CHAPTER

1

Aligning IT with Business

I have not kept the square, but that to come
Shall all be done by the rule.
William Shakespeare *(Antony and Cleopatra)*

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 role in this.

Well, if you believe that, you'll believe anything. You are already thinking 'Another silver bullet!' But stay with me for at least another few paragraphs, whilst I try to convince you that it may actually be worthwhile to read further.

When I started writing this book, this chapter had the provisional title of 'Why Business Rules?' or some such. As I started laying out the reasons, it became clear that I was ducking the main issues facing the world of IT (information technology) by thus restricting my focus. So I asked myself 'Why are we doing all this?'

According to Standish (1995; 2004), around 66% of large US projects fail, either through cancellation, overrunning their budgets or delivering software that is never put into production. Outright project failures account for 15% of all projects, a vast improvement over the 31% failure rate reported in the first survey in 1994, but still a scandal. On top of this, projects that are over time, over budget or lacking critical features and requirements total 51% of all projects in the 2004 survey. 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. Clearly we should be doing something differently. The Standish surveys also looked into the reasons why people involved in the sample projects thought such projects fail. 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; and
- lack of planning.

The first four of these relate strongly to the need for better requirements engineering and 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 nowadays 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 developers 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.

Furthermore, the level of abstraction at which we work is far too low. IT departments are often culturally and technically miles away from the concerns and thought processes of the customers they serve. The problem is, thus, far broader than the need for business rules management; the real problem we have to solve is how to align IT practice with business need.

To believe that adopting a business rules management system on its own will solve this problem is nothing short of naïve. Business rules management is only a part of the solution. To align IT with business we must also consider innovative approaches to requirements engineering and service oriented architecture. Whilst its focus remains on business rules, this book is about all these issues.

Briefly – because the next chapter will be devoted to a detailed discussion – service oriented architecture (SOA) is an architectural concept in software design that emphasizes the use of combined services to support business requirements directly. In SOA, resources are made available to service consumers in the network as independent artifacts that are accessed in a standardized way. SOA is precisely about raising the level of abstraction so that business processes can be discussed in a language understood by business people as well as IT folk. Business rules are about aligning IT with the business too. It is to them we now turn.

In this chapter, after a short look at the history of the idea and technology of business rules management systems (BRMS), 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.

In subsequent chapters we will relate business rules to the concept of service oriented architecture, look at different approaches to and philosophies of business rules management, cover the key technical features of a BRMS (including *inter alia* knowledge representation and inference techniques) and discuss requirements engineering, appropriate development methods and processes. Next we try to distil this knowledge into a prototype pattern language.

1.1 Historical Background

The first talk of business rules management emerged from discussions in the database community as long ago as the late 1980s, notably in a journal called *The Database Newsletter* – although the term was used as early as 1984 in an article in *Datamation*.

However, there is an older tradition in the artificial intelligence (AI) community going back, arguably, to EMYCIN, the first so-called expert system shell. MYCIN (Shortliffe, 1976) was an expert system that could diagnose infectious diseases of the blood – with some success too. MYCIN was not, in any sense, a business rules management system; its rules were pretty much hard coded and concerned a fairly esoteric domain: medicine. EMYCIN (Melle *et al.*, 1981) was 'empty' MYCIN: MYCIN with the rules taken out and two significant mechanisms. First, rules on any suitable domain (including business domains) could be typed in and run under the control of the same logic used by MYCIN. Secondly, an EMYCIN application could be asked to explain its conclusions when asked 'How?' or 'Why?' I will explain how all this works in a later chapter. For now, notice only that EMYCIN separated business rules from both data and the control logic that enabled conclusions to be reached, and this is a key principle of modern business rules management systems. Furthermore, the rules were entirely declarative (unconnected statements rather than the interdependent lines of a computer program); another key principle of the business rules approach.

The first implementations of business rules in databases were more limited in several ways, the first being that rules were usually implemented as stored procedures written in procedural and proprietary extensions of declarative SQL. Other rules, notably those for referential integrity, were implemented in the database system itself, but nothing more complex was to succumb to this approach. The next step forward took some time. Active databases incorporated triggers: if/then rules that caused updates dependent on the values entered into the database. But even triggers did not offer the flexibility of EMYCIN's general if/then rules.

As an example of the gulf between the two traditions, I recall attending the British launch of Sapiens (still a major player in the BRMS marketplace today) in around 1989. I have a fairly low tolerance for sales pitches, but I was aroused from my slumbers when told that the product (basically a database and 4GL) was object-oriented and rule-based. As the technical presentation wore on, it became clear that the 'objects' were merely relational tables; by the end nothing much had been said about rules.

'Can you give me an example of a rule?' I asked.

'Of course. All employees must be over 16.'

'No, I mean a proper rule with an "if" and a "then".'

The speaker paused for a second. 'OK, then. If you are to be an employee then your age must be greater than or equal to 16.'

I decided to hold my peace, and went away rather unimpressed.

The point here is not just that salesmen can sometimes be rather uneconomical with buzzwords, but that there is a misunderstanding about what constitutes a rule (and, indeed, an object in this particular case). I regarded the example given as a range constraint on an attribute, rather than a rule. What is evinced is a lack of common terminology among the two camps.

Consider the following (very slightly edited) dialogue between MYCIN and a human physician.

```
>Is the patient's illness with ORGANISM-1 a hospital
 acquired infection?
>WHY?
[i.e. Why is it important to determine whether or not
  the infection with ORGANISM-1 was acquired while the
  patient was hospitalized?]
>This will aid in determining the category of ORGANISM-1.
 It has already been established that
    [1.1]the gram stain of ORGANISM-1 is gramneg, and
    [1.2]the morphology of ORGANISM-1 is rod, and
    [1.3]the aerobicity of ORGANISM-1 is facultative
 Therefore, if
    [1.4]the infection with ORGANISM-1 was not acquired
     while the patient was hospitalized
 then
    There is strongly suggestive evidence (.8) that
     the category of ORGANISM-1 is enterobacteriaceae
 ALSO:
    There is weakly suggestive evidence (.1) that
     the identity of ORGANISM-1 is pseudomonas-aeruginosa
```

Even ignoring the specialized terminology, it should be clear that the implied rule is far more complex than a constraint saying that staff entered in the database must be over 16. We will see many examples of similarly complex rules in more familiar domains as we proceed. Furthermore, we will encounter more complex constraints that involve more than one attribute, object or database table.

The first step towards a reconciliation between these two camps came with Ron Ross's (1994) *Business Rule Book*, to be followed by his several subsequent publications that show that he is aware of both traditions, though mainly rooted, originally, in the database world. Ross founded Business Rule Solutions in 1997 to focus on applied business aligned models (strategy, process, vocabulary, rules, etc.) that would be completely independent of any IT tradition.

In 1995, a group of IT practitioners produced the GUIDE *Business Rules Project Report*, which also clarified the territory, though remaining database centred. The manifesto of the (now better informed) database-centred approach was finally published by Chris Date (2000). In the same year, the Business Rules Group published the first version of the *Business Rules Manifesto*, which established the ground rules for what constitutes a BRMS and the principles of the business rules approach. By 2002, Barbara von Halle, another database guru, had published the first comprehensive method for applying the approach and Tony Morgan became the first AI expert to publish a book on the subject.

In the interim, products evolved. Some of them were extensions of database or repository products, others evolved from expert systems shells. We will look at some of them later.

As I write, it seems to me that there is now enough maturity in both theory and practice for commercial organizations to apply the business rules approach, along with mature object-oriented modelling techniques, better requirements engineering and the philosophy of service oriented architecture, to the critical problem of aligning IT with business goals and practices.

1.2 What are Business Rules?

Most early definitions (e.g. Appleton, 1984) conflate business rules with database constraints. Ross (1987) is more general, defining a business rule as a rule or policy that governs the behaviour of the enterprise and distinguishes it from others. Elsewhere (1994), he defines a rule as a 'discrete operational business policy or practice', and insists that a rule is a declarative statement expressed in 'non-technical' terms. Of course some business domains are replete with technical jargon, so perhaps 'non-IT' is what is intended. The declarative point is key. Declarative is the opposite of procedural. In a procedural rule language the order of execution of the rules matters; in a declarative language the outcome is the same whatever execution order is selected. Date (2000) makes the same point, insisting that rules convey 'what not how'.

Halle (2002) sees rules as conditions that 'govern a business event so that it occurs in a way that is acceptable to the business'. Date (2000) makes it clear that these 'business events' are to be viewed as events that result in an update to a database; the rules are there to ensure that rogue updates are not allowed. Date too insists on the declarative nature of rules; he sees rules as predicates (statements that are true or false) concerning the database domains.

The GUIDE project (Hay and Healy, 1997) saw a rule as defining or constraining some aspect of a business and 'intended to assert business structure, or to control or influence the behaviour of the business'. Such a rule 'cannot be broken down' without the loss of important information; i.e. rules are **atomic**. But GUIDE too deliberately restricted its scope to row 3 of the Zachman framework (Zachman, 1987); i.e. to 'specific constraints on the creation, updating and removal of persistent data in an information system'. However, there is a major acknowledgement of the rôle of inference. GUIDE said that facts could be derived by mathematical calculation, deductive inference and even induction (i.e. data mining). It went so far as to say that each of these three derivation methods is 'itself a kind of business rule'.

The Business Rules Group, taking on the mantle of GUIDE, has given various revisions of the definition such as: 'a directive that is intended to influence or guide business behaviour ... in response to risks, threats or opportunities'. More importantly, the Business Rules Group has published the Business Rules Manifesto (reproduced as Appendix A). The manifesto provides principles, rather than a definition, insisting that rules are atomic, declarative, logically well-formed, separated from processes, procedures and technology and, critically, written in business terms.

In what is probably one of the best and most sensible and practical books 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). One focus in this book will be on the ease of expression of rules and the suitability of available products for business owners, business analysts, as well as on their technical features.

It is difficult to fault any of the above definitions, except if one were to criticize them in terms of scope and emphasis. I can find little or nothing to disagree with in the Business Rules Manifesto (BRM). To me, Morgan's definition seems to capture the essence of the notion best. However, there is one issue unaddressed so far.

All these definitions emphasize one business. Open business on the web, closer customer relationships, and collaborative ventures all indicate a need to share business rules. Some rules could be about more than one business. Some rules could be imposed by one business on another (e.g. taxation rules). Some rules might be better shared with customers – perhaps in the form of explanations (a BRM principle). Taking this into account and picking up some points from all the definitions, here is my definition for the purposes of this book, based most chiefly on Morgan's.

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

The term 'well-formed' comes from logic and needs explanation. The rules must be executable on a machine if they are to be of much use in a business rules management system. This implies that they must be convertible into statements in some formal logic: statements that are **well-formed** with respect to that logic.

One corollary of the declarative principle is that business rules do *not* describe business processes; they do, however, constrain what processes are permissible.

Business rules are statements expressed in a language, preferably a subset of a natural language such as English. I see two clear kinds of statements that must be distinguished: assertions and rules. **Assertions** or **facts** have the form: 'A is X' or 'P is true'. These are equivalent forms; e.g. I can convert the former into '"A is X" is true'. Simplifying slightly, until later in this book, **rules** have the equivalent forms: 'If A then X'; 'X if A'; 'When A then X'; and so on. Here X can be a fact or an action.

We can see from Table 1.1 that rule statements can be classified. Date, Ross and Halle all offer useful classification schemes, but I do not want to be so specific here.

Table 1-1 Examples of statements and their types

Eeyore is a donkey.	Assertion
Computers come in blue boxes.	Assertion
NetMargin = 2,000.	Assertion
Bill Gates is wealthy.	Assertion
If the computer's box is not blue then paint it blue.	Action rule
To paint something: acquire funds, visit shop, buy paint, paint article.	Procedure
Wealthy people are always tall and handsome (if Z is wealthy then Z is tall and handsome).	Rule
NetMargin = Revenue − Costs.	Procedure or Rule
Employees must be over 16.	Range constraint or Rule
A borrower may borrow up to 6 books.	Cardinality constraint or Rule
A borrowed book must be owned by the library that the member belongs to.	General constraint or Rule
If any employee has a salary greater than the MD then set the MD's salary to the maximum of all employee's salaries.	Trigger rule

Statements are always statements *about* something. Ross refers to these somethings as **terms**. Other authors refer to the **vocabulary** of the domain or even the **domain ontology**.

Strictly, ontology is the philosophical science concerned with what exists: the science of Being. Here, though, it is used to mean the model of the domain that we work with, including the things we can discuss, their properties and how they relate to each other. I will take the view in this book that the domain ontology is precisely an object model, usually expressed by a UML type diagram; but more on that later. Some readers might like to think of the ontology as the database schema – at least for the time being. The ontology tells us what we are allowed to discuss when we write rules. Without a sound ontology the rules are meaningless, and any attempt at writing them in natural language is certainly doomed. This means that we must modify our definition slightly. We can do so by adding just one sentence.

Business rules are always interpreted against a defined domain ontology.

Having defined what business rules are, there is still much more to say about them, such as how they may be linked together to derive new facts (inference), how they are best written (rule structure) or how they are to be discovered (knowledge elicitation). We will return to these topics (and more) in subsequent chapters. For now, let us take a look at how rules may be managed.

1.3 What is Business Rules Management?

Business rules management is the practical art of implementing systems based on the business rules approach. This can be done in many ways, but the most economical is to use a business rules management system. In addition, there will be some process adopted for managing and organizing projects and conducting tasks such as rule authoring, rule maintenance, and so on. We will return to such issues later.

Let us start with business rules management systems.

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 **inference**s 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 idea that the rules are stored in a repository is a critical one. If we are to manage rules there seems no alternative to storing them in some sort of central database. Furthermore, storing the rules in a layer separate from both applications and from the various databases that may exist in a real organization gives obvious maintenance advantages. We might even argue that centralizing the rules makes them more readily reusable. However, there is an opposing force: that of the need for reuse of the objects in our domain model. If the rules (and indeed rulesets) are not encapsulated within the objects that they constrain, then those objects are incomplete and, if reused, may function incorrectly.

Date (2000) also argues that, ideally, rules should be part of the database but then, rather reluctantly, concedes that storing the rules in a separate layer gives the advantage of DBMS-independence. Contrariwise, Bruce (1992) points out that treating rules separately 'avoids the debate over which object (or objects) should encapsulate the rules'. This is indeed a hard problem sometimes, and I will return to the issue in subsequent chapters. All design problems concern the resolution of contradictory forces such as the ones referred to: reuse *versus* independence. In Chapter 7, I present some patterns aimed at resolving these forces. For now, assume that rules live in a repository and are managed thereby.

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.
- Even in unregulated industries, companies subject to the Sarbanes-Oxley Act are required to make their business processes (and thus the rules that they follow) visible. If such rules are scattered through multiple applications, duplicated (consistently or otherwise) in different places and embedded in procedural code, this becomes a costly and nigh impossible exercise.
- 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:

- Automating procedures for such things as
  - ☞ 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.

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. It is these objects that are modelled by our domain ontology objects.

Morgan (2002) 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.
- 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.

1.4 Why use a Business Rules Management System?

As I have pointed out, according to Standish (1995; 2004) around 66% of large US projects fail. Clearly we should be doing *something* differently.

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 component(s) or service(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) can be stored (at least conceptually) within those objects for better reuse.

The separated rulesets need to be maintained and kept under version control. This implies that a good BRMS will store rulesets centrally in a repository. As we shall see later, the apparent contradiction between the need for encapsulation and centralization can be resolved using patterns, notably the POLICY BLACKBOARD and ENCAPSULATE A REFERENCE patterns (cf. Chapter 7).

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. 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 Chapter 4, Section 4.2.1 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.

1.5 The Benefits

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

- Faster development.
- Faster maintenance, which is 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.

However, business rules management systems alone will not suffice. They need to be implemented side by side with business-oriented requirements engineering, best practice in object and component modelling and, I believe, service oriented architecture. It is to the latter topic that we turn in the next chapter.

And there are a few pitfalls. There can be technical problems in debugging a system with thousands of rules. Large, badly segmented rulesets become increasingly difficult to manage. This is because rules are sometimes invented in restricted contexts which do not consider all the background assumptions explicitly. As more rules are added to handle particular or exceptional cases, this can affect the global consistency of the ruleset and the ability to select which rules should fire. These problems are best addressed by paying attention to sound requirements engineering practices, solid architectural patterns, the dangers of potential 'entropy' in rulesets and conflict resolution strategies (such as most-recent, least-recent, refractoriness; cf. Jackson, 1986).

1.6 Summary

Software engineering practice has not delivered on its promises and needs to change.

The problem we need to solve is broader than just business rules management; the real problem is how to align IT practice with business need.

There are two traditions underpinning the business rules approach: database theory and AI.

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

Business rules management systems separate the rules from data and control logic and maintain them in a repository. Rules are grouped into rulesets, and inference over and within rulesets is both possible and transparent.

BRMSs have applications across all industries and many types of business problem.

Businesses strive for shorter time to market and lower development and maintenance costs. Business rules management technologies can play an important rôle in this. Using BRMSs together with better requirements engineering and business modelling within the context of SOA should decrease development costs and dramatically shorten development and maintenance cycles.

1.7 Bibliographical Notes

The seminal works on the business rules approach are probably those by Ross (1987; 1994; 2005), Date (2000), Halle (2002), and Morgan (2002).

I have assumed in this chapter that the reader is familiar with certain well-known developments in IT. If any terms are in fact unfamiliar, Graham (2001) contains a discussion of database terminology and introduces the Zachman framework. It also contains useful background on object modelling and my approach to requirements engineering.