Business Rules Management and Service Oriented Architecture

A Pattern Language

Ian Graham

John Wiley & Sons, Ltd

Business Rules Management and Service Oriented Architecture

Business Rules Management and Service Oriented Architecture

A Pattern Language

Ian Graham

John Wiley & Sons, Ltd

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester,
West Sussex PO19 8SQ, England

Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.co.uk
Visit our Home Page on www.wiley.com

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 6045 Freemont Blvd, Mississauga, ONT, L5R 4J3, Canada

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN-13: 978-0-470-02721-9 (PB)
ISBN-10: 0-470-02721-5 (PB)

Typeset in 10.5/13 Palatino by Laserwords Private Limited, Chennai, India
Printed and bound in Great Britain by Bell & Bain, Glasgow
This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

Contents

Trademark Notice

Foreword

In *Business Rules Management and Service Oriented Architecture*, Ian Graham provides a solid architectural introduction to business rules for IT professionals and architects taking the next steps into SOA, components, and other state-of-the-art software engineering techniques. He speaks of concerns I find just about every IT architect has these days, offering a wide-ranging set of solutions. It's a compelling story.

Let me share with you briefly some of the things Ian gets right in this book.

- Separating concerns of business from those of the infrastructure (the 'plumbing') is fundamental to building better architecture. He deftly explains how both business rules and SOA can help you make that happen.
- SOA and business rules management systems (BRMS) are parallel and complimentary technologies. They're both about the quest for agility – creating new levers to manage (and encourage!) endless, fast-paced change. Is there anything much more urgent than that these days?
- It's all about re-use – but the right *kind* of re-use. A BRMS allows reuse of rules across services. Why does that matter to you? You want your services to be easily reconfigured. When the business changes, you want to be able to change the rules without ever digging into the code. Ian claims (and I certainly agree) that this alone can speed development and ease maintenance even more than the adoption of SOA on its own.
- There are some areas where I'm afraid we need a bit of attitude adjustment. (Those are my words, not Ian's – he's much more diplomatic about it.) Creating a business model is *not* a waste of time. More and more descriptive use cases are *not* going to solve all your problems!

I could go on and on about that last one, but Ian more than does the topic justice, so I'll just invite you to jump right into the book.

- The notion of business rules is on an inevitable collision course with the notion of *patterns*. As one who studied this area a great deal in the formative years of business rules (*The Business Rule Book*, 2nd edition was published in 1997), I applaud Ian for breaking new ground in this important area.

There are many other things I could mention about what Ian gets right in this book. For example, what about legacy systems? Ian points out how adopting a BRMS will assist in the transition to SOA because service-based and legacy applications can be coupled using the BRMS as the common decision engine. What about semantics and pragmatic management of business intellectual property (IP)? Business rules provide a pragmatic, proven answer on that one.

The main thing I want to mention, however, is that Ian says he finds little to disagree with in the *Business Rules Manifesto* (Business Rules Group, 2003). That's an important statement – one that as an IT architect you should find comforting – because it reflects a growing consensus in the industry as to just what business rules are about. I mentioned the 'formative' period of business rules above – well, that period is just about over. By any reasonable measure, business rules are mainstream now. Do have a quick look at the *Manifesto* – it's conveniently included right here in the book for you. Incidentally, the *Manifesto* has been translated into about a dozen languages as of this writing. It's impact is truly global.

It's exciting to see new ideas become reality. That's especially so when the ideas make the professional's job easier, and the resulting systems better for business. Fortunately, Ian's work is highly approachable. If you want to know how to go about building a world-class rule-based, service-oriented architecture, read on!

Ronald G. Ross
Executive Editor, www.BRCommunity.com
Principle, Business Rule Solutions, LLC

Preface

There has been a great deal of interest in business rules management systems (BRMS) for several years now and the technology has matured considerably. At last it seems that the time is ripe and a plethora of commercial applications are beginning to be fielded, driven by the escalating difficulty of maintaining essential computer systems, the onus of greater regulatory compliance, the increasing complexity and volatility of business processes and many other factors. The existing literature is surprisingly sparse and most of it approaches business rules management from the standpoints of database practice and project management or concentrates on perfecting rule syntax. All these approaches are valuable but the origins of the subject are more diverse. It is now time, therefore, for a concise but comprehensive look at the subject that gets away from both database-centred tunnel vision and from the exaggerated (and thus discredited) claims of the erstwhile expert systems community.

The other factor that has moulded the approach I have taken here is the massive explosion of interest in service oriented architecture (SOA), one of the most significant potential steps forward in computing for a decade. Here there is much confusion. Some commentators seem to identify service oriented architecture with web services, whilst others claim that the main idea is to build an 'orchestration' layer that will glue any new services together with APIs to the goulash of legacy systems. Both these claims are wrong and the second one is downright dangerous. With many of my clients now adopting SOA (and some implementing business rules too) I have become more convinced than ever that the key to success with both technologies is to pay serious attention to modelling not only systems issues but the business environment as well. With the help of my colleague Derek Andrews, I have tried to explain this in Chapter 2.

As the manuscript developed, and looking constantly at the interactions between these strands, I found that a constant theme emerged, almost organically, from my researches, practice and discussions: service oriented architecture without business rules management is not going to crack the nut. Similarly, BRMS without SOA is unlikely to address all the pressing needs of business that so desperately need addressing by IT practice. So the propaganda message here is simple: SOA and BRMS; do both, or don't bother with either. *And* do them on the basis of first class requirements engineering and modelling practices too.

Even companies that decide not to invest in a full-blown BRMS product can benefit from externalizing their rules; writing them down in a clear and consistent style leads to immediate benefits. One of my clients, for example, is developing its own customized variant of Ross's RuleSpeak. This will enable their business analysts, users and developers to communicate more effectively and has already led to the discovery of errors and inconsistencies in existing documentation.

What the Book Covers

The aim of this book is to bring together the following key ideas in modern enterprise system development best practice.

- The need to separate business logic cleanly from the software 'plumbing'.
- The need for service-oriented architecture.
- How the former depends on component-based development (CBD).
- Database-centred approaches to business rules.
- Knowledge-based approaches to business rules.
- Best software engineering practice for designing robust, flexible systems and aligning IT with business more closely than has hitherto been the case.
- Using patterns to design and develop service oriented business rules management systems.

The text starts with a business case for adopting BRMSs and surveys the wide range of possible application areas for this technology. Then we present a tutorial on and discussion of service oriented architecture, its role, concepts, and supporting technologies. In this chapter we meet the central role of modelling in the design of successful computer systems, which a major theme of the book. The ideas of greater business alignment and of intelligent software agents are used to pull together the two strands of BRMS and SOA. Chapter 3 is an historical digression looking at the sources of the main ideas of BRMS, but it also discusses trends and emerging standards.

Chapter 4 is a technical tutorial on business rules management systems.

Chapter 5 applies the ideas of the previous chapters to existing and notional BRMS products.

Chapter 6 looks at knowledge elicitation and requirements engineering techniques insofar as they are specific to BRMS.

Finally, we gather together all the book's techniques and guidelines into a pattern language that is intended to be a 'how to' guide to running an actual BRMS/SOA project. Using the language, if done as intended, should generate specific solutions to a range of concrete development problems. The two appendices support the material in this chapter.

I believe that pattern languages are far more powerful and flexible than mere checklists. However, as with a checklist, no pattern language is ever complete and finished, and the reader will undoubtedly want to refer to the work of other authors as well as mine. Notably, I have drawn on the modelling patterns of Peter Coad when discussing SOA, Barbara von Halle's work on method and as yet unpublished SOA patterns under development by Derek Andrews, Hubert Matthews and (to a smaller extent) myself. On rule writing style, I have tried to capture the essence of the works on Ronald G. Ross and Tony Morgan but, as always, there is no substitute for reading the originals. The bibliographical notes to each chapter provide pointers to references of this kind. There are also some references to my own earlier works, notably those on requirements engineering, but I have tried to make such material self-contained within this text.

Intended Readership and Scope

The book is intended to be accessible to readers who do not have deep knowledge of theoretical computer science, but at the same time it attempts to treat the important issues accurately and in depth. It provides a tutorial on the technology and advice on how best to exploit business rules management in practice.

The primary audience is IT professionals (architects, analysts, developers, strategists, managers) and some of their interested customers. It may be of use to undergraduate and postgraduate students studying information technology or software engineering. It will therefore be of interest to teachers of Computer Science and Business IT. I have assumed that the reader has at least a nodding acquaintance with the basic UML notations for use cases, class diagrams and state models.

The book is designed to be read sequentially, although readers with differing interests may safely omit some sections. For example, readers with a less technical focus may skip the material on web services. The impatient reader, who already knows what backward chaining is, may even jump straight into

Chapter 7, which contains the RulePatterns pattern language. This chapter is intended as a stand-alone reference. Here too, the introductory sections (7.1 and 7.2) may be of no interest to people who already possess a sound knowledge of the idea of pattern languages.

While the scope of this book is broad and intended to cover the gamut of topics pertinent to a move to service oriented architecture and business rules management, it does not attempt to duplicate unnecessarily the work of other authors. Notably, there is scant consideration given to such issues as analytics, business rule maturity models, tying business rules to other deliverables, integration of the business rules approach with proprietary methods such as RUP or non-proprietary methods such as that of Halle (2002). These, I feel, are either already adequately dealt with in other texts or deserve a treatment separate from the one given here.

Acknowledgements

Although it contains much original material, this book is largely a survey of other people's work and could not have been written without that work. I would like to acknowledge the contribution of these other authors. Also, many of the ideas contained were honed in discussions with my colleagues at Trireme and participants in various conferences and seminars. In particular, at EuroPLoP 2006, Ademar Aguiar, Jon Bennett, Frank Buschmann, Alexander Fülleborn, Marina Hasse, Michaelis Hadjisimou, Kevlin Henney, Lise Hvatum, Maria Kavanagh, Alan Kelly, Klaus Marquardt and Martin Schmettow all made very helpful comments on some of the patterns presented in Chapter 7. Conversations with members of staff at some of my clients, whom I may not identify here, also provided invaluable insights and helped me keep my feet on the ground; thanks to you too, you know who you are.

Special thanks are due to Derek Andrews for his contribution to Chapter 2 and to him, Clive Menhinick and Hubert Matthews for many interesting and sometimes formative discussions. The remarks of several anonymous referees were very helpful too. Of the reviewers whose names I do know, I especially want to thank Barbara von Halle, Tony Simons, Ron Ross and Paul Vincent for their comments and kind suggestions for improvements. I have tried to incorporate them as best I could.

I am grateful to the Business Rules Group for permission to reproduce their seminal *Business Rules Manifesto* as Appendix A.

The team at Wiley were a joy to work with on this project. I don't know the names of all the production and other back-room workers there but I can express my profound thanks to all of them and to my editors, Drew Kennerley and Sally Tickner.

Even with all this help, the responsibility for any mistakes or omissions is entirely mine. If you can get past any of these that remain, then I do hope you find the book entertaining as well as merely informative. I would be most interested to read any comments you may have.

Ian Graham
Balham, August MMVI
(ian@trireme.com)

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).