Here is an example of just *one* of the twelve equivalent rules (also in Haley syntax).

```
If the child is a relative and the taxpayer cared for the child and the
child is married and the child is a dependent of the taxpayer and the
taxpayer can claim the child as an exemption
then the child meets the relationship test.
```

There is a point of controversy on this style of rule reduction. If you combine multiple rules into a single rule you save space but are you effectively 'rule programming'? What happens if you need to add a side effect: another action for one of the cases? You may need to rewrite your whole rule instead of just modifying a single rule. Of course HaleyAuthority does not make the choice for you. You need to choose. In my opinion, the solution to this conundrum is to develop domain-specific analysis patterns to assist in matching the rule style to problem type, performance requirements, etc.

IDC also identifies impressive reductions in the number of conditions and overall words. This 'applicability condition' style of rule writing is a powerful alternative to the more usual if/then style. Applicability conditions can be dragged to other modules to save retyping them.

5.1.1 Related Haley Systems products

Haley Systems Inc. (formerly The Haley Enterprise) specializes in rule-based and case-based reasoning technology and in automating managed knowledge within business solutions. Software from Haley is embedded in a variety of commercial software packages and applications by many companies internationally.

The company was established in 1989. Its principals have led the commercialization of BRMS since that time. It core products are HaleyRules, which is – as far as we can tell from published and anecdotal evidence – the fastest rete-based rule engine available, and HaleyAuthority, the only system to support rule authoring in natural language.

HaleyRules is embedded in every copy of HaleyAuthority but may be purchased separately. Haley's rule engine has a very small footprint when implemented (1,000 rules use less than

1Mbyte). The engine can run in less than 4 MB of RAM. It has even been ported to a Palm Pilot and other PDAs and will shortly be available on Z/OS. {58}

HaleyRules builds on and integrates Haley's other products: Eclipse, Rete++ and CaféRete. Eclipse is a high-performance production system that provides an extended version of the CLIPS rule language syntax using the rete algorithm to support both forward and backward chaining. It is implemented in a layered architecture for maximum development and integration flexibility or embedding in the smallest footprint. It includes an extensive programming interface for ANSI C and Visual Basic to libraries at each level of its layered architecture and is encapsulated in and integrated within C++ as Rete++. It is also available as Agent OCX for COM integration with OLE automation under Windows and as an ActiveX control that supports Microsoft COM, Visual J++, and Internet Explorer.

Custom vocabularies for HaleyAuthority are available or being developed to support some vertical market sectors.

5.1.2 Conclusions

HaleyAuthority is significantly different from all its competitors. The system can really understand natural language expressions, as opposed to systems that substitute pseudo-English in place of pseudo-code. It is especially easy to use when making changes to the rules once the knowledge base has been built. Unique features such as nested conditions and 'unless' clauses serve to both make applications more efficient and reduce the overall time it takes to develop and edit rules.

The HaleyRules rule engine offers good support for rete, forward, backward, and mixed chaining, automatic truth maintenance and conflict resolution. It has the ability to handle a large number of rules and large numbers of concurrent requests/transactions, users, and rule executions. HaleyAuthority also supports multiple evaluation and ruleflow control strategies. Priorities and applicability conditions may be applied at both the module and statement levels. {35,36,48,49,67}

Haley's footprints for the rules repository and rule engine are small and enable the system to be deployed on a wide range of platforms including, for example, Palm Pilots and Windows CE devices. Haley's rules engine performance specifications are impressive and it is clear that the tweaks and advances that Haley has made over the last decade have resulted in a fast, efficient rules processing engine. {54,58}

We concluded that HaleyAuthority will fit well into development environments where business experts or non-technical business analysts need to create, maintain or test the business rules that reflect evolving business policy. This is absolutely necessary when policies change rapidly and time to market may not be sacrificed to the application backlog. It is also very suitable in situations where applications need to be deployed on mobile devices in the field.

5.2 Blaze Advisor

Fair Isaac's Blaze Advisor Version 6.0 provides the same BRMS on 2 main platforms, Java and .NET, with an option on the Java version allowing the generation of COBOL code. The former requires a (freely available) JVM to be installed, and the latter requires Microsoft's .NET Framework (and recommends Visual Studio) to be installed. After reading the instructions, installation of the Java product tested was trouble free. No changes to environment variables were needed and the product ran as described first time. {3}

The tutorial is thorough and reveals Blaze Advisor as a mature product with complex features; although beginning users may well not understand the need for all of them. {4,17,18}

Application development (viewed as 'rule service development') proceeds by importing or creating objects, rulesets, functions, event rules, questions sets, enumerations and ruleflows within a project repository. This can be done in any order but it is natural to start with the basic object model: classes and enumerations. These can be imported via wizards from Java, COM/.NET, XML, or a database – plus there is a mechanism for defining your own Business Object Model

Adapter. You can create your own classes and instances too, although Fair Isaac stresses that this is usually done for prototyping and testing only. Once you have created the classes, you can type rules into a ruleset window. Backward chaining requires the creation of event rules or questions sets, which can be used to generate prompts for missing values, so that interactive testing is possible. {5,35,43,46,48}

An application calls upon the rule server and engine to provide a service, such as classifying a situation or diagnosing some problem. In turn, the rule server accesses any needed data. Rules are maintained in a repository with features comparable with HaleyAuthority or JRules. This enables changes to be managed for multiple concurrent users and supports workflow in development with permissions for change management. This architecture is summarized in Figure 5.2. {5,39,46}

The rule syntax has moved on a lot since the days of Nexpert Object, and we find a good compromise between natural language syntax and formal syntax, similar to that of JRules. Business users though are protected from the actual rule syntax as they use rule maintenance applications that access the repository directly in maintenance mode. {5,39}

Blaze Advisor offers three different ways to address the rule authoring problem.

- \# An English-like Structured Rule Language (SRL) for expressing the rules, as well as any data patterns or local classes and instances.
- \# Decision trees, decision tables and scorecard models, which are graphical or tabular ways creating rules.
- \# Rule Maintenance Applications (RMA), which are customizable web-based rule authoring interfaces that can be generated directly from Blaze Advisor. {4,25}
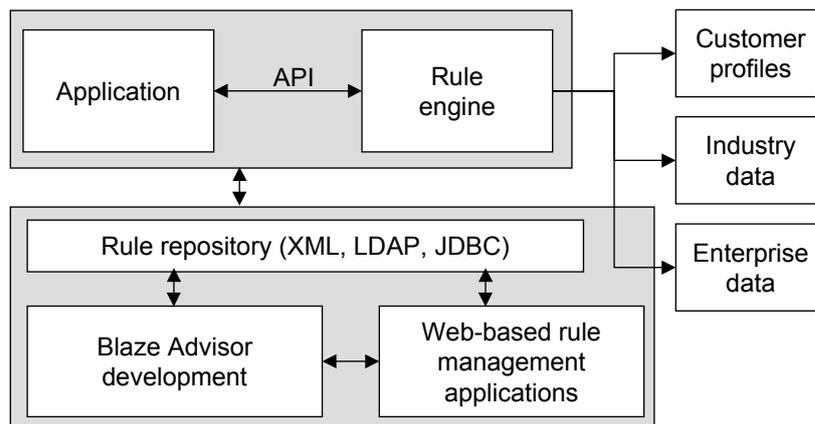

Figure 5.2 Blaze Advisor architecture.

The SRL is an object-oriented programming language designed to make writing and reading business rules seem English-like. It has the features of a programming language, and is intended for use by programmers (as well as "technical business analysts" who are not averse to tasks like programming Excel macros) to create the entities, control the execution flow, and perform the operations required by the rule service. However, it also provides a syntax for authoring rules that is understandable (i.e. readable) by people with little or no programming background. The IDE includes a set of editors that simplify the creation of SRL entities, and generate much of the syntax. Normally, programmers must use the IDE to develop the data model and execution flow, and then provide the business user with access to edit specific rulesets. This access can either be through direct editing of the SRL, the use of ruleset metaphors, or through an RMA. Regardless of which is used, the business user's edits are compiled along with any other SRL to produce the executable rule service, which is either a project that is loaded by the runtime Rule Server, or in

the case of COBOL output, compilable COBOL code. The SRL syntax falls a long way short of natural language but for me, as a "techie", it was fairly easy to read. Writing rules in it requires knowledge of the object model's structure and variable names. Developers can also choose to write templates for rules, which are used by the RMA Generator wizard to provide an interface for business users. This interface follows the philosophy that typing any formal language is too much to expect a busy business person to achieve, and instead displays the parameters that make up the rule within any text – or indeed, HTML construct – that the user needs. This can obviously be much closer to natural language syntax. The assumption is that developers are available to learn how to create rule templates and RMA forms. {43,74,75}

Blaze Advisor provides good facilities for entering rulesets as row-oriented decision tables. It also has a useful graphical decision tree representation. The latter does not support probabilistic trees. Uniquely, Blaze Advisor offers 'scorecards': a special form of table that lets the system reason using additive scores. For example, in a credit scoring application, we may score professional and skilled occupations more highly than unskilled ones. These scores can be combined with other factors such as outstanding mortgage liabilities to arrive at a final score for credit worthiness. This is the mechanism used by Fair Isaac for its credit scoring applications, which the company is well known for, especially in the USA. Other applications include fault diagnosis and sales promotion targeting. Entering textual 'reason codes' for each score makes the method auditable. The score model metaphor provides a limited way of handling probabilistic rules. {36,43}

There is a built-in library of mathematical and financial functions and there is a complete procedural language supporting most familiar programming constructs. This language is at a higher level than Java, giving Blaze Advisor (in our opinion) a slight edge on products like JRules that use unadorned Java for this purpose. Business rules are written in a structured if/then syntax. Rulesets execute under a rete-based inference engine, or may be selected to run in a procedural fashion ("sequentially") if the rules do not need to be declarative or to chain with one another. There is also a license option for a "compiled sequential" mode, in which the rules are compiled to Java or .NET bytecode as appropriate, although this is transparent to the user and still allows for rule changes to be made to a running rule server as necessary. Another license option is the aforementioned COBOL output. The rulesets, together with procedures and functions, can be chained procedurally or 'orchestrated' using ruleflows. Blaze Advisor lets you group your rules into functional rulesets, and then lets you control the sequence in which rulesets are called by using a ruleflow. Ruleflows are displayed using a proprietary, but clear, graphical notation. {43,74}

The product supports different policy and rule expression formats: if/then, constraints, declarative definitions. Rules can use Java-style dot notation or more English-like, business-friendly constructs (aliases for attributes), or both interchangeably. Rule templates are supported well. {25}

Effective dates and times can be stated for each rule.

Rule inheritance is Blaze Advisor's approach to rule subsumption. Any rule can refer to any other rule, in which case it inherits the referred rule's conditions. This is used, for example, in the decision tree metaphor. Rule inheritance saves development time and leads to cleaner rulebases, but it can be dangerous when used improperly but Blaze Advisor is a tool that, when used properly, can help companies focus on the important things in business life.

A good analysis tool supports conflict resolution: potential rule conflict. Rules whose actions do not change values in the same ruleset are identified as candidates for, more efficient, procedural execution. Infinite loops with the rules are detected automatically. An HTML conflict report can be produced and printed. The AnalyzeRuleService test picks up rulesets that do not cover the state space (as in our example in Section 6). {67}

Blaze Advisor's Business Object Model Adapters (BOMA) provide a common business object representation for writing rules against. In this way, a rule that is written against a Java entity can also be used against a .NET entity. Then, at runtime, you pass corresponding Java

objects, COM objects, COBOL copybook entries, database records, or XML documents to the rule engine, and the BOMA automatically maps them to the correct types of business objects.

Rule developers using SRL can use a full range of debugging facilities including stepping through rules, setting breakpoints on any internal or external data item referenced in the rules, viewing cross references and execution traces, and monitoring performance. The development environment can also be used to debug a rule service transaction taking place on a remote server.

We consider the report generation capabilities of all three products as adequate. {45}

The company claims good compliance with standards, including JavaBeans, EJB, COM+/DCOM, W3C XML and CORBA (the OMG's Common Object Request Broker Architecture). We did not test this claim. {12}

It is relatively easy to deliver an application into most architectures, including thin clients. There are built-in wizards, called Quick Deployers, that generate the necessary client code to invoke the Advisor Rule Server. Options include vanilla J2EE as well as specializations for IBM Websphere, BEA Weblogic, Oracle and Sun application servers, as well as a variety of different deployment types such as EJB, Message-driven Beans, Web Services, as well as plain old embedded code. Naturally, the .NET version only deploys to .NET environments, but that includes C# and Visual Basic interface generation. All versions, including generated COBOL, share the same repository; for greater rule consistency. The Quick Deployer also configures the rule server's configuration file, as this component is not so much programmed via API but configured via XML. Rule servers can manage multiple rule services, and in turn multiple rule engines (called "agents") can be configured for rule services to provide multi-user scalability. A separate deployment component called a Deployment Manager can coordinate rule updates to a live rule server. {46}

Blaze Advisor provides an internal versioning and access control mechanism. In addition to providing a rule check-in/check-out repository, Blaze Advisor allows you to have several versions of the rules for different applications and permits control over who has access to which rule or rulesets. New in Version 6.0 is improved ownership control, ruleset segmentation and release management. Customizable management properties and queries for both technical and business users further enhance rules maintenance applications. New "best practice" documentation helps guide repository design for best performance and maintainability. IBM has certified that Blaze Advisor is optimized across all of its major platforms, including IBM iSeries, pSeries, xSeries, and zSeries.

Another key feature of Version 6.0 is the integration of the Innovator Workbench with the rule builder product. This gives better integration of features such as decision tables and scorecards and makes the interface to RMAs more transparent.

The biggest downsides, in general terms, were performance and price. Specifically, comparing Blaze Advisor with HaleyAuthority, rule authoring wasn't nearly so easy for the non-technical business analyst or user; nor was ruleset consistency ensured so well at rule authoring time (see Section 6) although *post facto* rule analysis detects gaps and conflicts well.

The things we liked most about this product was its interactive testing facilities, making backward chaining far easier to realize, good rule analysis features and its general level of tool integration and ease of use. Also, the RMA is a very sensible alternative to natural language rule authoring, in the right circumstances. {40,41,44,68}

5.2.1 Related Fair Isaac products

Founded in 1956, Fair Isaac is a company that focuses on the provision of decision support and analytics software. Blaze Advisor integrates with Fair Isaac's other tools. For example, Model Builder for Decision Trees helps define strategies such as for marketing campaigns, and once defined, the resulting decision tree can be imported into Blaze Advisor. Model Builder for Predictive Analytics is a data mining and analysis tool whose analytic models can be executed in a Blaze Advisor ruleflow. Decision Optimizer is aimed at resource allocation problems in the style of linear programming. Blaze Advisor can invoke these models when complex calculations are

required. Blaze Advisor also provides rule-based services for Fair Isaac's vertical-market solutions for loan originations (TRIAD), fraud detection (FALCON), debt recovery (PLACEMENTS PLUS), and others. A custom Blaze Advisor solution for the ACORD insurance standard, already deployed in some insurance carriers in the US, and SmartForms – which adds a fourth deployment platform for business rules developed with Blaze Advisor: XML-based web forms (XForms) – are both announced. SmartForms automates data validation by caching rules on the client using XForms technology and is an important addition to Blaze Advisor's capabilities.

5.2.2 Conclusions

As someone with considerable experience of building systems using older expert systems 'shells', I felt immediately more at home with Blaze Advisor than with JRules. I was able to use the product to get my sample applications up and running within a relative short time, given my background in this technology. A non-technically inclined business analyst would have struggled more with the design interface: the rule language syntax falling far short of the natural language approach of Haley. Therefore, we conclude that Blaze Advisor is not really suitable for environments where business users need to get closely involved in the creation and maintenance of business logic unless there are development resources available to create a custom rule interface. The rules can be presented to users in a non-technical way via generated browser pages, but this approach is, we feel, limiting compared to HaleyAuthority's natural language approach. However, if one takes the view that natural language is not always the best way for busy users to enter rules then Blaze Advisor's web-based rule interface is a very attractive alternative. Such a view is reasonable in situations where the rules have complex interrelationships or where the users have no desire to enter rules in their raw form. {5,6,39,50,54}

Blaze Advisor is a mature product with good integration features and is capable of addressing many BRMS situations. We felt that rule authoring was far less easy than in HaleyAuthority although easier than in JRules. Blaze Advisor recovers from this criticism by offering the user the possibility of interacting via RMAs. But Blaze Advisor's light dims in terms of rule engine performance compared to both Haley and ILOG. Although some tuning can be applied to improve the situation (for example, procedural rules execution is possible), this may still be a problem for large rulesets or high-speed transaction-oriented applications. Blaze Advisor shines in the number of tools it offers to programmers. Versions after 5.1 added to these new (but perhaps dangerous) features such as rule inheritance, in addition to improvements in existing features such as decision tables and versioning. {2,54}

Compared to JRules, Blaze Advisor offers far more tools and views for design, analysis, and debugging, and makes it easier to consolidate rules into a central, easily maintained repository. Although Blaze Advisor doesn't match HaleyRules or even JRules in performance, it will nonetheless attract some companies seeking to reduce development time across large numbers of developers.

When we carried out a feature-by-feature comparison (see Section 6.2), Blaze Advisor scored slightly higher than JRules in Scenario 1. But, taking our subjective impression into account as well, we feel that Blaze Advisor is most suitable for organizations that need to use technically aware business analysts – as opposed to actual users or Java developers – for rule creation and maintenance. This positions it between HaleyAuthority and JRules on the scale of the technical prowess of rule authors. In Scenario 2, where the feature weightings are more appropriate to the concerns of a conventional IT organization, Blaze Advisor beats both HaleyAuthority and JRules. In this scenario, developers create RMAs which allow users to interact with the rulebase in the way most natural to them.

5.3 JRules

ILOG's JRules is a BRMS for the Java environment that includes a set of tools for modeling, writing, testing, deploying, and maintaining business rules. It provides a repository in which to

organize and store the rules and a rule engine to execute them. Developers can combine rule-based and object-oriented programming to add business rule processing capabilities to new and existing applications. {5,39}

As with all genuine BRMSs, the rule engine separates business rules from the rest of the source code, executes them (after transformation into executable form) and manages them within the BRMS.

JRules is built on a set of Java foundation classes that provide application programming interfaces (APIs), allowing customization of every aspect of a business rule application.

JRules is the Java version of an older (and almost certainly slightly faster) library of inferencing and rules management components written in C++. It has a long and respectable history and is by now a robust and reliable offering. The version we used for evaluation was 4.6, which runs under most Windows and Unix environments, basically on any platform with an appropriate Java virtual machine. JRules requires the Java SDK Version 1.3 or higher.

The installation required that the JVM was installed first. Installation was then trouble free under XP but almost a nightmare on Win98. I was required to alter the AUTOEXEC and CONFIG files by hand and, even then, it could only be run under ANT at the command line prompt. Navigation through the Windows path structures was a little confusing at first on both machines. {3}

The developer approaches the product using an authoring and testing environment called Rule Builder. Rule Builder provides a graphical user interface (GUI) and several editors to write business rules and create a business object model (BOM). The BOM classes map the natural language syntax of the business rules to the underlying application objects, which can be in Java, XML, or exposed as Web services. A repository (to store, organize, and manage the rules) is created first, followed by an object model. Unlike HaleyAuthority and Blaze Advisor, defining the implementation model as Java objects is a necessary first step to defining rules. {4,5,39,46,55}

A JRules application consists of objects (classes and their instances) and rules. Objects have attributes and methods. Only instances are stored in working memory (because, in Java and C++, classes are not objects). Rules refer to these. Related rules are grouped into rulesets. Rulesets are related to working memory by a JRules 'context', which also holds the current state of the inference process (the 'agenda'). Inference is standard rete forward chaining: execute all applicable rules, remove fired rules from the agenda and then loop until no new applicable rules exist. {43}

Rules are then created, numbered and named. A standard, if rather wordy, if/then syntax is simplified by making relational operators, such as 'less than or equal to', available from pop-up menus. {25}
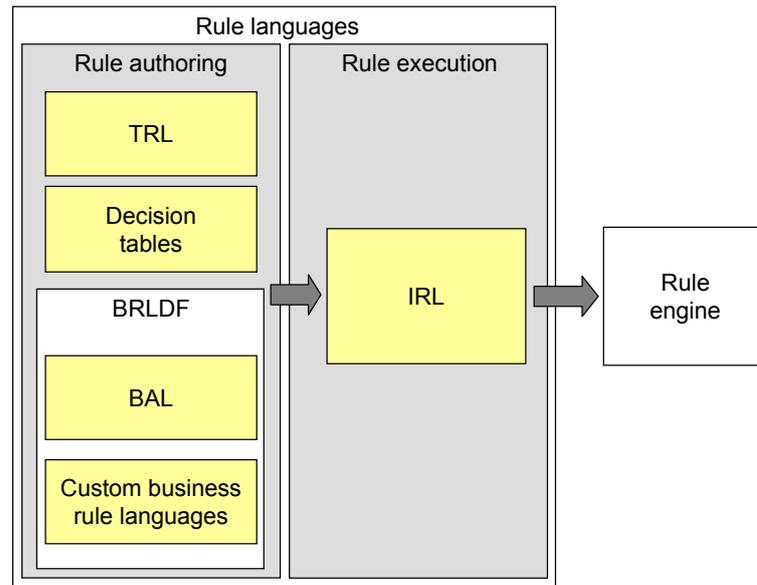
Figure 5.3 JRules architecture.

There are several rules languages to choose from when writing rules that can then access objects in working memory.

The BAL (Business Action Language) is a general purpose business rule language with a syntax close to natural language. The BAL is designed to cover most needs when writing business rules.
The Technical Rule Language (TRL), which is a syntax driven form of the ILOG Rule Language (IRL), and mainly of use to developers.
Decision tables, which are rules composed of rows and columns and used to lay out in tabular form all possible situations that a business decision may encounter. The actions to be taken in each of these situations are specified.
The ILOG Rule Language (IRL) is the language that can be directly executed by the rule engine. The IRL has a Java-like syntax and is mostly used by developers. Business rule languages, like the BAL or the TRL, can be used by a developer to write rules. These are then translated to IRL.

JRules provides support for creating a customizable business rule language, using its Business Rule Language Definition Framework (BRLDF). This enables languages to be defined in XML files. The Business Rule Language Definition Framework (BRLDF) can be used to develop a custom business rule language. The BRLDF sits on top of the token model, which can be used for advanced customization. It seems to us that there is a considerable amount of effort involved in creating a custom business rule language. {25}

The relationships among these rule languages and the high level architecture of JRules is shown in Figure 5.3.

A business rule language is a language that is written against objects contained in a BOM. It is designed to be used by a business analyst or policy manager and uses a business rather than a technical vocabulary. It is important to emphasize that the business level of presentation is achieved by attaching text strings to Java objects – compared to HaleyAuthority, there is no real natural language capability. The Business Object Model is translated into the XOM (eXecution Object Model) which can access Java instances and XML data. The rule engine works on this. To write business rules in Rule Builder you must first have a business rule language and a BOM defined. Alternatively, ILOG Rule Language (IRL) can be used by developers. The BOM defines

the classes and methods to which the business rules will be applied and maps the 'natural language' syntax of the business rule language to these classes.

Based upon our experience, the skills needed to use the business rule language are way beyond those of a typical knowledge engineer, business analyst or policy manager – unless that person is closely coupled with an IT developer, who must continually be on hand to change the BOM in advance of authoring or updating the rules.

When the business rules have been written and tested they are translated into ILOG Rule Language, which is the language understood by the rule engine. During the translation of business rules into execution rules, the BOM classes are translated into XOM classes. Execution rules are business rules that have been translated into IRL for execution in the rule engine.

In order to use English words, the user must first implement them in the Java object model, and then 'translate' them into a specific English text string which can be used across multiple rules. Note, however, that the use of the text string must be exact; the system would not even allow me to use 'a client' when I had previously defined 'the client'. JRules, and for that matter Blaze Advisor too, hasn't any way of dealing of common alternative forms of expression (for example 'the client is risk averse' or 'the client is averse to risk'). It merely concatenates a series of specific text strings for easier readability.

The classes available to the rule engine in the XOM and can be dynamic or native in nature. The XOM classes are said to be **native** if they originate from an existing Java object model or **dynamic** if they originate from XML schemas or Web service schemas. To execute a ruleset, it must first be parsed by a rule engine instance. The XOM classes provide the rule engine with the data required to evaluate the business rules. {46,55}

Data sources to be converted into objects for inclusion in the XOM may be XML, a web service, Java objects, databases, or any combination of them. {46,55}

Web Rule Builder provides rule editing capabilities over an intranet or extranet using a Web browser.

There are good debugging facilities, and consistency checking features let the rule author identify broken or redundant rules. Rule templates are supported to aid rapid rule editing. A template can be based on any business rule language, such as the BAL, the TRL, or a custom language. A template library contains a set of templates and a BOM that defines the vocabulary of these templates. {40,41,44,50}

We consider the report generation capabilities of all three products as adequate. {45}

The product supports different policy and rule expression formats: if/then, constraints, declarative definitions. Rules can also be entered in the form of row-oriented decision tables, which is sometimes useful if the knowledge is presented in this way. However, not every knowledge engineer likes this form because of its 'verbosity': each row in a table is a rule, but a single rule (with ANDs or ORs) can correspond to several rows. In practice, decision tables get far too large to be practical for realistically sized rulebases. However, this feature is useful if one's raw data are presented in a suitable tabular form. {25}

If you need to extract the rules, rule subsumption checks can be use to tidy them up. Rule subsumption detects relationships between rules and helps debug the rulebase. A business rule is said to subsume another if the conditions of the subsumed rule are included in the conditions of the subsuming rule. Thus, if the antecedent of one rule says 'if X is greater than 20 then do A' and another says 'if X is greater than 10 then do B' then the second rule subsumes the first. We need to correct the first rule to read 'if X is between 10 and 20 then do A'. {50}

Rules can be written in English, French and Japanese and embedded in Web services. There are good rule query facilities. Time related conditions and actions can be included in rules but rule history information is limited. Once written, the rules can be deployed – using the APIs – to either J2EE or JSE environments. {4,46,55}

On deployment, instances are inserted into working memory either using keywords within rules or from the application, using this kind of syntax:

```
mycontext.insert(customer);
```

To execute we need to call the following method on IlrContext object.

```
myContext.FireAllRules();
```

The rete algorithm and any custom code now handle the execution.

Incremental development and selective deployment are compromised because of JRules' multiple representations. {33}

The repository handles versioning, permissions, change history, persistence and locking. It also provides a query service that allows one to query business rules using any rule property, including user-defined properties such as business rule author, effective/expiration date, and business rule status. You can also query on classes, attributes and methods referenced in the rules. Queries are written using the Business Query Language (BQL). BQL is a language derived from the BAL tailored for querying rules. BQL has SQL-like syntax. Standard management queries can be created. {5,39}

Ruleflows allow the developer to define the execution order of rulesets. A ruleflow is defined by a diagram (reminiscent of a UML activity diagram) that defines a sequence of tasks (rulesets, other ruleflows or functions) that solve a particular problem or execute a business process. It consists of tasks and transitions between these that define their chaining. A transition can have a guard, which must be true the transition to be allowed. Rule execution is controlled by task properties, which can be set by the user. These properties determine:

the rule ordering, using static or dynamic priorities or following a user-defined sequence;
the rule firing strategy (e.g. fire all eligible rules, or fire one rule and stop); and
the execution algorithm (rete or sequential bytecode generation for optimal performance). {42,43,54}

ILOG claims that JRules conforms with current and emerging Object Management Group (OMG), Java Community Process (JCP), and World Wide Web Consortium (W3C) standards. We did not test this claim. {12}

5.3.1 Related ILOG products

Business Rule Studio is a free extension to IBM's Eclipse Java IDE that will be useful to developers who work in that environment. They can create and edit both business rules in the ILOG rule language and the Java XOM code.

ILOG also offers constraint logic programming tools. These are more specialist and aimed at a particular kind of artificial intelligence search problem where the knowledge is stored in the form of constraints. Inference then proceeds by backward chaining over the constraints and applying mathematical algorithms to search for feasible solutions. Typical applications include planning and scheduling, resource allocation, transport and logistics, and circuit design and verification. We do not consider constraint programming further in this paper.

5.3.2 Conclusions

At the end of our evaluation, we were convinced that JRules was a sound product with a rich range of tools and features for developing business rules management systems. On a purely technical level, most business rule applications can be built using this product. However, it failed to pass our tests for usability by business analysts and, when we scored it against all our evaluation criteria, it came out with the lowest score of the three products considered. Much of the documentation is largely written in terms that can only be fully appreciated by Java programmers. The business analyst is protected from this to some extent but, we felt, not nearly enough. The plethora of rule languages and the translations between them is confusing for the analyst new to the product, as is the distinction between the BOM and XOM. In other words, JRules sacrifices simplicity to its architecture.

We concluded that JRules is a tool that is most suitable for use within projects where considerable technical skills are on call. Even though business analysts can use elements of the Rule Builder, to make this possible will require a significant amount of customization by IT professionals and a deal of tuition. The technical support that we received confirmed this impression; it was prompt and helpful but, on one occasion, I got the impression that anyone who wasn't a Java propeller-head wasn't regarded as fully human.

JRules will sit comfortably in developer-centric organizational cultures, especially where Java and J2EE constitute the prevailing development environment. But we have already warned of the commercial dangers of developer-centric cultures.

5.4 Other products considered and ruled out

There is a large number of products that allow users to develop rule-based systems. Not all of these may be classified as BRMS because some do not offer repository-based rule management. I have not considered such products in this comparison.

RulesPower from the eponymously named company co-founded in 2001 by Charles Forgy, author of the rete algorithm, is intended to allow business management personnel to create and maintain the business logic that represents their business policies. However, the focus of RulesPower is on business process modelling and there is no focus on providing the rule engine as a component within a larger application, which is the focus of our interest and that of most of our clients. RulesPower offers instead a good but monolithic solution to the problem of building a BRMS. [www.rulespower.com]

I also excluded a number of popular open source solutions because they are not rule *management* systems. Examples include Jess and Drools.

Jess is a rule engine and scripting environment written entirely in Java language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA. Jess was originally inspired by the CLIPS expert system language, but has grown into a complete, distinct, dynamic environment of its own. Using Jess, you can build Java software that has the capacity to perform inferences on declarative rules. Jess is small, light, and one of the fastest rule engines available. The Jess language is still compatible with CLIPS, in that many Jess scripts are valid CLIPS scripts and vice-versa. Like CLIPS, Jess uses the rete algorithm. Jess adds many features to CLIPS, including backward chaining, working memory queries, and the ability to manipulate and reason directly about Java objects. Jess is also a powerful Java scripting environment, from which you can create Java objects and call Java methods without compiling any Java code. [http://herzberg.ca.sandia.gov/jess/]

Drools is another augmented implementation of Forgy's rete algorithm tailored for the Java language. Adapting rete to an object-oriented interface allows for more natural expression of business rules with regards to business objects. It is an engine for processing rete graphs and is therefore a purely forward chaining system. Drools wraps the semantics of the normal relational rete into an object-oriented model compatible with Java. Additionally, by mapping to objects, domain specific languages can be created that operate upon an application's own object model. [http://drools.org/]

With Jess and Drools there is the problem of lack of support. Neither are there any high-level authoring and management tools. These deficiencies render such tools unsuitable in most commercial environments.

Computer Associates' CleverPath Aion Business Rules Expert is a descendent of the Aion expert system shell. It is rete-based and offers a component-based development environment. However, it has a slightly monolithic character and the rule engine is not offered as an embeddable component. Nor is it repository-based; rules are stored as a collection of rulesets. CleverPath Aion BRE supports decision tables, dynamic rules, and a sophisticated inference engine. There are links to CA's neural net-based machine learning system, Neugent. [www3.ca.com/Solutions/Product.asp?ID=250]

Mindbox's ART*Enterprise* is another born-again shell and considered monolithic rather than component-based – for all its strengths. [www.mindbox.com]

Corticon is a non-rete-based rule engine. It will generate web service, Java and J2EE applications on top off existing applications but does not offer a set of components for embedding in them. Nor is it repository based. In rete-based engines, the best performance arises when the objects are fairly simple (the number of attributes to be tested is small) and the number of rules is large. However, when the item of work moving through a business process is more complex (such as an insurance claim with all its attendant objects - policy, injury, employer, medical bills, and litigation motions) the number of possible variables that need to be examined by the rules becomes large. The number and depth of association paths to be traversed is also significant. In these scenarios, rete engines do not scale well - their agenda management phase consumes noticeable time. Corticon integrates with Staffware, a leading business process modelling tool and probably shines brightest for this kind of application rather than when applied to normal BRMS problems. [www.corticon.com]

ESI's Logist is a purely forward chaining rule-based expert system shell with a pseudo-natural-language interface. Typically, ESI claim, Logist is used by organizations to offer their clients customized services, to promote customer retention and avoid revenue leakage and billing errors. [www.esi-knowledge.com/BR_logis.asp]

PegaRULES from Pegasystems Inc appears to be a candidate product for this exercise: a BRMS with good Java integration and a repository. Unfortunately, their website provides insufficient information to help decide if the effort spent on evaluation would be worthwhile. Furthermore, there is a number of *faux pas* that inhibit technical credibility; e.g. 'Provides both forward chaining (procedural logic) and backward chaining (goal-based logic)'. Forward chaining is non-procedural of course.
[http://www.pegasystems.com/productsservices/RulesTechnology/RulesTechnology.asp]

As an aside, Netherlands based LibRT offers an interesting complimentary product to any BRMS. LibRT VALENS is claimed to be the first independent product targeted at verifying and validating business rules created in third-party business rules management systems. There is a known relationship with CA's CleverPath Aion and Blaze Advisor. [www.LibRT.com]

6 PRODUCT COMPARISON

As well as running the example knowledgebases supplied with each product, we tested each of them on a small application designed to test the features typically required of a rule-based application. We also used a multi-attribute decision making method to compare the products on a weighted feature-by-feature basis.

6.1 A simple application

The sample application scenario is this. A life assurance company employs sales representatives who visit potential customers in their homes. The reps have laptops or PDAs on which they can perform various financial calculations to do with disposable income, requirements for retirement income, death benefit, school fees, marital outlays, etc. The problem is to add to this an application that can recommend the best type of product for a particular client, based on both the numerical data and more 'soft' factors, such as their personal aversion to or preference for taking risks with their money.

Here are the rules for our simplified life assurance advisory system as they might be in an actual business policy or requirements statement.

> The system needs to recommend a best policy for each client.
>
> An Annuity is best for clients that are retired and risk averse. An Endowment policy is best for clients that are young and not averse to risk. An Equity linked policy is recommended if the client is a mature adult and is risk prone or at least neutral about risk. A Bond linked policy is recommended for a client that is averse to risk unless the client is retired. In any case, we assume that a client is averse to risk if the client has one or more children.

It would normally be good practice to use a more consistent style and sentence structure. The second paragraph might be clearer if written as follows.

> An Annuity is recommended for a client if the client is retired and is risk averse. An Endowment policy is recommended for a client if the client is young and is not averse to risk. An Equity linked policy is recommended for a client if the client is a mature adult and is prone to risk or is neutral about risk. A Bond linked policy is recommended for a client if the client is averse to risk unless the client is retired. A client is averse to risk if the client has children.

The classic approach is to invent some simple pseudocode, readily understandable to most technically savvy knowledge engineers, such as this.

```
Goal = Client.bestProduct
If Client.status is 'retired'
     and Client.preference is 'riskAverse'
     then Client.bestProduct is 'Annuity'
If Client.status is 'young'
     and Client.preference is not 'riskAverse'
     then Client.bestProduct is 'Endowment'
if Client.status is 'matureAdult'
     and Client.preference = 'riskProne' or client.preference =
'riskNeutral'
     then Client.bestProduct is 'EquityLinked'
If Client.preference is 'riskAverse'
     then Client.bestProduct is 'BondLinked'
If Client.children: > 0
```

```
     then Client.preference is 'riskAverse'
```

6.1.1 The application in Blaze Advisor

Here is how the rules came out in the Blaze Advisor SRL rule language.

```
if    client.status is retired
      and   client.preference is riskAverse
      then  {client.bestProduct is "Annuity", return
client.bestProduct}.
if    client.status is young
      and   client.preference <> riskAverse
      then  {client.bestProduct is "Endowment", return
client.bestProduct}.
if    client.status is matureAdult
      and   (client.preference = riskProne or client.preference =
riskNeutral)
      then  {client.bestProduct is "EquityLinked", return
client.bestProduct}.
if    client.preference is riskAverse
      then  {client.bestProduct is "BondLinked", return
client.bestProduct}.
if    client.children > 0
      then  client.preference is riskAverse.
```

Note that the syntax is miles away from our original statement of the rules in English. However, it is only little changed from my crude pseudocode, except for the need to include the return statements. It looks a little more like a programming language than the original; but it is a fairly easy language to learn for a technically competent knowledge engineer. Using SRL's alternative syntax would also have allowed me to write in a different style, for example:

```
if    client's status is retired
      and   client's preference is riskAverse
      then  {client's bestProduct is "Annuity",
      return client's bestProduct}.
```

There are still five rules. The equivalent of the goal statement is two 'event rules' as follows.

```
event rule getStatus is
whenever status of a Client is needed do { it.status =
promptEnumerationItem(status, "What is the client's status?")}

event rule getChildrenNumber is
whenever children of a Client is needed do { it.children =
promptInteger( "How many children has the client?")}

event rule getPreference is
whenever preference of a Client is needed do {  it.preference =
promptEnumerationItem(preference,"What is the client's risk
preference?")
```

The 'it' object is the current instance (this in Java; self in Smalltalk). This is looking a bit more like programming but it's not that frightening. Of course, we had to create the attributes of a Client class and two enumeration lists. The only other programming was the creation of a main routine to test the rule execution. Here it is.

```
client is a Client.
print ("A few questions about the life to be assured...").
      client.bestProduct = apply matchingRules(client).
print ("The best product for this client is " client.bestProduct).
```

Here are the results of test executions, showing that backward chaining is working exactly as expected.

```
A few questions about the life to be assured...
Question: What is the client's status?
Answer: retired
Question: What is the client's risk preference?
Answer: riskAverse
The best product for this client is Annuity

A few questions about the life to be assured...
Question: What is the client's status?
Answer: matureAdult
Question: What is the client's risk preference?
Answer: dontKnow
Question: How many children has the client?
Answer: 2.0
The best product for this client is BondLinked

A few questions about the life to be assured...
Question: What is the client's status?
Answer: matureAdult
Question: What is the client's risk preference?
Answer: riskNeutral
The best product for this client is EquityLinked

A few questions about the life to be assured...
Question: What is the client's status?
Answer: retired
Question: What is the client's risk preference?
Answer: riskProne
Question: How many children has the client?
Answer: 0.0
The best product for this client is unavailable
```

The last result merely shows that the ruleset does not cover the state space and that more rules would be needed in a live application.

However, it should be noted that rules 2 to 3 were given a higher priority than rules 1 and 4. If we relax this and also move rule 4 to the beginning of the list of rules, then rule 4 fires first and we get an erroneous result of BondLinked for a retired and risk averse client. In other words, rule order can have an effect that may not be obvious prior to debugging. Against this, the ability to assign priorities to individual rules is a powerful tool in the hands of an analyst that knows what they are doing.

6.1.2 The application in JRules

To set up the same rules in JRules required much more effort. Creating the repository involved specifying its location in the directory tree. Then the BOM classes had to be created and their properties set. This involved naming attributes and providing more English-like alternative names. The dialogue boxes exposed several programming concepts for the attributes such as their data types: float etc. Following the tutorial, I created methods such as GetStatus with a string return value and renamed the latter as 'the client status'. With my business analyst's hat on, I felt uncomfortably exposed to these details. Entering the rules was better, once the initial investment in the object model had been made. This used drop down menus showing attributes from the object model and various relational operators. The rules looked pretty much like my original pseudocode.

```
If
      the client status equals 'retired'
      and the client preference is 'riskAverse'
Then
      the best product is 'Annuity'
```

... and so on. Closer to English than Blaze Advisor but much more of a pain to set up.

Two things should be noted here. First, JRules requires the creation of an object model as a pre-requisite to creating these rules. Second, this English-like syntax is only brought about by creating strings that are attached to the objects in the JRules interface.

As with Blaze Advisor, testing the rules required writing a main routine; but the syntax of Java was needed to create test instances using the Java 'new' keyword. There was another way to do this in the rule builder, involving use of an 'assert' dialogue to set up attribute values. It didn't have the interactive feel of Blaze Advisor. Setting up list of allowed values (enumerated types) exposed a useful UML graphical view of the BOM. Testing the rules, all the results came out as expected. Backward chaining seems to be working, even though ILOG told me (during one of their excellent web seminars) that Java programming would be required to implement backward or opportunistic chaining in this manner. Evaluation time constraints precluded me from pursuing this.

At this point, I had formed a number of impressions about the ILOG product.

- # It was clearly up to doing the job I had in mind.
- # The rules looked a bit friendlier than they did in Blaze Advisor SRL but the object model looked a bit more like a tech-fest.
- # It took quite a bit more work to get to the same point and I felt rather swamped by the detail.
- # Even with the advantage of the UML graphical editor (which none of the other products boast), only someone who was comfortable with Java would take to it naturally.
- # For a typical business analyst, set up time would be considerably longer than Blaze Advisor or, indeed, HaleyAuthority (see below).

Against this, price, rule execution performance and memory footprint were all better than Blaze Advisor in its default settings for its rete mode. Time to look at our third shortlisted product. {1,54,58}

6.1.3 The application in HaleyAuthority

In HaleyAuthority, the rules came out in much plainer English. I was able to write statements as sentences almost identical to the more structured version of the original policy. My first version came out like this.

BestProductRules
Statements:
> An Annuity is recommended for a client only if the client is averse to risk and the client is retired
> An Endowment policy is recommended for a client only if the client is young and the client is not averse to risk
> An Equity linked policy is recommended for a client only if the client is a mature adult and the client is prone to risk or is neutral about risk
> A Bond linked policy is recommended for a client if the client is averse to risk unless the client is retired
> A client is averse to risk if the client has children

As I familiarized myself with the product, I soon discovered a more structured way to write and present the rules, as follows.

BestProductRules

Statements:

> An Annuity is recommended for a client
> > *only if* : the client is retired
> > *only if* : the client is averse to risk
>
> An Endowment policy is recommended for a client
> > *only if* : the client is young
> > *only if* : the client is not averse to risk
>
> An Equity linked policy is recommended for a client
> > *only if* : the client is a mature adult
> > *only if* : the client is prone to risk or is neutral about risk
>
> A Bond linked policy is recommended for a client
> > *if* : the client is averse to risk
> > *unless* : the client is retired
>
> A client is averse to risk
> > *if* : the client has children

We have used the 'applicability condition' rule style here. There are three kinds of applicability condition: 'if' conditions are ORed and 'only if' conditions ANDed. 'Unless' conditions are self-explanatory and help to make rulesets (modules) more concise.

Realizing this I went back to Blaze Advisor and corrected one of my rules to read as follows.

```
if    client.preference is riskAverse
      and client.status is not retired
then  {client.bestProduct is "BondLinked", return
      client.bestProduct}.
```

A similar change to the JRules ruleset also used the and/not construct to represent 'unless'.

As explained in Appendix 8.2, writing rules in HaleyAuthority is guided by Haley's knowledge acquisition method:

1. Identify the decisions to be made
2. For each decision write the policies or rule logic and indicate any exceptions or qualifications
3. Define the business vocabulary and phrasings that HaleyAuthority needs to know about in order to understand and interpret the rules that you have defined
4. Test and simulate your rules incrementally as you go

This makes the applicability condition style very natural.

Note that the first rule can also be written (as above) as

An annuity is recommended for a client if the client is averse to risk and the client is retired

In fact, if you type this in, HaleyAuthority suggests the next permissible words as you type.

Rule execution transcripts for a few test cases were put out by the system as follows.

Test Case: Clapton
Execute: condition 10: the client is a mature adult .
Execute: condition 11: the client is prone to risk or is neutral about risk .
Execute: statement 9: An Equity linked policy is recommended for a client .

Test Case: Idle
Execute: condition 4: the client is retired .
Execute: condition 15: the client has children .
Execute: statement 14: A client is averse to risk .
Explanation: [client] 32 is averse to risk
Execute: condition 17: the client is averse to risk .
Execute: condition 20: the client is retired .

Execute: condition 5: the client is averse to risk .
Execute: statement 3: An Annuity is recommended for a client .

Test Case: Jong
Execute: condition 7: the client is young .
Execute: condition 17: the client is averse to risk .
Execute: statement 12: A Bond linked policy is recommended for a client .

Test Case: Leach
Execute: condition 4: the client is retired .
Execute: condition 5: the client is averse to risk .
Execute: condition 17: the client is averse to risk .
Execute: condition 20: the client is retired .
Execute: statement 3: An Annuity is recommended for a client .

Test Case: Redhand
Execute: condition 7: the client is young .
Execute: condition 8: the client is not averse to risk .
Execute: statement 6: An Endowment policy is recommended for a client .

All the results are as expected.

In all three products, priorities can be set for rule modules. This enables the analysts to control the order of execution. In Blaze Advisor, for example, priorities can be set at the individual rule level, giving some finer control but at the risk of complexity. To do this in HaleyAuthority, one has to write rules about the priorities in the same natural language style as the main rules. One approach to this, in the above example, is as follows. First, deduce possible or allowable conclusions and then write explicit policies about such preferences. Another approach is to use unless conditions, but still using a representation that is aware of preferences.

The first approach might write policies such as 'An annuity may be (could be) recommended for a client'. Subsequent policies may state:

```
A bond linked policy is recommended for a client
    only if: An annuity may not be recommended for the client
    only if: A bond linked policy may be recommended to a client
```

The second approach would generalize to the understanding that certain products are preferred to others but still distinguishes between what 'may be' and what 'is'. For example:

```
An annuity is preferred to a bond linked policy
    if: An annuity may be recommended for a client
A product is recommended to a client
    only if: the product may be recommended for the client
    only if: no product is preferred to the first product
```

Either of these approaches may be employed to convey the additional desired knowledge that certain products are 'more applicable' or 'preferable' under certain circumstances. The approach is slightly wordy but very clear in meaning and less prone to error.

6.2 Comparative evaluation

We evaluated the three products in three different scenarios, representing three different sets of cultural and technical imperatives and concerns.

In scenario 1, the customer is an early adopter where the users are keen to be involved in rule creation, with the help of their colleagues in IT. The business analysts are not typically skilled programmers but do understand the business and their clients well. The IT department is relatively small. The application is a knowledge intensive extension to a larger business system. Imagine, if you will, a sales advisory system, like the one used in our example above, a system for regulatory

compliance or one for benefit entitlement. In this scenario, therefore, the emphasis is on the ease of rule authoring and maintenance by users or relatively non-technical business analysts.

Scenario 2 maintains the viewpoint of Scenario 1 but lays greater stress on the level of integration with the commercial and technical environment. Furthermore, in this scenario, rule input by users and non-technical business analysts is not required, because users are too busy. They will maintain rules via custom applications, where appropriate. Scenario 2 envisages a large, more conventional IT department where the users are available for knowledge elicitation but do not have the time or inclination to create the rules themselves. There is a strong mainframe culture and the applications must be integrated closely with the legacy. Imagine, in this case, an application like credit card fraud detection or credit scoring where the BRMS will be closely integrated with multiple existing databases. In this scenarios the users are busy and do not want to interact with the rules often. They do, however, require rule-based data validation at the time of data entry. Development resources are available to create the rules and write predefined rule maintenance applications.

Scenario 3 abandons the emphasis on rule creation by end users and assumes a strong commitment to a technical architecture, such as J2EE. This scenario is not explored in this version (Version 2) of this report. In scenario 3, the client is committed to a modern distributed computing architecture such as J2EE. There is a strong Java culture and the users will not be involved in rule maintenance except via the good offices of programmers and analysts. Imagine, here, an investment bank developing a credit rating system that must integrate with an existing J2EE application suite and architecture.

The scores in each scenario are the same but the weightings vary from scenario to scenario, depending on the technical and cultural imperatives given in each situation.

In every scenario, we regard support for object modelling features such as inheritance and encapsulation as essential. All three products scored well on this.

Help and explanation are limited in all three products. In each case they can be coded in applications. I can also envisage writing rules in HaleyAuthority to provide this feature, but did not have time to test this. In Blaze Advisor, one might include help facilities in an RMA. In JRules, custom Java code would do the trick because, of course, you can do anything in Java.

Uncertainty management is limited in all three products. Blaze Advisor offers scorecards. Haley relies on uncertain linguistic formulations. Alternative logics were not supported by any of the products considered, but Haley permits deontic and modal constructions (should have, ought to, might have, etc.) to be expressed in natural language. {36}

To get an idea of memory footprints we used the Windows task manager and conducted an informal survey of applications known to us to estimate performance. It should be pointed out here that our test application had very few rules and the size of a knowledgebase in memory does not necessarily grow as more rules are added. We are relying, therefore, largely on anecdotal and published estimates in this respect.

The results of our evaluation of features are summarized in the multi-attribute decision making (MADM) analyses shown in tabular form in the following subsections. Scores are on a scale from 0 to 5 with 0 meaning 'not worth considering at all' and 5 being 'as good as could be expected'.

The weightings given to each attribute reflect the needs of the particular environment envisaged. Similar weightings would apply to many application types such as loan or credit approval, fraud detection, eligibility screening and so on. However, there is obviously scope for amending the weights in particular contexts. You can easily rerun the analysis with your own weights. The final scores are the sum of the individual scores for each product multiplied by their weightings (something like SUMPRODUCT($B3:$B82,C3:C82) in Excel). The normalized scores are arrived at by dividing the totals by the sum of the weights (if non-zero) and are thus on a scale from 0 to 1.

Clearly, there is a subjective element involved in arrived at some of the scores but we have tried to minimize the effect of this. In point of fact, some of the factors that lead to product acceptance within an organization are often subjective ones.

Bearing these points in mind, the analysis should be taken as a fairly rough indication of which product is fittest for purpose rather than as a definitive description. However, with so many attributes, it is actually quite hard to bias the result in practice; so I am fairly confident of the conclusions in terms of rank ordering.

6.2.1 Evaluation in scenario 1

In this scenario, our evaluation concentrated on the ease with which a business analyst could create a business rules management system. We were also interested in the degree of coverage of the full development lifecycle from knowledge capture to implementation and testing and the level of integration of the knowledge management tool(s) with the rule engine: the number of steps involved, their relative complexity and the level of automation of each step.

The Blaze Advisor SRL syntax most closely resembles pseudo-code and is far from easily understood natural language. While the syntax would be relatively easy to learn for a technical IT professional, it would be very difficult for a business professional. To be fair, Fair Isaac would expect such a user to interact with an RMA rather than the SRL. The JRules syntax isn't much better from this point of view and, even then, I had to sweat hard at the object model before I could write these kinds of rules.

HaleyAuthority's deep understanding of grammar enables users to use alternative syntactic constructs at will, as long as the semantic aspect (the concept model) remains the same. This provides great flexibility to users. It is also important to note that allowing you the map an external implementation model to a business-level conceptual model helps separate the business rules from the implementation model in all three products, with all the benefits that this entails. For example, as a .NET or Java application changes, you only need change the mappings and not the statements.

On price, we asked each company to quote a price based on the following assumptions.

The target application is an investor needs/life product matching system to be used by sales staff in the field and over the telephone. The client is a life office. The application architecture is yet to be decided but will probably be either J2EE or .NET using SQL Server. There will be a development team of six multi-skilled people including a project manager, three development specialists and two business analysts. We think that five development licences should suffice.

Deployment will be on a variety of machines, from aging (Win98 to XP) laptops to Palm pilots and XP desktops (often to be used by the laptop/palm users when in the office), although we did not ask the vendors for any architectural advice (for example, Fair Isaac would likely recommend using its SmartForms product on PDAs if the rules task was data capture). Estimates are as follow.

- # 40 Laptops
- # 10 Palm pilots
- # 30 Desktops

They were asked to estimate the total cost for licenses, training costs for the six development team members and maintenance fees. Only ILOG was unable to supply a quotation in time for our deadline, so we estimated their costs on the basis of information published on the internet. We then converted the figures into a ranking scheme, as shown in the tables below. {1}

Usability is at the best of times a subjective matter. All three products passed our tests for basic usability but HaleyAuthority and Blaze Advisor came out strongest in this respect. After getting over the different approach found in HaleyAuthority, we concluded that, on balance, it was the most usable rule authoring product for this scenario.

In evaluating and comparing the products we used many of the criteria already discussed as well as a number of others. We considered the accessibility and ease of use for untrained users, asking what pre-requisite skills were needed for effective use of the tool by subject matter experts, business analysts and IT staff. A high score was achieved when all these were accommodated well. Was there a clear and consistent separation of business knowledge from implementation

details? How expressive was the language for business users compared with implementation – in particular, the ability to support description of rules in or near natural language? Relevant to this is the question of who does the translation of business rules into the execution language syntax. HaleyAuthority offered the highest level of automation in this respect.

Meta model availability and extensibility refers to the availability of built in concept libraries covering such things as time, units and quantities. Haley, in particular, provides a rich library of concepts that can be readily extended into domain specific areas (available as add-ons).

For each product, we compared the completeness of representation of concepts within the knowledge base. Does it include concepts, relationships, vocabulary, phrasings, definitions, policies, constraints, rules, and so on? All product score well but HaleyAuthority's notion of 'understanding' a rule relative to the object model is particularly powerful. It provided the best support for documenting, defining and standardizing an organizational vocabulary and it prevent ambiguity in our rule definitions.

The results of our evaluation of features are summarized in the MADM analysis shown in Table 6.1.

My interpretation of the results for this scenario, therefore, is this. HaleyAuthority scores significantly higher than the other two products based both on the feature-by-feature comparison and my subjective evaluation.

6.2.2 Evaluation in scenario 2

Scenario 2 is a large conventional IT department. The users are busy and have no desire to engage in rule authoring although they will need to modify rules and need rule-based data validation. Developers and trained analysts will write and test the rules. Resources are also available to create custom rule maintenance applications. There is an historic COBOL and mainframe culture and applications must be closely integrated with channel-hungry mainframe applications: both legacy and evolving. Nevertheless, there is a commitment to service oriented architecture and rulesets must be presented to applications as services; as in Scenario 1. RUP is used as the main development method for new systems.

We assume that this scenario encounters similar price and performance characteristics to Scenario 1.

In this scenario the natural language input that so strongly characterizes Scenario 1 is regarded as a positive disadvantage.

One of Blaze Advisor's great strengths is its RMAs. Building one of these involves abstracting from the rules to create more generic rule templates within which the user may select values, ranges, objects and so on. A template contains value holders and contents. Each value holder specifies a particular type of value, or enumeration list of values. The contents contain standard rules and includes embedded placeholders. Each placeholder refers to a value holder. The value defined by the value holder will be inserted into the contents to replace the placeholder. Sets of values corresponding to the value holders are stored in separate repository items called instances. When an instance is resolved, the result is a set of rule entities.

A template can be as simple as a single SRL statement that exposes a single value for editing such as "theCustomer.age > [minimum age]". Templates can get quite complex too. They can include multiple value holders, and can define complete rule entities or even multiple entities. In addition, a value holder can hold multiple values of the same type. A value holder can also refer to another template. In such a case, a placeholder for the value holder will resolve into a resolved instance of the referred template. This permits flexible, hierarchical structures that support precise control over what can and cannot be edited. For example, a ruleset template could contain a value holder referring to a rule template that defined a particular form of a rule. The rule template could in turn contain value holders pointing to various code templates that define particular conditions and actions that are valid for the rule.

RMAs are generated from the rule templates within a web browser.

Creating a rule template required a careful reading of the tutorial and some practice. I copied the text of one of my rules into the content section of the template and then replaced the parts that I wanted to be modifiable with placeholders. I concluded that that I would have benefited from more practice or some training. On the other hand, generating a crude RMA from the template was relatively straightforward.

The results of our evaluation of features are summarized in the MADM analysis shown in Table 6.2.

In this situation we find that JRules catches up with HaleyAuthority and Blaze Advisor, with its RMAs and SmartForms, overtakes it. Adding my subjective interpretation and in recognition of the fact that my decision tables are only an approximate guide, my interpretation of the results for this scenario, therefore, is this. Blaze Advisor scores significantly higher than the other two products based both on the feature-by-feature comparison and my subjective evaluation.

6.2.3 Evaluation in scenario 3

This Section is still under development in this version of the report. *A priori* we would expect JRules to do better in this scenario. Version 3 of this report will expand this scenario and evaluate the later Version 5 of JRules.

	Attributes	Weight	Auth-ority	Blaze	J Rules
	General Attributes				
1	Price	*2*	*4*	*2*	*3*
2	Defect free	*3*	*4*	*4*	*4*
3	Ease of installation	*3*	*5*	*4*	*2*
4	Interface/Usability	*5*	*4*	*3*	*3*
5	Repository-based	*5*	*5*	*5*	*5*
6	Technical support	*5*	*5*	*5*	*3*
7	Availability and coverage of professional services – including training	*5*	*4*	*4*	*4*
8	Availability of a defined knowledge engineering method	*3*	*5*	*3*	*3*
9	Coverage of full-lifecycle development from knowledge capture to implementation and testing	*5*	*5*	*5*	*4*
10	Availability of a knowledge capture and management (KM) tool	*5*	*5*	*4*	*4*
11	Plans for forthcoming upgrades	*2*	*4*	*4*	*4*
12	Adherence to IT industry standards	*3*	*3*	*4*	*4*
	Integration of KM tool with the rule engine				
13	Steps involved	*4*	*4*	*3*	*3*
14	Simplicity	*4*	*4*	*5*	*3*
15	Level of automation	*4*	*5*	*5*	*4*
	Knowledge Capture and Management Tool				
16	Accessibility and Ease of use by untrained users – pre-requisite skills for effective use of the tool (perspectives of Subject matter experts, business analysts, IT staff)	*5*	*4*	*3*	*2*
	Clarity, depth and coverage of supplied documentation				
17	User manuals	*4*	*5*	*4*	*3*
18	Examples	*3*	*4*	*4*	*3*
19	Separation and consistency of business knowledge from implementation details	*4*	*5*	*3*	*1*
20	Availability and expressiveness of language for business users vs. implementation (in particular ability to support description of rules in natural language)	*5*	*5*	*3*	*2*
21	Automatic translation of business statements into the execution language/syntax?	*4*	*5*	*4*	*3*
	Knowledge Management features				
22	Meta model availability and extensibility	*4*	*4*	*4*	*2*
23	Completeness of representation within the KB (concepts, relationships, vocabulary, phrasings, definitions, policies, constraints, rules)	*4*	*4*	*4*	*4*
24	Support for documenting, defining and standardizing an organizational vocabulary and using it to prevent ambiguity in rule definitions	*4*	*4*	*3*	*3*
25	Support for different rule expression formats (if/then, declarative/definitional statements, constraints, general English statements)	*4*	*5*	*3*	*4*
26	Change management and version control features	*5*	*5*	*5*	*5*
27	Support for archetypes, templates, overrides/ specialization, and exclusions – Including the ability to specify the conditions under which an override or exclusion should apply (i.e. applicability conditions)	*4*	*4*	*3*	*3*
28	Reuse of applicability statements in relation to rules and rule sets (drag & drop)	*4*	*4*	*2*	*3*
29	Table and decision table support	*3*	*2*	*4*	*4*
30	Decision threshold support	*2*	*3*	*4*	*2*
31	Automatic, multiple, cross-referencing of rules and concepts (ontology)	*5*	*5*	*3*	*4*
32	Ability to proactively check for ambiguity in statements	*4*	*5*	*3*	*4*
33	Support for incremental development (selective deployment)	*5*	*4*	*4*	*2*

34	Ability to create deployment polices with future effective and expiration dates	3	5	5	4
35	Support for multiple evaluation and control strategies	4	4	4	5
36	Support for inexact reasoning	4	2	3	1
37	Support for reasoning with time	3	4	2	1
38	Built-in support for importing, mapping to business vocabulary, usage, orchestration, and integration with external data representations, procedures, methods (XML, .NET, Java)	2	3	3	3
	Multi-user features				
39	Concurrent KB development. Support for a team repository	3	5	5	4
	Built in support for testing and simulation within the KM tool – without the need for an external implementation				
40	Ability to specify test cases	5	5	4	4
41	Ability to execute test cases	5	4	5	3
42	Code generation capabilities	5	5	4	3
43	Ability to define/maintain/organize rule groups/sets	5	5	5	5
44	Debug/trace facilities	5	4	4	3
45	Report generation capabilities	4	3	3	3
46	Support for Web Services	4	5	4	4

	The Rule Engine				
47	Support for rete	5	5	5	5
48	Support for backward and mixed chaining	5	3	4	2
49	Automatic truth maintenance	4	5	4	4
50	Support for debugging/audit trail of rule firing	5	5	4	4
51	Support for XML input	4	5	4	4
52	Automation of integration with Java, .NET, and databases	5	4	3	3
53	Ability to handle large no. of rules	5	5	4	5
54	Performance and scalability (ability to handle large no. of concurrent requests/transactions, users, and rule executions)	5	5	3	4
55	Ease of integration with external applications (e.g. Web Services, embedded)	4	3	3	3
56	Does the language have the power to handle procedural or technical functions without requiring a call to an external routine?	4	2	3	4
57	Can the language call on external and mathematical routines when desired?	4	4	5	5
58	Memory footprint (suitability for embedding in small devices)	4	5	4	4
59	Availability across multiple platforms	4	4	4	3
60	Availability of alternative interfaces (e.g. C, C++, Java, .NET)	3	5	3	4
61	Support for dynamic 'hot' deployment	4	4	2	3
62	Runtime rule updates	2	4	4	4
63	Support for multiple concurrent KBs	4	4	4	3
64	Ability to update KBs with minimal user impact	5	5	5	3
65	Ability to handle different deployment dates	4	5	5	4
66	Maturity in rule engine market with proven rete-based implementations.	3	4	5	5
67	Conflict resolution	4	3	4	3
68	Interactive testing	4	1	3	1

	Other factors (Technical environment, culture, etc.)				
69	Integration with supplier's product range	0	1	5	3
70	Need to leverage technical skills	0	1	4	5
71	Suitable for mainframe culture	0	3	5	3
72	Integration with J2EE environment	0	2	3	5
73	UML object model input	0	0	1	3
74	COBOL integration	0	1	4	2
75	Custom rule maintenance screens	0	0	4	1
76	RUP plug-ins available	0	0	3	0
77	Supplier involved with standards bodies	0	1	4	4
78	Foreign (i.e. not English) Language support	1	0	0	3
79	Java compatible coding style	0	2	2	5

Total weights and scores	*1375*	1172	1053	935
Normalized scores		**0.85**	**0.77**	**0.68**

Table 6.1 Multi-attribute decision making analysis. Scenario 1.

	Attributes	Weight	Auth-ority	Blaze	J Rules
	General Attributes				
1	Price	3	4	2	3
2	Defect free	4	4	4	4
3	Ease of installation	3	5	4	2
4	Interface/Usability	4	4	3	3
5	Repository-based	5	5	5	5
6	Technical support	5	5	5	3
7	Availability and coverage of professional services – including training	5	4	4	4
8	Availability of a defined knowledge engineering method	3	5	3	3
9	Coverage of full-lifecycle development from knowledge capture to implementation and testing	5	5	5	4
10	Availability of a knowledge capture and management (KM) tool	3	5	4	4
11	Plans for forthcoming upgrades	2	4	4	4
12	Adherence to IT industry standards	4	3	4	4
	Integration of KM tool with the rule engine				
13	Steps involved	3	4	3	3
14	Simplicity	3	4	5	3
15	Level of automation	3	5	5	4
	Knowledge Capture and Management Tool				
16	Accessibility and Ease of use by untrained users – pre-requisite skills for effective use of the tool (perspectives of Subject matter experts, business analysts, IT staff)	1	4	3	2
	Clarity, depth and coverage of supplied documentation				
17	User manuals	4	5	4	3
18	Examples	3	4	4	3
19	Separation and consistency of business knowledge from implementation details	2	5	3	1
20	Availability and expressiveness of language for business users vs. implementation (in particular ability to support description of rules in natural language)	0	5	3	2
21	Automatic translation of business statements into the execution language/syntax?	2	5	4	3
	Knowledge Management features				
22	Meta model availability and extensibility	3	4	4	2
23	Completeness of representation within the KB (concepts, relationships, vocabulary, phrasings, definitions, policies, constraints, rules)	4	4	4	4
24	Support for documenting, defining and standardizing an organizational vocabulary and using it to prevent ambiguity in rule definitions	2	4	3	3
25	Support for different rule expression formats (if/then, declarative/definitional statements, constraints, general English statements)	4	5	3	4
26	Change management and version control features	5	5	5	5
27	Support for archetypes, templates, overrides/ specialization, and exclusions – Including the ability to specify the conditions under which an override or exclusion should apply (i.e. applicability conditions)	4	4	3	3
28	Reuse of applicability statements in relation to rules and rule sets (drag & drop)	0	4	2	3
29	Table and decision table support	4	2	4	4
30	Decision threshold support	5	3	4	2
31	Automatic, multiple, cross-referencing of rules and concepts (ontology)	3	5	4	4
32	Ability to proactively check for ambiguity in statements	3	5	3	4
33	Support for incremental development (selective deployment)	5	4	4	2

34	Ability to create deployment polices with future effective and expiration dates	3	5	5	4
35	Support for multiple evaluation and control strategies	3	4	4	5
36	Support for inexact reasoning	4	2	3	1
37	Support for reasoning with time	3	4	2	1
38	Built-in support for importing, mapping to business vocabulary, usage, orchestration, and integration with external data representations, procedures, methods (XML, .NET, Java)	3	3	3	3
	Multi-user features				
39	Concurrent KB development. Support for a team repository	3	5	5	4
	Built in support for testing and simulation within the KM tool – without the need for an external implementation				
40	Ability to specify test cases	5	5	4	4
41	Ability to execute test cases	5	4	5	3
42	Code generation capabilities	4	5	4	3
43	Ability to define/maintain/organize rule groups/sets	5	5	5	5
44	Debug/trace facilities	5	4	4	3
45	Report generation capabilities	4	3	3	3
46	Support for Web Services	3	5	4	4

	The Rule Engine				
47	Support for rete	5	5	5	5
48	Support for backward and mixed chaining	5	3	4	2
49	Automatic truth maintenance	4	5	4	4
50	Support for debugging/audit trail of rule firing	4	5	4	4
51	Support for XML input	2	5	4	4
52	Automation of integration with Java, .NET, and databases	5	4	3	3
53	Ability to handle large no. of rules	5	5	4	5
54	Performance and scalability (ability to handle large no. of concurrent requests/transactions, users, and rule executions)	3	5	3	4
55	Ease of integration with external applications (e.g. Web Services, embedded)	3	3	3	3
56	Does the language have the power to handle procedural or technical functions without requiring a call to an external routine?	4	2	3	4
57	Can the language call on external and mathematical routines when desired?	4	4	5	5
58	Memory footprint (suitability for embedding in small devices)	2	5	4	4
59	Availability across multiple platforms	4	4	4	3
60	Availability of alternative interfaces (e.g. C, C++, Java, .NET)	2	5	3	4
61	Support for dynamic 'hot' deployment	4	4	2	3
62	Runtime rule updates	4	4	4	4
63	Support for multiple concurrent KBs	2	4	4	3
64	Ability to update KBs with minimal user impact	5	5	5	3
65	Ability to handle different deployment dates	4	5	5	4
66	Maturity in rule engine market with proven rete-based implementations.	5	4	5	5
67	Conflict resolution	4	3	4	3
68	Interactive testing	5	1	3	1

	Other factors (Technical environment, culture, etc.)				
69	Integration with supplier's product range	5	1	5	3
70	Need to leverage technical skills	4	1	4	5
71	Suitable for mainframe culture	5	3	5	3
72	Integration with J2EE environment	0	2	3	5
73	UML object model input	2	0	1	3
74	COBOL integration	5	1	4	2
75	Custom rule maintenance screens	5	0	4	1
76	RUP plug-ins available	4	0	3	0
77	Supplier involved with standards bodies	3	1	4	4
78	Foreign (i.e. not English) Language support	1	0	0	3
79	Java compatible coding style	0	2	2	5

Total weights and scores	*1390*	1050	1094	929
Normalized scores		**0.76**	**0.79**	**0.67**

Table 6.2 Multi-attribute decision making analysis. Scenario 2.

7 CONCLUSIONS

Along with service oriented architectures and component-based development, business rules management systems are an essential component of modern agile businesses. They vastly reduce the problems associated with the evolution of complex and volatile business strategies and policies.

We identified three enterprise-class BRMS products that can be used within a component-based development organization. All three products are capable of delivering effective solutions. However, there is a number of factors that discriminate among them.

The basic JRules demo was a system for applying sales discounts to the item in a shopping basket of the type one finds on internet sites. There was a J2EE demo, but it would not run in my standalone environment. Other demos are embedded in the JRules documentation as applets. I ran one that painted an aquarium on my screen that was too big for the screen size. It let me fire rules one at a time. At each rule firing a fish changed position and species. Unimpressed, I went back to the discounts example. No really sophisticated examples were included with the evaluation pack.

In contrast, Blaze Advisor offers a loan approval application demo with a more realistic feel to it, as well as a library of 192 examples and additional tutorials.

Haley also provided large and realistic demos. I tried one of them, a fragment of a system to approve tax credits, in all three products and concluded that setting up a large rulebase was faster in Haley HaleyAuthority than either of the other two products. The most time-consuming of the three was JRules, but for a skilled Java programmer, building on an existing object model, this may be much better. I also exercised a Haley demo that showed what could be done in terms of effective dates and was impressed.

Our weighted multi-attribute analysis gave the following results for Scenarios 1 and 2.

In Scenario 1, HaleyAuthority had the highest score: 85% of the maximum possible score. Blaze Advisor came second with 77%. JRules scored 68%; all very respectable scores, indicating that these are all good products.

HaleyAuthority is ideal for situations where users and non-technical business analysts need to create and maintain the rules and where development resources are at a premium. If you really want to engage users, as well as business analysts, in the development process and thus reduce the time to market of new versions of an application, as policy evolves, then HaleyAuthority is the indicated choice. Its natural language syntax capability and automatic inferencing and code generation facilities make it the clear winner in Scenario 1. We predict that it will reduce knowledge base development and maintenance costs significantly, compared to its competitors. HaleyAuthority is the right product when you have good access to users and domain experts and need them to help create and maintain the rulebase.

In Scenario 2, Blaze Advisor had the highest score: 79% of the maximum possible score. HaleyAuthority came second with 76%. JRules scored 67%; all respectable scores again.

Blaze Advisor is most suitable for environments where multiple deployment types may be required and business users require customized rule maintenance. We think that it is a much more productive environment than JRules and that it puts the business more in control of application development than the latter. From the point of view of the rule language, it stands midway between JRules and HaleyAuthority, but the rule maintenance application features make it the most suitable in this scenario. Adopting Blaze Advisor will, we think, reduce development and deployment times and is much more suitable then JRules for use by technically inclined business analysts who are not necessarily expert in Java technologies. However, the rule syntax is still rather opaque to business users so that RMAs are essential. Blaze is a mature product that technically savvy business analysts can use to create rule-based applications providing they take the time to learn the product.

If your culture is developer-centric – i.e. the developers create applications largely in isolation from the users after an initial period of knowledge gathering and during acceptance testing – then ILOG JRules is a viable option. Developers will code and largely maintain the knowledge base. Creating and changing the application may be subject to the usual application backlog delays. Users may read and understand the rules, providing that enough developer effort has been put into the rule language customization that the product makes possible. My feeling is that you should only consider JRules if you develop under a J2EE or like environment, although there is now a new .NET product (not evaluated). {60}

JRules, Blaze Advisor and HaleyAuthority all have the features needed to support enterprise projects: multiple views of the same rules, rapid code deployment for various installations, easily maintainable code, version control, structured user access, excellent debugging tools, and English-like rule-building languages that makes maintaining rules easy for developers if not for business analysts and users. Haley's natural language approach is far superior to that of either of its competitors for pure rule writing, whereas for custom rule maintenance by business users, Blaze Advisor's approach may be preferable.

All three products can be used to implement Morgan's recommendations on rule syntax and style but it is a lot easier to do this in HaleyAuthority. The use of a rule maintenance application, however, might make Morgan's recommendations irrelevant.

If performance is a significant factor then JRules will be preferred over Blaze Advisor and Haley scores even higher than JRules in this respect. Similarly, if execution on small memory devices (e.g. PDAs, embedded systems) is critical then you will tend towards the Haley solution, followed by ILOG and then Blaze Advisor.

All three products have weaknesses as well as strengths. For Blaze Advisor, these are the need to understand the technology, price and what is required to obtain maximum performance. For JRules, the highly technical nature of the product will deter many organizations from adopting it. If the business users do not want to write rules themselves, or knowledge happens to be presented in the form of decision tables then you will need to translate these to rules before Haley becomes a viable option. None of the three products should be adopted without training.

This report does *not* conclude that any one product is better than any other. It merely sets out the relative strengths and weakness of the three products and considers their suitability for different types of application. As a consultant, I will be able happily to recommend all three products in the future - if they match the business and technical problems to be addressed and are to be deployed in an appropriate organizational culture.

8 APPENDICES

8.1 Business rules management system technology and terminology

This appendix presents the key scientific ideas and terms needed to understand the remainder of this report. We cover several techniques for representing and discovering knowledge and the main techniques for reasoning with it.

The knowledge representation techniques include rules, semantic networks, object models, decision tables and decision trees. We also explain the basic forms of inference used in BRMSs, including techniques of rule induction and data mining.

8.1.1 Rules and other forms of knowledge representation

Most of us are familiar with the notion of data; that is, unstructured sets of numbers, facts and symbols. These data can convey information only in virtue of some structure or decoding mechanism. In the limiting case, this distinction can be illustrated by two people who may communicate via a channel that may only carry one message consisting of a single symbol. The datum, the symbol itself, carries no information except in virtue of the presence of the channel, whose structure determines that the receiver may learn from the absence of a symbol as well from its transmission. This structure is, in turn, determined by the shared knowledge of the sender and receiver. Two points emerge from this example. Information always has a context while data may be context free; thus if I say 'she shot up' that is a datum for which I would need to explain whether the person in question was an astronaut or a heroin addict to convey unambiguous information. Knowledge is usually seen as a concept at a higher level of abstraction, and there is a sense in which this is true. For example, '1000' is a datum, '1000 millibars at noon' could be information about the weather in some situations but 'Most people feel better when the pressure rises above 1000 millibars' is knowledge about barometric information and people. The realization that much knowledge is expressed in the form of heuristic descriptions or rules of thumb is what gives rise to the conception of knowledge as more abstract than information.

Apart from asking what it is, epistemologists have traditionally raised several other problems concerning knowledge, including:

- # How it may be classified;
- # How it is obtained;
- # Whether it has objective reality;
- # If it is limited in principle.

As a preliminary attempt at classification we might note that there are several evidently different types of knowledge at hand; knowledge about objects, events, task performance, and even about knowledge itself. If we know something about objects such as tomatoes we will probably know that tomatoes are red. However, we are still prepared to recognise a green tomato as a tomato; so that contradictions often coexist within our knowledge. Object knowledge is often expressed in the form of assertions, although this is by no means the only available formalism and OO-style objects or frames are particularly well suited to this purpose. Here are a few typical assertions:

1. Tomatoes are red
2. Zoë is very lively
3. This house is built with bricks and mortar

Knowledge of causality, however, is expressed typically as a chain of statements relating cause to effect. A typical such statement might be 'If you boil tomatoes with the right accompaniments, chutney results.'

Such knowledge is well represented by sets of rules that can be chained together or by logical propositions within a particular logical calculus.

To perform a task as commonplace as walking requires a very complex interacting system of knowledge about balance, muscle tone, etc.; much of which is held subconsciously and is deeply integrated with our biological hardware. Knowledge about cognition, often called meta-knowledge, also needs to be represented when such questions as 'What do I know?' and 'How useful or complete is a particular knowledge system or inference strategy?' are raised. This, I hope, shows that there is no clear boundary between knowledge and inference, as practices. Each interpenetrates the other; we have inference with knowledge and knowledge about inference.

There are various dimensions along which knowledge can be evaluated:

Scope   – What does it cover?
Granularity   – How detailed is it?
Uncertainty   – How likely, certain or plausible is it?
Completeness   – Might we have to retract conclusions if new knowledge comes to light?
Consistency   – How easily can we live with its contradictions?
Modality   – Can we avoid its consequences?

The above dimensions are all connected with some form of uncertainty. This arises from the contradictory nature of knowledge. Knowledge presents itself in two basic forms as absolute and relative. To understand this, consider the whole of the history of science, which is an attempt to arrive at a knowledge of the environment we inhabit and change our relationship with it. The scientist develops various theories that explain the experimental evidence and are further verified in practice. S/he never suspects that any theory is comprehensively correct, at least not nowadays. Newton's models overthrew the theories of earlier times and were in their turn overthrown by Einstein's. If nature exists beyond, before and apart from us then it represents, in all its complexity, an absolute truth which is (in principle) beyond knowledge because nature is not in itself human and knowledge is. To assume otherwise is to assert that nature is either a totally human construct or that the whole may be totally assimilated by a fragment of itself. This is not to say that the finite may not know the infinite, only that the knowledge may only be relative. Otherwise the finite would contain the infinite and thus become infinite itself. Thus all truth seeking aims at the absolute but achieves the relative and here it is that we see why all knowledge must perforce be uncertain. This is why the correct handling of uncertainty is one of the primary concerns for builders of knowledge-based systems of any sort.

The dimensions of knowledge mentioned above all will have some bearing on the techniques used to represent knowledge. If we choose logic as the representation then, if our knowledge is incomplete, non-monotonic logic will be required in preference to first order predicate logic and, in the presence of uncertainty, a logic capable of handling it will be required. Similar remarks apply to inconsistent knowledge where contradiction must be handled either by the logic or the control structure or metalogic. Modality will require the use of a logic that can deal with necessity and possibility.

If, on the other hand, we choose objects, frames or semantic network representations, the scope and granularity will affect the amount of storage we can expect to use. For this, it is useful to have some metrics. Granularity is often measured in **chunks**. Anderson (1976) defines a chunk to be a learnt configuration of symbols which comes to act as a single symbol. The passage between levels of representation is an important theme in AI research and has great bearing on the practical question of efficiency of storage and execution. Generally speaking, you should choose a granularity close to that adopted by human experts, if this can be discerned, and use chunking whenever gains are not made at the expense of understandability.

8.1.1.1 Rules and production systems

The concept of reducing systems to a few primitives and production rules for generating the rest of the system goes back to Post, who with Church and Turing all worked on the idea of formal models of computers independently. Post's original work was concerned with the theory of semigroups, which is of interest in algebraic models of language. Newell and Simon (1963) introduced them in the form in which we find them in knowledge based systems as part of their work on GPS, the general problem solver, which was an attempt to built an intelligent system which did not rely for its problem solving abilities on a store of domain specific knowledge but would *inter alia* generate production rules as required. For example, Marvin the robot wants to go to Boston. He is faced with an immediate problem before this goal can be satisfied: how to get there. He can fly, walk/swim, ride a bus or train, and so on. To make the decision he might weigh up the cost and the journey time and decide to fly, but this strategy will not work because he is not at an airport. Thus he must solve a subproblem of how to get to an airfield which runs a service that takes him close to Boston. In production rules his reasoning so far (he hasn't solved the whole problem yet) might look like this:

1. If I want to go to new York then I must choose a transport mode
2. Flying is a mode of transport which I will choose
3. If you are at an airport then you can fly
4. I am not at an airport
5. If I want to be at an airport then I must choose a transport mode

Incidentally, we should note the distinction between Marvin's goal, being in Boston, and his tasks, the steps to be taken to get there. All reasoning of this nature can be equally well viewed as goal decomposition or task decomposition. It can easily be expressed in a UML use case model.

The five statements above consist of assertions and productions and together these represent some of the knowledge Marvin needs to begin reasoning about his problems. There are many reasoning strategies or inference methods he can employ. For the time being, we are interested in the representation of knowledge by production rules, as these IF/THEN constructions are known.

The left hand side, A, of a production rule of the form 'If A then X' is called its **antecedent** clause and the right hand side, X, its **consequent**. It may be interpreted in many ways: if a certain condition is satisfied then a certain action is appropriate; if a certain statement is true then another can be inferred; if a certain syntactic structure is present then some other can be generated grammatically. In general the A and X can be complex statements constructed from simpler ones using the connectives AND and OR and the NOT operator. In practice only A is permitted this rich structure so that a typical production would look like this:

> IF (animal bears live young AND animal suckles young)
> OR location is mammal-house
> THEN animal is mammal

The parentheses disambiguate the precedence of the connectives and avoid the need to repeat clauses unnecessarily. Production systems combine rules as if there were an OR between the rules; that is between the antecedents of rules with the same consequent. A production rule system may be regarded as a machine which takes as input values of the variables mentioned in antecedent clauses and puts out values for the consequent variables. Clearly, it is equivalent to a system with one machine for each consequent variable unless we allow feedback among the variables. When feedback is present we enter the realms of inference.

Production rules are easy for humans to understand and, since each rule represents a small independent granule of knowledge, can be easily added or subtracted from a knowledge base. For this reason they have formed the basis of several well known, large scale applications such as DENDRAL, MYCIN and PROSPECTOR. They form the basis of nearly all BRMS products. Because the rules are, in principle, independent from each other, they support a declarative style of programming which considerably reduces maintenance problems. However, care must be taken

that contradictory rules are not introduced since this can lead to inefficiency at best and incorrect conclusions at worst. Another advantage that has been exploited in rule-based systems is the ease with which a production system can stack up a record of a program's use of each rule and thus provide rudimentary explanations of the systems reasoning. Lastly, productions make fairly light demands on a processor, although large amount of memory or secondary storage will typically be required.

Precisely because they are memory intensive, production systems can be very inefficient. Also it is difficult to model associations among objects or processes. This makes the taking of short cuts in reasoning difficult to implement. The declarative style makes algorithms extremely difficult to represent, and flow of control is hard to supervise for a system designer. Lastly, the formalism – as described so far – makes no allowances for uncertain knowledge. For these reasons, it is now becoming more common to find that knowledge based systems use several different kinds of knowledge representation, usually a mixture of rules, objects and procedures. The formalism can be directly extended to cope with some kinds of uncertainty.

8.1.1.2 Knowledge and inference

The question of how human beings store and manipulate knowledge is a question we can only touch upon here. The questions of how knowledge comes about and how it may be substantiated, what philosophers call the problem of cognition, or epistemology, is sufficiently neglected in the existing literature of knowledge engineering to deserve a little attention though. We also ask how the interconnexion between knowledge and inference is mediated. In my view, it is this relationship that leads to the need for uncertainty management in expert systems.

Consider two important questions about the representation of knowledge. First there was the question of how knowledge is represented in the human or animal brain, and now there is that of what structures may be used for computer representation. The first question is the concern of cognitive psychology and psychoanalysis and will not exercise us greatly here. However, the theories of psychologists and psychoanalysts have much to offer in the way of ideas for object discovery techniques. The inter-disciplinary subject of artificial intelligence has been defined as 'the study of mental faculties through the use of computational models' (Charniak & McDermott, 1985); exactly the reverse of what interests builders of knowledge based systems. Perhaps this is why there is such confusion between the fields today. One important point to make categorically is that no one knows how the human brain works and no one could give a prescription for the best computer knowledge representation formalism even if they did. Until some pretty fundamental advances are made, the best bet for system builders is to use whatever formalism best suits the task at hand, pragmatically.

Apart from its ability to be abstract at various levels, knowledge is concerned with action. It is concerned with practice in the world. Knowing how people feel under different atmospheric conditions helps us to respond better to their moods, work with them or even improve their air-conditioning (if we have some knowledge about ventilation engineering as well). Incidentally, it also assumes the existence of various socially evolved measuring devices, such as the barometer, thermometer and so on. Knowledge is a guide to informed practice and relates to information as a processor of it; that is, we **understand** knowledge but we **process** information. It is no use knowing that people respond well to high pressure if you cannot measure that pressure. Effective use of knowledge leads to the formation of plans for action and ultimately to deeper understanding. This leads to a subsidiary definition that knowledge is concerned with using information effectively. The next level of abstraction might be called 'theory'.

From this point of view, inference is to knowledge as processing is to information. Inference is the method used to transform perceptions (perhaps via some symbolic representation) into a form suitable for re-conversion into actions. It may also be viewed as an abstraction from practical activity. In our experience of the world we observe, both individually and collectively, that certain consequences follow from certain actions. We give this phenomenon the name causality, and say

that action A 'causes' perception B. Later (both in ontogenesis and philogenesis[2]) we generalize this to include causal relations between external events independent of ourselves. From there it is a short step (one originally taken at the end of the Bronze Age) to the idea that ideas are related in a similar way; that symbol A can 'imply' symbol B. This process of abstraction corresponds, according to Piaget, to the process of child development. Historically, it corresponds to the development of the division of labour. In other words just as tool making and social behaviour make knowledge possible, so the interdependencies of the world of nature are developed into the abstract relations of human thinking; part of this system of relationships corresponding to inference.

Of course, computers do not partake of social activity, nor yet do they create tools (although they may manufacture and use them if we include robots in our perception of computing machinery). As far as inference is concerned we cannot expect computers to encompass the richness and depth of human reasoning (at least not in the foreseeable future). For many thousands of years it has been convenient, for certain applications in the special sciences, to reason with a formalized subset of human reasoning. This 'formal logic' has been the basis of most western technological developments and, while not capturing the scope of human informal reasoning, is immensely powerful in resolving many practical problems. Thus, we are converging on a definition of inference which will serve the purposes of knowledge engineering. Inference in this sense is the abstract, formal process by which conclusions may be drawn from premises. It is a special kind of metaknowledge about the abstract relationships between symbols representing knowledge.

Many philosophers have questioned whether true artificial intelligence is possible in principle. In my view the question is merely maladroit. Clearly, if we are able in future to genetically (or otherwise) engineer an artificial human being there is no reason (excluding spurious religious arguments) why the constructed entity should not be 'intelligent' by any normal criteria. If, on the other hand, the question is posed as to whether electronic computers of the type currently existing or foreseeable can pass the Turing test, then matters are a little different. Human cognition is a process mediated by both society and the artefacts of Man's construction. It may well be that no entity (be it a computer or a totally dissimilar organism from outer space) could ever dissemble its true non-social, non-tool making character sufficiently to deceive the testers. My belief is that artificial intelligence in this sense is impossible but that useful results are to be obtained by trying to achieve an approximation.

8.1.1.3 *Semantic networks*

A semantic network consists of a set of nodes and a set of ordered pairs of nodes called 'links', together with an interpretation of the meaning of these. Terminal links are called 'slots' if they represent properties (predicates) rather than objects or classes of objects. A frame is a semantic net representing an object (or a stereotype of that object) and will consist of a number of slots and a number of outbound links. Frames correspond to classes and instances in object modelling.

Semantic networks and object models are both used to represent knowledge about objects and the static relationships among them rather than knowledge about the dynamic relationships expressed by rules. Object models can express knowledge such as 'all healthy dogs have two eyes'. Rules can express knowledge such as 'if a dog starts to bark then an intruder may be present'. These are two quite different kinds of knowledge. *Both of them are essential* in dealing with any problem.

Semantic nets generalize object models. Classes and their instances are represented uniformly in the former whereas, in object-oriented programming, there is a profound distinction. In both representations, we have associations (links) between classes. But in object-oriented programming

---

[2]Ontogenesis is the origin or developmental history of the being (the individual in this sense) and philogenesis the origin of the species. I deliberately choose these terms to remind the reader of the ancient and famous Greek aphorism: "Philogenesis recapitulates ontogenesis".

inheritance between classes and classes and between classes and instances is treated differently. We can say that a Dog is *a kind of* Mammal but, when we encounter Fido, we have to say that Fido *is a* Dog. Furthermore, once created, Fido is a dog forever. He can't ever migrate to the class of GuardDogs during his lifetime; he's destined to be just plain old Dog till he dies. Obviously, this is not the way most people think or express themselves in natural language. Because of this limitation, it is tricky to model rôles, such as guard dog or retired person. In fact, patterns are used to get round the problem (STATE and VISITOR usually).

The semantic network approach corresponds far more closely to common sense. In HaleyAuthority, using an example supplied by Haley Systems, we can say 'an applicant provides an answer to a health question'. Health questions are a kind of question. Questions have instances like 'What is your name?' whilst health questions have instances like 'Do you smoke?' or 'Question 17'. We can also, in the same underwriting application, talk readily about dangerous occupations specifying, perhaps, 'Iraq security consultant' as an instance.

Semantic networks are thus far more expressive. Nevertheless they can be readily (and automatically) mapped onto the Java object model.

Let us now descend from these abstract considerations and ask how computers can be made to simulate reasoning.

8.1.2 Inference in business rules management systems

Given that knowledge is stored in a computer in some convenient representation or representations, the system will require facilities for navigating through and manipulating the knowledge if anything is to be achieved at all. Inference in the usual logical sense is this process of drawing valid conclusions from premises. In our wider sense it is any computational mechanism whereby stored knowledge can be applied to data and information structures to arrive at conclusions which are to be plausible rather than valid in the strict logical sense. This, of course, poses problems in relation to how to judge whether the conclusions are reasonable, and how to represent knowledge about how to test conclusions and how to evaluate plausibility. Thus, we can see that knowledge representation and inference are inextricably bound together, though as opposites.

8.1.2.1 Forward, backward and mixed chaining strategies

Up to now we have only considered the problem of how to infer the truth value of one proposition from another using a rule of inference in just one step. Clearly however, there will be occasions when such inferences (or proofs) will involve long chains of reasoning using the rules of inference and some initial suppositions (or axioms). We now turn to the inference methods that feature strongly in all rule-based systems and are often supplied as standard in BRMS products.

Forward chaining

To fix ideas, consider a system whose knowledge is represented in the form of production rules and whose domain is the truth of abstract propositions: A, B, C, ...

The knowledge base consists solely of rules as follows.

Rule 1: A and B and C	implies D
Rule 2: D and F	implies G
Rule 3: E	implies F
Rule 4: F	implies B
Rule 5: B	implies C
Rule 6: G	implies H
Rule 7: I	implies J
Rule 8: A and F	implies H

To start with, assume that the system has been asked whether proposition H is true given that propositions A and F are true. We will show that the system may approach the problem in two quite distinct ways. Assume for the present that the computer stores these rules on a sequential device such as magnetic tape, so that it must access the rules in order unless it rewinds to rule 1.

What I am about to describe is a basic forward chaining inference strategy. This itself has several variants. We may pass through the rules until a single rule fires, we may continue until all rules have been processed once, or we may continue firing in either manner until either the conclusion we desire has been achieved or until the database of proven propositions ceases to be changed by the process. A little thought shows that this gives at least four different varieties of forward chaining. This will become clearer as we proceed.

The assumption is that A and F are known to be true at the outset. If we apply all the rules to this database the only rules that fire are 4 and 8 and the firing of rule 8 assigns the value true to H, which is what we were after. Suppose now that rule 8 is excised from the knowledge base. Can we still prove H? This time only rule 4 fires, so we have to rewind and apply the rules again to have any chance of proving the target proposition. Below we show what happens to the truth values in the database on successive applications of the rules 1 to 7.

Proposition	Iteration number							
	0	1	2	3	4	5	6	7
A	T	T	T	T	T	T	T	T
B		T	T	T	T	T	T	T
C			T	T	T	T	T	T
D				T	T	T	T	T
E								
F	T	T	T	T	T	T	T	T
G						T	T	T
H						T	T	T
I								
J								

Table 8.1 Naïve forward chaining.

So, H is proven after 5 iterations. Note, in passing, that further iterations do not succeed in proving any further propositions in this particular case. Since we are considering a computer strategy, we need to program some means by which the machine is to know when to stop applying rules. From the above example there are two methods; either 'stop when H becomes true' or 'stop when the database ceases to change on rule application'. Which one of these two we select depends on the system's purpose; for one interesting side effect of the latter procedure is that we have proved the propositions B, C, D and G and, were we later to need to know their truth values, we need do no more computation. On the other hand, if this is not an important consideration we might have proved H long before we can prove everything else.

It should be noted that we have assumed that the rules are applied 'in parallel', which is to say that in any one iteration every rule fires on the basis that the data are as they were at the beginning of the cycle. This is not necessary, but we would warn of the confusion that would result from the alternative in any practical applications; a knowledge-based, and thus essentially declarative system, should not be dependent of the order in which the rules are entered, stored or processed unless there is some very good reason for forcing modularity on the rules. Very efficient algorithms, notably the rete algorithm, have been developed for this type of reasoning.

These strategies are known as **forward chaining** or data directed reasoning, because they begin with the data known and apply the rules successively to find out what results are implied. This strategy is particularly appropriate in situations where data are expensive to collect but

potentially few in quantity. Typical domains are loan approval, financial planning, process control, scheduling, the configuration of complex systems and system tuning.

In the example given, the antecedents and consequents of the rules are all of the same type: propositions in some logical system. However, this need not be the case. For example, the industrial control applications the inputs might be measurements and the output control actions. In that case it does not make sense to add these incommensurables together in the database. Variations on forward chaining now include: 'pass through the rules until a single rule fires then act'; 'pass through all the rules once and then act'.

Backward chaining

There is a completely different way we could have set about proving H, and that is to start with the desired goal 'H is true' and attempt to find evidence for this to be the case. This is **backward chaining** or goal directed inference. It is usual when the only thing we need to do is prove H and are not interested in the values of other propositions.

Backwards chaining arises typically in situations where the quantity of data is potentially very large and where some specific characteristic of the system under consideration is of interest. Most typical are various problems of diagnosis, such as medical diagnosis or fault finding in electrical or mechanical equipment. Most first generation expert system shells were based on some form of backward chaining, although some early production rule languages such as OPS5 used forward chaining.

Returning to our original 8 rules, the system is asked to find a rule that proves H. The only candidate rules are 6 and 8, but 6 is encountered first. Let us ignore rule 8 for the present. At this point we establish a new subgoal of proving that G is true, for if we can do this then it would follow that H were true by *modus ponens*. Our next subgoal will be to prove that D and F are true. Recall that we have told the system that A and F are true, so it is only necessary to prove D (by $\land$ introduction). The whole proof proceeds as shown in Figure 8.1

Trying to prove H
Try rule 6
Trying to prove G
Try rule 2
F is true, trying to prove D
Try rule 1
A is true, trying to prove B
Try rule 4
It works. B is true
Backtrack to trying rule 1
Trying to prove C
Try rule 5, it works C is true
Apply rule 1, D is true
Apply rule 2, G is true
Apply rule 6, H is true

Figure 8.1 Proof by backward chaining or recursive descent.

The observant reader will have noticed that we could have proved H in one step from rule 8. The point is that rule 8 was not reached and the system could not know in advance that it was going to be quicker to explore that rule than rule 6. On the other hand if the original line of exploration had failed (suppose rule 4 was deleted) then the system would have had to backtrack and try rule 8. Figure 8.2 illustrates the proof strategy more pictorially.

Backward chaining can thus be viewed as a strategy for searching through trees built in some solution space. The strategy we have described is usually called depth-first search in that context. We now look at other strategies.
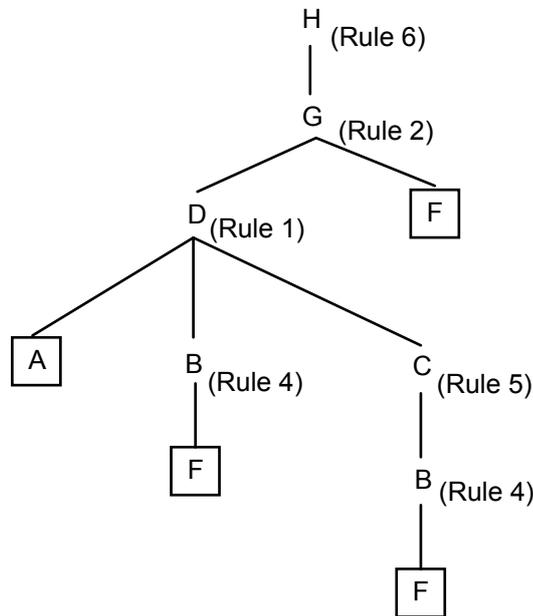
Figure 8.1 A proof tree. Propositions in boxes are those found in
the database (i.e. those known to be true).

Mixed strategies

We have looked at two fundamental forms of inference, forward and backward chaining. In practice, most reasoning is a mixture of at least these two. Given some initial assumption, we infer a conclusion by reasoning forwards and then apply backward chaining to find other data that confirm these conclusions. Alternatively, we start with a goal, backward chain to some plausible reason and then forward chain to exploit the consequences of this new datum. This is often called **opportunistic chaining** or, less succinctly, 'backwards reasoning with opportunistic forward chaining', because the data directed search exploits the consequences of data as they become available 'opportunistically'. This method is commonly found in the better BRMS products. Another way of looking at it is to observe that every rule becomes a demon.

Rete

There have been several attempts to construct computer languages specifically for knowledge representation. The best known, early languages were probably KRL (Bobrow and Winograd, 1977) and OPS5 (Forgy, 1982). The basic form of representation in OPS5 is production rules. OPS5 first achieved notoriety because it was used in the highly successful XCON system, which was used by Digital Equipment Corporation (DEC) to configure orders for VAX™ computers. The fact that a large chunk of XCON, concerned with database access, was written in the procedural language Bliss32 is rarely mentioned, but that does not change the fact that the knowledge incorporated in the system is the key to its success. DEC's success rate in the configuration task increased by a factor of more than two, resulting in huge savings. Even more important is that XCON enabled DEC to maintain its distinctive policy of delivering just what the

customer asks for, however non-standard. The maintenance of the OPS5 rulebase was in fact a vastly costly operation, because of the continual updates in the product range.

Rete is a very efficient mechanism for solving the difficult many-to-many matching problem in artificial intelligence. Rete is an algorithm that evaluates a declarative predicate against a changing set of set in real time. Consider an SQL select statement that executes a WHERE clause to find matching rows. Rete uses a progressive relational join to update a view of matching rows. As rows are added to any table, it's evaluated against the predicate and mapped into or out of the matching view.

Rete is much more efficient at determining the relevance of rules, given particular data, than the equivalent nested if/then/else or select/case constructs. The greater the number of rules, the greater rete's advantage over procedural code. This applies to rule execution. Of course, writing the rules is also far more efficient in a BRMS.

When a rulebase becomes large, the naïve algorithm for forward chaining illustrated in Table 1 can become very slow because few changes are made to the facts in working memory at each cycle. Rete compiles the rules into a network of predicate tests, inferences and actions. The rete network modifies itself after each rule firing, so that unneeded rules do not fire. For further details of rete see, for example, Russell and Norvig (1995). Each product reviewed here has a proprietary improvement on the basic published algorithm. These improvements are largely responsible for the variation in performance of the three rule engines. The three engines all also modify basic rete to permit backward and mixed chaining.

8.1.2.2 Data mining and rule induction

The other principal mode of inference is **induction**. Broadly, induction enables us to infer new rules from collections of facts and data. The word 'induction' has two senses: the Aristotelian sense of a syllogism in which the major premise in conjunction with instances entails the generalization, or the sense of empirical generalization from observations. A third sense, the principle of mathematical induction, need not concern us here. It is with the second sense we shall be concerned. Most authorities talk about induction in terms of probabilities; if we observe that sheep on two hundred hillsides all have wool and four legs, then we may induce the generalization 'all sheep have wool and four legs'. Every observation we then make increases the probability of this statement being true, but never confirms it completely. Only one observation of a shorn, three-legged merino is needed to refute the theory. From our point of view, this cannot be correct. There are many kinds of uncertainty, and it can be said equally that our degree of knowledge, belief or the relevance of the rules is what is changed by experience rather than probability. The obsession with probability derives (probably) from the prevailing empiricist climate in the philosophy of science; experience being seen only as experiments performed by external observers trying to refute some hypothesis. Another view is possible. The history of quantum physics shows that we can no longer regard observers as independent from what they observe. Experience takes place in a world of which we humans are an internal part but from which we are able to differentiate ourselves. We do this by internalizing a representation of nature and checking the validity of the representation through continuous practice. However, the very internalization process is a practice, and practice is guided by the representation so far achieved. From this point of view induction is the process of practice that confirms our existing theories of all kinds. The other important general point to note is that the syllogism of induction moves from the particular to the general, whereas deductive and abductive syllogisms tend to work in the opposite direction; from the general to the particular.

The probabilistic definition of induction does have merit in many cases, especially in the case of new knowledge. It is this case that current computer learning systems always face. In nearly every case, computer programs which reason by induction are presented with a number of examples and expected to find a pattern, generalization or program that can reproduce and extend the training set.

Suppose we are given the training set of examples Shown in Table 8.1.

The simplest possible algorithm enables us to infer that:

```
IF female THEN analyst
IF male AND (blue eyes OR grey eyes) THEN programmer
IF brown hair AND brown eyes THEN operator
```

However, the addition of a new example (brown eyes, brown hair, female, programmer) makes the position less clear. The first and last rules must be withdrawn, but the second can remain although it no longer has quite the same force.

Name	Eye colour	Hair colour	Sex	Job
J. Stalin	blue	blonde	male	programmer
A. Capone	grey	brown	male	programmer
M. Thatcher	brown	black	female	analyst
R. Kray	brown	brown	male	operator
E. Braune	blue	black	female	analyst

Table 8.2 Training set

The first attempts at machine learning came out of the cybernetics movement of the 1950s. Cybernetics, according to its founder Wiener (1948), is the science of control and communication in animal and machine. Several attempts were made, using primitive technology by today's standards, to build machinery simulating aspects of animal behaviour. In particular, analogue machines called homeostats simulated the ability to remain in unstable equilibrium; see Ashby (1956). Perceptrons are hinted at in Wiener's earliest work on neural networks, and, as the name suggests, were attempts to simulate the functionality of the visual cortex. Learning came in because of the need to classify and recognise physical objects. The technique employed was to weight the input in each of a number of dimensions and, if the resultant vector exceeded a certain threshold, to class the input as a positive example. Neural network technology has now overcome an apparent flaw discovered by Minsky and Papert (1969), and impressive learning systems have been built.

Rule based learning systems also exist. Quinlan's interactive dichotomizer algorithm, known as ID3, selects an arbitrary subset of the training set and partitions it according to the variable with the greatest discriminatory power using an information theoretic measure of the latter. This is repeated until a rule is found which is added to the rule set as in the above example on jobs. Next the entire training set is searched for exceptions to the new rule and if any are found they are inserted in the sample and the process repeated. The difficulties with this approach are that the end result is a sometimes huge decision tree which is difficult to understand and modify, and that the algorithm does not do very well in the presence of noisy data, though suitable modifications have been proposed based on statistical tests.

One of the problems with totally deterministic algorithms like ID3 is that, although they are guaranteed to find a rule to explain the data in the training set, if one exists, they cannot deal with situations where the rules can only be expressed subject to uncertainty. In complex situations, such as weather forecasting or betting - where only some of the contributory variables can be measured and modelled - often no exact, dichotomizing rules exist. With the simple problem of forecasting whether it will rain tomorrow it is well known that a reasonably successful rule is 'if it is raining today then it will rain tomorrow'. This is not always true but it is a reasonable approximation for some purposes. ID3 would reject this as a rule if it found one single counter-example. Statistical tests, however useful, require complex independence assumptions and interpretative skills on the part of users.

A completely different class of learning algorithm is based on the concept of adaptation or Darwinian selection. The general idea is to generate rules at random and compute some measure of performance for each rule relative to the training set. Inefficient rules are deleted and operations

based on the ideas of mutation, crossover and inversion are applied to generate new rules. These techniques are referred to as **genetic algorithms**.

Genetic algorithms are also closely related to neural nets as pattern classification devices. Genetic programming is a form of machine learning that takes a random collection of computer programs and a representation of some problem and then 'evolves' a program that solves the problem. It does this by representing each program as a binary vector, or string, that can be thought of as a chromosome. The chromosomes in each successive sample can 'mate' by crossing over portions of themselves, inverting substrings of their bodies and mutating at random[3]. Programs that score well against some objective function that represents the problem to be solved are allowed to participate in the next mating round and, after many generations, there is a good chance that a successful - but not necessarily optimal - program will evolve.

None of the products considered herein offer any sort of rule induction facility. However, there are several products on the market that do and we envisage some benefit from taking the output from such systems and offering the resultant rules to a BRMS.

8.1.3 Techniques for representing rules

In all BRMS products, rules are represented as sentences, usually containing the words IF and THEN. Morgan (2002) recommends a better style aimed at removing ambiguity, making relationships explicit, avoiding obscure terminology, removing wordiness, and so on. His style is remarkably close to natural language. He ends up preferring forms such as

```
A loan may be approved
     if the status of the customer is high and the loan is less
     than 2000
     unless the customer has a low rating
```

to

```
if the customer status is high and the loan is less than 2000
     and the customer does not have a low rating
     then approve the loan
if the customer status is high and the loan is less than 2000
     and the customer has a low rating
     then don't approve the loan
```

In some products there are alternative representations to rules. We now consider two of these.

8.1.3.1 Decision trees and decision tables

Decision trees

Behavioural science has evolved several theories as to how people reach decisions. Such descriptive theories usually conclude by stating that managers do not make decisions on a purely rational basis. To help managers improve their decision making however, a normative theory such as decision analysis is required. Decision analysis consists of three principal stages:

1. Determine problem structure
2. Assess uncertainties and possible outcome states
3. Determine a 'best' strategy for achieving a desirable outcome

A decision problem is characterised as one of selecting one from several options so as to maximise some function of possibly many variables, attributes or criteria. The naive formulation is to organize these into a table of options against attributes. Many methods are available to achieve

---

[3]Given two binary strings (representing chromosomes) 110101 and 111000, their crossover (at the fourth place) could either be 110000 or 111101. Crossing over at the first place corresponds to choosing one of the original strings.

the requisite selection: maximising, minimaxing, regret and so on. The disadvantage of this method is that complex problems are sometimes oversimplified by it, a method of overcoming this will be considered in due course. The so-called modelling school of decision analysis would attempt to construct a more explicit model of the relationships, usually as a decision tree such as the one in Figure 8.3.
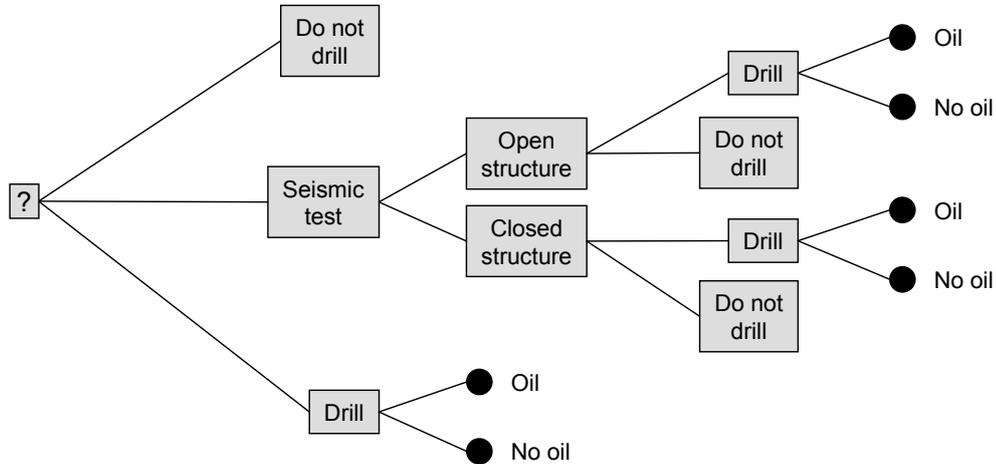
Figure 8.3 The oilman's problem

In most professions and businesses, decision making takes place in an environment where the cost of obtaining precise information is unjustifiably high. In recognition of this fact the classical theories of decision analysis, operational research and decision theory make extensive use of normative statistical techniques. The decision problem is either a question of choosing an optimal course of action, such as the ideal mix of ingredients in animal foodstuffs, subject to constraints such as lowest cost and some requisite nutritional value, or it is concerned with generating a plausible set of alternatives. It is the first case, which has received most attention. A decision problem, in this latter sense, is given by stating a set of options, a set of states, a transformation which to every pair consisting of a state and an option returns a new state representing the consequence of choosing that option, *ceteris paribus*. Since the null option (do nothing) is always included, this provides a model of the evolution of the system to which may be added feedback and/or feedforward control of options. Thus, we see that cybernetics becomes a special case of decision theory, and indeed many of the mathematical techniques are held in common. In addition, decision models include a utility function, which represents the ranking of outcomes with regard to their desirability in a given context. This function is analogous to the metrics required for homeostasis in cybernetic systems. In the cases where decisions can be made in the presence of certain data, the techniques of operational research, such as linear and dynamic programming and systems dynamics, are the most commonly used. This leaves us with essentially only one tool: the decision tree. A decision tree is merely a hierarchy showing the dependencies between decisions. It is a shorthand description of some aspects of the general decision model whose chief value is to clarify our thinking about the consequences of certain decisions being made. However, with the introduction of probabilities the decision tree becomes a powerful tool.

To see this, consider a very simple example. If one wishes to open a sweet shop, one must decide where it is to be located. There are, let us suppose, three options: near a school in an expensive suburb, in the busy high street or opposite a playground in a deprived inner city area. Let us call these options A, B and C. To each of these we can assign a probability of financial success, based on basic cost/revenue calculations and the history of similar ventures. In each case, however, there are other decisions to make, such as how much to invest in stock. Suppose the

options and probabilities of success are as displayed in Figure 8.4, where X, Y and Z represent these other decisions.

Combining the probabilities shows that option C is the most likely to succeed, despite the fact that on the basis of the first level of decision it was the worst option. Exploring the decision tree further might change the position again. Enhancements of this application of probability theory have proved most effective in attacking a wide range of decision problems. It is also possible to use certainty factors in place of probabilities, in which case the arithmetic is different.

In many cases the branches of a tree will be annotated with and followed when particular ranges of values hold for a variable. For example we might set a particular credit limit for a client with annual income in the range 50,000 to 100,000.
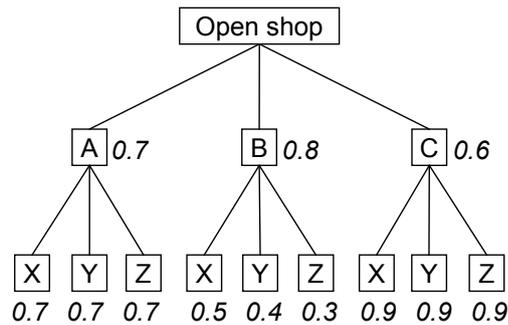
Figure 8.4 A decision tree with probabilities.

Decision tables

Decision tables represent the same knowledge and rules as decision trees in a tabular format. For example, Table 8.2 is equivalent to a ruleset stating:

```
If card type is "Standard"
     then discount code is 1
     unless age is between 18 and 30
If card type is "Standard" and age is between 18 and 30
     then discount code is unknown
If card type is "Gold"
     then discount code is 2
     unless age is between 31 and 40
If card type is "Gold" and age is between 31 and 40
     then discount code is 1
If card type is "Platinum"
     then discount code is 3
     unless age is between 31 and 40
If card type is "Platinum" and age is between 31 and 40
     then discount code is 1
```

We can see that the techniques of rule induction discussed above may be applied to extract the rules from the table automatically. We can also generate a table from a ruleset.

The main problem with decision tables is that they grow unmanageably large when there is a large number of conditions in the rulebase. Even in the example above, where this is not the case, six rules translate to 72 table entries. Their advantage arises when the organization already holds the knowledge in this form: pricing charts, rate tables, etc. However, this advantage largely evaporates when the rules can access the same data in the form of lookup tables.

There is a cruder approach that regards each row in the table as a separate rule; so that row two would correspond to a rule stating:

```
If card type is "Gold" and age is between 18 and 30
then discount code is 2
```

Clearly this approach gives as a larger number of rules – one for each row – and the rules will be hard to read and understand. We characterize the approach as **row-oriented decision tables**. Rule subsumption checks may allow the author to tidy up the resultant rulesets but we think the induction approach is far sounder. It is better to use a data mining system to extract rules from decision tables and feed them into a BRMS.

Min age	Max age	Card type	Discount code
18	30	Standard	
18	30	Gold	2
18	30	Platinum	3
31	40	Standard	1
31	40	Gold	1
31	40	Platinum	1
41	50	Standard	1
41	50	Gold	2
41	50	Platinum	3
51	60	Standard	1
51	60	Gold	2
51	60	Platinum	3
61	70	Standard	1
61	70	Gold	2
61	70	Platinum	3
71	120	Standard	1
71	120	Gold	2
71	120	Platinum	3

Table 8.2 A decision table

8.1.4 Ontology and Epistemology: the rôle of object modelling and natural language processing

There are, arguably, four basic branches of Philosophy: Epistemology, Ontology, Ethics and Aesthetics. All other branches of Philosophy, such as the Philosophies of Politics or Language, draw on these disciplines to some extent. All of them are relevant to system development

Epistemology concerns what we know. Some knowledge can be expressed as procedures, rules and relationships. Epistemology includes the science of method: Methodology. Ontology, sometimes called Metaphysics, concerns what exists: what are the objects in our world. Aesthetics is clearly relevant to user interface design. Ethics or Moral Philosophy tells us whether the systems we build are useful, legal or morally sound.

Business rules management system cannot be built without paying attention to rules (epistemological facts) and the objects that these rules refer to (the domain ontology).

In particular, any attempt to represent rules in natural language will fail unless there is a well-constructed model of the objects and concept in the domain, their attributes and relationships. Furthermore, there must be a link between the object model and the rule language,

All the products considered in this report require that you build such an object model. The only questions are the order in which you create the models (rules first or objects first) and the degree of integration of the models.

One approach, used in early expert systems shells, is to create the objects automatically by parsing or compiling the rules. This leads to an impoverished, flat object model that often can't distinguish object from their attributes and makes it difficult to attach methods or rulesets to objects. A better approach is to create a good packaged and layered component model separately from the rules. This is usually not a trivial task and most organizations will benefit from help in enhancing their component modelling skills. Indeed, one of TriReme's specialities is mentoring in this area and we have developed advanced methods for component based design, including Catalysis™.

If natural language processing is to be attempted we need to do two additional things. The model needs to reflect the way people think about objects and the way they construct sentences to talk about them.

The way normal people think about objects is not quite the same as the semantics of a C++ or Java object model. For example, in rule-based applications, rôles are important concepts and, in real life, instances often change rôles or adopt multiple rôles; I can be a student *and* an employee. For this reason a modelling approach based on semantic networks is more appropriate than on based on the semantics of programming languages. Such a model, however, must be translatable into code.

To capture the way people construct sentences to talk about objects is rich and varied. We have a choice: either restrict the syntax or teach the machine how to understand a wide range of phrasing. Both approaches have advantages. A well-design formal syntax can look like English and is quick and easy to type once you know it. A parser can warn you if you have violated the syntax or referred to an object that doesn't exist. However, the rules may look strange to the untutored eyes and it is impossible to just pick up rules written by business experts and just drop then into the application. On the other hand, natural language phrasings need to be made explicit by the knowledge base creator and this takes time and effort. The reward for the extra effort is that practically anyone can now add or change rules with the domain.

That last caveat is important; the system has to *know* about a particular domain. The object model and the phrasings constitute the limits of the system's knowledge. An old joke illustrates this point well. A clever artificial intelligence programmer taught his system to use metaphor, so that it understood the sentence 'Haste is needed because time flies like an arrow.' The first user he demonstrated it to typed 'A screen is needed because fruit flies like a banana,' and crashed the system.

8.2 Development methods

Adopters of BRMS technology are well advised to follow documented development methods. There are two particularly significant areas where methods are important in this context: methods for knowledge acquisition and methods for component, service and system development.

8.2.1 Knowledge acquisition

One of the hard problems in the development of rule-based systems is knowledge acquisition: the process of discovering the knowledge assets of the organization. These may be found in documentation but are often locked up in the heads of domain experts and other staff. Business analysts will need to learn a repertoire of knowledge elicitation techniques to implement BRMSs successfully.

None of the tools considered here offer specific knowledge acquisition facilities but the natural language facilities on the products will help to widen the knowledge acquisition bottleneck. Haley's extremely strong natural language features could make it almost disappear in terms of rule authoring. However, in each case, business analysts will still have to mine the object knowledge to create the object model.

Organizations adopting such products will almost certainly need training and – even better – mentoring on knowledge acquisition techniques. Proprietary and published knowledge acquisition methods, such as KADS (Gardner *et al*., 1998), may be used.

Haley Systems publishes a knowledge acquisition method specific to HaleyAuthority. It includes advice and procedures for breaking the rules up into modules that I think would be better represented as patterns: DO THE ANALYSIS RULES BEFORE THE ACTION RULES, DO THE VALIDATION RULES EARLY ON, DISTINGUISH POSSIBLE FROM FINAL ACTIONS. The company recommends the following procedures.

1. Identify output decisions.
1. Create analysis statements that make recommendations in a medium priority module.
2. Create statements that lead to actions in a low priority module.
3. Create a module to handle exceptions in a high priority module.
4. Write applicability conditions for each module, showing the circumstances under which each conclusion is true.
5. Ensure that the conditions match the consequences of other rules where appropriate.

This method leads to a natural order of questioning domain experts. It could be used in conjunction with almost any BRMS.

8.2.2 System development

All three products are compatible with published system development methods and we think that they are more suitable to agile methods. Within such methods, a microprocess for component based development is beneficial and we recommend Catalysis™ for this purpose (D'Souza and Wills, 1999). Any adopted method must also include a knowledge acquisition component.

Strong involvement by business users militates against developer-centric methods such as RUP or XP. However, taking good ideas and fragments from these methods may well be appropriate. For example, we like the XP-like idea of writing test cases for every ruleset and using the tests to control system evolution. Also, the choice of product will affect the method selected. The method appropriate for a business rules management system like Blaze Advisor or JRules proceeds as follows.

1. Knowledge acquisition
1. Translation of policy into rule format
2. Programming the application
3. Validation

4. Programming the environment (glue code)
5. Production

HaleyAuthority eliminates some of these steps using automatic .NET or Java code generation from the rules, as follows.

1. Knowledge acquisition
1. Validation
2. Programming the environment (glue code)
3. Production

Once again we see the strengths of HaleyAuthority in non-developer-centric cultures.

The repository must record the rule authors and maintain permissions. Haley's published method recommends assigning a knowledge base administrator and setting up review and authorization procedures involving authors.

8.3 Study method

The documentation for both products available from the public internet sites was downloaded, printed and studied. Then the free evaluation versions of the three products were each downloaded. Each product was evaluated against the list of criteria listed in Section 7.2. Both the examples supplied by the vendors and our own sample insurance application were executed. Timing comparisons were performed where possible. All suppliers were contacted by telephone and email to make a judgment about technical support and to understand their unique sales propositions better.

ILOG offered the shortest time for evaluation (15 days) which may not have benefited their case, bearing in mind that – like most evaluation work – this could not be a full-time project due to other, commercial commitments. Haley offered a month-long trial and Fair Isaac a generous 45 days.

I set a notional two-day limit of the time available from downloading each trial and producing a working application (the life assurance example). Only JRules forced me to stretch this arbitrary time limit. The two days included the time needed to learn how to use each product. I had no training in any of them prior to beginning this exercise.

We used Windows 98 SE running on a relatively old Intel machine and Windows XP running on a more up to date model. For the products that required a Java virtual machine, we used version 1.4.2.

8.4 Further work

The suggested next stage of this research is to explore the topic of patterns and pattern languages for developing business rules management systems. Such patterns will be essential for organizations adopting BRMS as the field evolves and experience is gained. A good pattern language, with product-specific variants, will guide and educate less experienced developers through the construction of robust BRMS solutions.

Typical patterns might include

- #   ESTABLISH THE BUSINESS OBJECTIVES
- #   MODEL THE DOMAIN
- #   MODEL THE USER
- #   WRITE DOWN THE GOALS
- #   WRITE DOWN THE AVAILABLE FACTS
- #   SELECT THE INFERENCE STRATEGY
- #   TEST THE BOUNDARIES OF KNOWLEDGE
- #   Etc.

There is considerable scope for the integration of Haley's natural language interface with speech technology as the latter develops and matures.

8.5 About TriReme

TriReme International Ltd is the leading specialist consultancy in Europe on IT systems modelling and enterprise architecture. Founded in 1994 by Alan Cameron Wills, co-author of the Catalysis™ method of component-based design, it has since offered consulting, mentoring and training services to an impressive list of clients. [www.trireme.com]

TriReme's consultants are all experts in their respective fields with a minimum of ten years practical IT experience. For example, TriReme Director Derek Andrews is a leading expert on formal methods and component-based design. He is currently authoring a book describing Catalysis II, an evolution of Alan Wills' work. Ian Graham (see Section 3.6) is an authority on object-oriented modelling, user interface design and knowledge based systems. He has worked on business rule systems since 1982.

8.6 About the report author

Ian Graham is Chief Technology Officer and Principal Consultant with TriReme International Ltd. Ian is an industry consultant with over 20 years experience: a practitioner in IT for over 25. He is internationally recognized as an authority on business modelling, object-oriented software development methods, software development processes and rule based systems and has written extensively on these topics (e.g. Graham and Jones 1988; Graham 2001).

Before joining TriReme he spent six years in senior management positions with Chase Manhattan Bank and the Swiss Bank Corporation (now UBS). While at the latter, he was lead analyst on a $500M global re-engineering project. He created the System Development Methods for both Chase and SBC and acted as their chief specialist in object technology.

Ian began his working life as an actuary and subsequently moved to Civil Engineering where he worked on one of the largest computer models in the world at the time. The experience thus gained led him to pursue a career in IT consultancy. The benefit of many project-based experiences has made him both a thought leader in his field and a mature practitioner with a sense of the realities of IT within organizations and the human side of computing. He splits his time evenly between consulting, change management, training and development work and has advised many major corporations at a strategic level in his areas of expertise.

Ian has a significant public presence, being associated with both UK and international professional organizations in a responsible capacity, and is frequently quoted in the IT and financial press. He is well known in the UK and internationally as a public speaker and writer on advanced computing and has published over 88 articles and papers. He is the author or editor of 13 books on the subject, including one specifically on rule-based systems. He has lectured in 16 countries across 4 continents and is regularly invited to be a panellist and keynote speaker at major conferences.

Ian is a Fellow of the British Computer Society, Chartered Engineer and Chartered IT Practitioner. He has a Masters degree (with distinction) in Mathematics. He was Secretary of the British Computer Society Specialist Group on Expert Systems from 1991 to 1996 and Visiting Professor of Requirements Engineering at De Montfort University from 1998 to 2001.

9 REFERENCES

Anderson, J.R. (1976) *Language, Memory and Thought*, Laurence Erlbaum

Ashby, W.R. (1956) *An Introduction to Cybernetics*, London: Chapman & Hall

Bobrow, D. G. and Winograd, T. (1977) An Overview of KRL, A Knowledge Representation Language, *Cognitive Science* **1**(1), 3–46

Charniak, E. and McDermott, D. (1985) *Introduction to Artificial Intelligence*, Reading MA: Addison-Wesley

Cooper, A. (1999) *The Inmates are Running the Asylum*, New York: SAMS

D'Souza, D.F. and Wills, A.C. (1999) *Objects, Components and Frameworks with UML: The Catalysis Approach*, Reading MA: Addison-Wesley

Forgy, C.L. (1982), RETE: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem, *Artificial Intelligence* **19**, 17-37

Gardner, K.M., Rush, A., Crist, M.K., Konitzer,R. and Teegarden, B. (1998) Cognitive Patterns, Cambridge: University Press

Graham, I. (2001) *Object-Oriented Methods – Principles and Practice, 3rd Edition*, Harlow, England: Addison-Wesley

Graham, I. and Jones, P.L.K. (1988) *Expert Systems: Knowledge, Uncertainty and Decision*, London: Chapman & Hall

Minsky, M.L. and Papert, S. (1969) *Perceptrons*, MIT Press.

Morgan, A. (2002) *Business Rules and Information Systems*, Boston MA: Addison-Wesley

Newell, A. and Simon, H.A. (1963) GPS: A Program that Simulates Human Thought. In Feigenbaum, E.A. and Feldman, J.A. (Eds.), *Computers and Thought*, McGraw Hill

Russell, S. and Norvig, P. (1995) *Artificial Intelligence: A Modern Approach*, Englewood Cliffs NJ: Prentice Hall

Weiner, N. (1948) *Cybernetics*, Cambridge MA: MIT Press

10 TRADEMARK NOTICE

Authorete, HaleyAuthority and HaleyRules are trademarks of Haley Systems Inc.

Blaze Advisor is a trademark of Fair Isaac Inc.

Catalysis is a trademark of TriReme International Ltd. and service mark of Computer Associates Inc.

Java is a trademark of Sun Microsystems Inc.

JRules is a trademark of ILOG SA

RulesPower is a trademark of RulesPower Inc.

Windows is a trademark of Microsoft Inc.

Any other trademarks are the property of their respective owners.

11 INDEX