

# Rule Modelling & Management

Author: Richard Veryard  
Version: April 1999

[richard@veryard.com](mailto:richard@veryard.com)  
<http://www.veryard.com>

For more information about SCIPIO, please  
contact the SCIPIO Consortium.

[info@scipio.org](mailto:info@scipio.org)  
<http://www.scipio.org>

# Preface

## Purpose of document

This document describes the SCIPIO approach to rule modelling.

## Questions and exercises



Question

## Acknowledgements

A number of precepts and examples have been borrowed from the Catalysis book and the Select Perspective book.

The Accounts Payable example was developed jointly with Michael Mills and Clive Mabey.

# Introduction

Rule modelling is one of the key techniques of SCIPIO. It complements object modelling and process modelling.

There are many uses of rule models within SCIPIO.

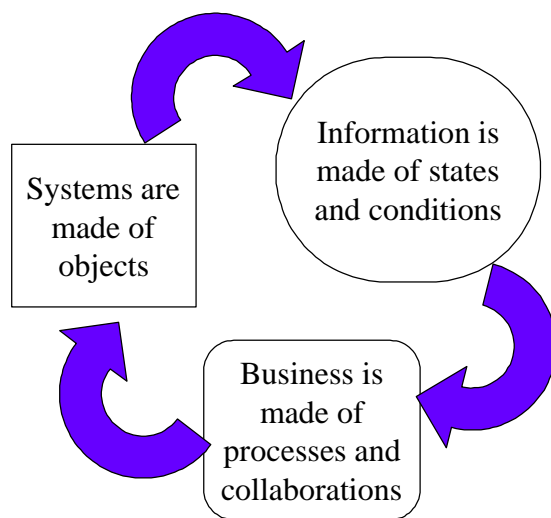
We use a tree diagram to describe and analyse the logical structure of conditions (including states, preconditions, postconditions and invariants). This allows complex logical statements to be presented and checked visually. It also allows branches of the tree to be labelled and reused.

We also use rule models when decomposing a set of business requirements into software components. In this context, it complements object-based decomposition and process-based decomposition.

The intention of the business rules tree diagram is twofold: (a) it should be meaningful to and verifiable by business people, and (b) a formal specification of a component in terms of its pre- and post-conditions can be derived from it.

## We need several ways of modelling the world.

We have identified three different ways of conceptualizing the world. These are sometimes called **paradigms**. They might also be called **ontologies**.



**Figure 1: Three ways of modelling**

### Objects

According to the Object-Oriented paradigm, the world can be understood as a system of objects. This is a useful insight, as long as it is not taken to imply that this is the only valid way of understanding the world.

## Processes: relationships and collaborations

A business is not just a group of connected things. It is not even just a group of connected people. In business, we are not merely interested in things and people that happen to have particular behaviours. We are interested in business processes and relationships.

One of the phrases used in English for a business is a "going concern". This phrase focuses our attention on two aspects of business: activity and intentionality.

The Process-Oriented Paradigm serves as a useful counterweight to the Object-Oriented Paradigm.

## Information: states and rules

Wittgenstein defined the world in terms of information: "The world is all that is the case."<sup>1</sup>

Bateson defined information in terms of difference: "A difference that makes a difference." <sup>2</sup> Our decisions are based, not on the absolute value of a given attribute of a given object, but on a comparison between values. Information (as opposed to raw data) requires a context that makes such comparison meaningful.

Within traditional information modelling, an attribute that makes a difference is known as a **state** attribute. Differences are made up of states. States, in the form of **conditions**, can be composed into **rules**.

And **knowledge** can be expressed as a set of higher-order rules.



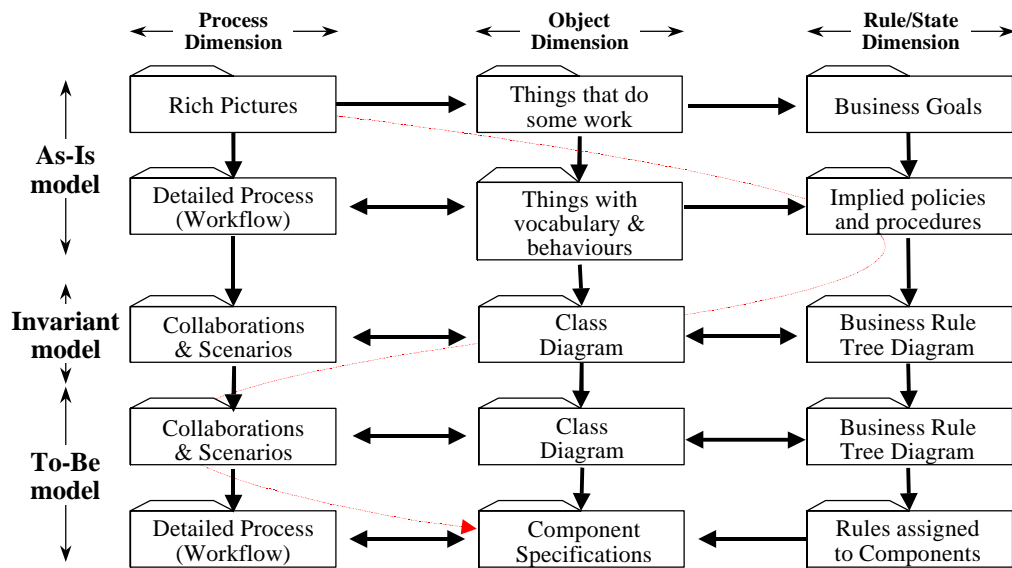
Is there a real difference between information and knowledge? If so, what is it?

---

<sup>1</sup> Tractatus

<sup>2</sup> Bateson references required.

We model three dimensions in parallel.



**Figure 2: Parallel Modelling**

## Motivation

### Expression of business rules

Much business modelling claims to express business rules of one form or another.

An explicit and precise statement of business rules helps with the identification of software requirements, and enhances the traceability (in both directions) between requirements and solution/specification.

### Implementation of business rules

Systems design claims to implement and enforce business rules. The business rules implemented and enforced by traditional data processing systems are mostly concerned with object associations (for example, every sales order must be delivered to a named customer) and activity sequence (for example, every sales order must be paid for before delivery).

Many systems incorporate constraints that are not driven by the business requirements, but are the result of the way the systems have been designed. In some cases, these constraints are deliberately chosen, for the sake of simplicity or cost. In other cases, the constraints result from some limitations in the technology or platform. But in many cases, these constraints can be traced to poor design practice - typically the designer or design team lacks awareness, imagination or technique. Such constraints are unnecessary.

By comparing the rule structure of the business requirements with the rule structure of the implemented systems we can identify, and possibly eliminate, unnecessary constraints from a system.

### Business rules and system flexibility

How do we scope components? And how do we make the components robust against change? We can answer this by addressing what makes systems inflexible.

The software industry has already learned (although it often forgets) how to make systems data-flexible. From normalization techniques, through to providing data services through a separate layer, we have many ways of containing (or encapsulating) the impact of data structure changes.

It is a widespread tenet of structured design methods (such as Information Engineering) that data structures are more stable than procedural structures. For this reason, many designers decompose a system into components from a data perspective. They typically draw clusters onto a class diagram, and use this clustering to drive the system design.

The software industry also knows how to make systems procedure-flexible, so that the users are not forced to perform tasks in a fixed sequence. This can typically be done by putting the workflow into a separate layer. Clustering services by business function is relevant here.

Furthermore, new technologies for open distributed processing, such as CORBA, provide ways of making systems platform- and location-flexible.

However, business change is often expressed in terms of a change in business rules, an alteration or exception to the rules in force. We therefore want to scope components to encapsulate rules. We draw the business rules in a tree diagram, and use this to identify suitable clusters. This provides an important alternative view to the data-oriented clustering or procedure-oriented clustering described above.

## Other methods

If rule modelling is so important, why don't other methods practice it?

Most methods recognise the importance of business rules. But these rules, instead of being expressed directly as rules, are translated into processes or data structures.

For example, in Catalysis, business rules are restated in terms of the following (in descending order of preference):

- Static invariants over the model
- Effect invariants that must apply to every action
- Timing constraints
- Action specifications

[D'Souza & Wills, p 551]

And in the Select Perspective, business rules are translated into processes and class invariants:

*A technique that helps in identifying indirect infrastructure processes is to consider them as the means by which the business ensures that required business rules are adhered to. ... Having identified the business rules, these can be translated into the associated checking processes; for example, Assess New Part Change or Manage Stock Level. Note that the associated business rules help to shape class modelling and component modelling; often they are implemented in terms of class invariants.* [Allen & Frost, p 55]

This can be interpreted as saying that the processes and classes are part of the specification of the **solution**, derived from an original **requirement** in the form of a set of business rules.

- Q Some people draw a careful distinction between requirements on the one hand and solution or specification on the other hand. Other people find this distinction unnecessary or cumbersome. What are the arguments on either side?
- Q What is your own position on this debate? Is the distinction sometimes important - if so when? Is the distinction sometime otiose - if so when?

## Note on complexity

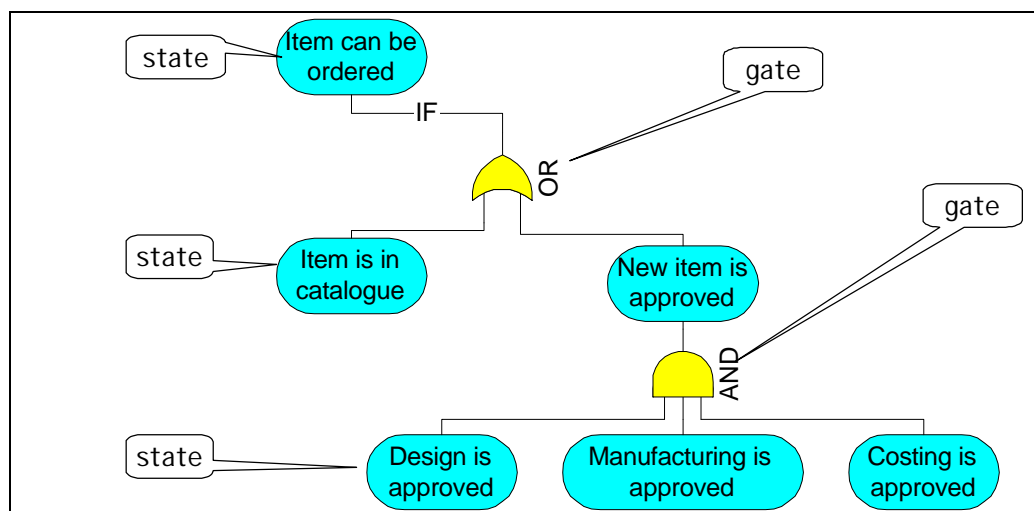
As a final warning to the reader, please note that most of the examples of rule diagrams contained in this document are extremely simple. Do not conclude from this that the technique is trivial. On the contrary, the logical complexity of the business rules can grow exponentially.

# Rule Models

## Simple rule diagram

The rule diagram helps to specify the logic of collaborations, processes or operations. It offers a user-friendly notation for first-order predicate logic. It can therefore be used to show:

- the invariants of a collaboration or process,
- the preconditions for a process or operation,
- the postconditions of an operation.



**Figure 3: Simple Rule Diagram**

One of the advantages of this notation is that, once a branch has been named and defined, it can be reused elsewhere. This enables and encourages top-down analysis of business logic, and enhances the reuse of components that encapsulate coherent chunks of business logic.

*We are a little puzzled that this diagram hasn't already been automated. We can only suppose that the kind of people who build tools are so comfortable with complex expressions in first-order predicate logic, that they fail to appreciate the value of presenting the logic in any other form.*

## Specifying rules for objects and collaborations

In Catalysis, many business rules can be expressed as **invariants**.

Here are two examples from the Catalysis book (p12).

- *for every CourseRun, its instructor qualifications must include the course*



- *for any Instructor, and for any date you can think of, the number of the instructor's outages on that date is never more than 1*

In Catalysis, these are expressed in a simple language of Boolean conditions and set relationships. More formally, they can be expressed in any language that includes first-order predicate calculus with arithmetic. Such expressions are ideally suited to formal implementation and verification techniques, and some of these are described in detail in the Catalysis book.

However, most business people find such expressions hard to read, interpret and verify, especially when they are at all complex.

## Naming and reusing rules

Compound rules are often equivalent to states. Thus in Figure 3, there is a rule corresponding to the state *new item is approved*, composed of three subconditions. Formally, we might refer to this rule by the state name; informally, we might prefer to name it as “the new item approval rule” or suchlike.

Once the compound rule is named, we can then reference it as often as necessary. Furthermore, when we come to system design, we probably want this rule to be implemented in one place only. This may lead us to the specification of a component whose sole responsibility is to approve new items, and/or to check which items are approved.

## Template rules

Catalysis also contains the notion of **model frameworks as templates**. These are more generic models, with generic placeholders instead of specific object names. These framework models also contain conditions, such as invariants and postconditions.

If the placeholders stand only for the objects, then these model frameworks can be expressed in the same first-order predicate calculus that we have already seen.

If placeholders are also used to stand for the properties and predicates, then we need a more advanced logical notation, such as second-order predicate calculus.

Here are two examples from the Catalysis book (p14).

- *job's resource must be okay for its requirement*
- *no overlapping outages for any resource*

In a particular context, such as instructors and courses, we can define precisely what is meant by such terms as *okay* and *overlapping*. But in a framework we lack this context. Thus we need to regard these terms as predicate placeholders. When we apply the framework to a particular type of resource, we shall need to provide a precise specification of these predicates.

# Worked Example

## Context

This example describes an Accounts Payable process. It is loosely based on the Ford Motor example, described by Michael Hammer.

The solution is well-known: to eliminate the invoice control activity altogether, and to pay suppliers automatically on delivery of goods. There are several possible ways of arriving at this solution. We are going to show how rule modelling provides one possible route to the solution.

## Establish business performance objectives

We need to establish the intentions of the Accounts Payable process. We can identify four:

1. **Payment Correctness.** Payments should only be made against genuine debts.
2. **Company Cash Flow.** Payments should not be made too early.
3. **Supplier Satisfaction.** Payments should not be made too late. We need to maintain good relationships with our preferred suppliers.
4. **Administration Costs.** Reduce headcount and complexity.

In Hammer's account, it is the last of these objectives that is the focus of the reengineering effort. Hammer does not mention the other three. But they are important to a complete analysis, because they underlie the high administrative costs.

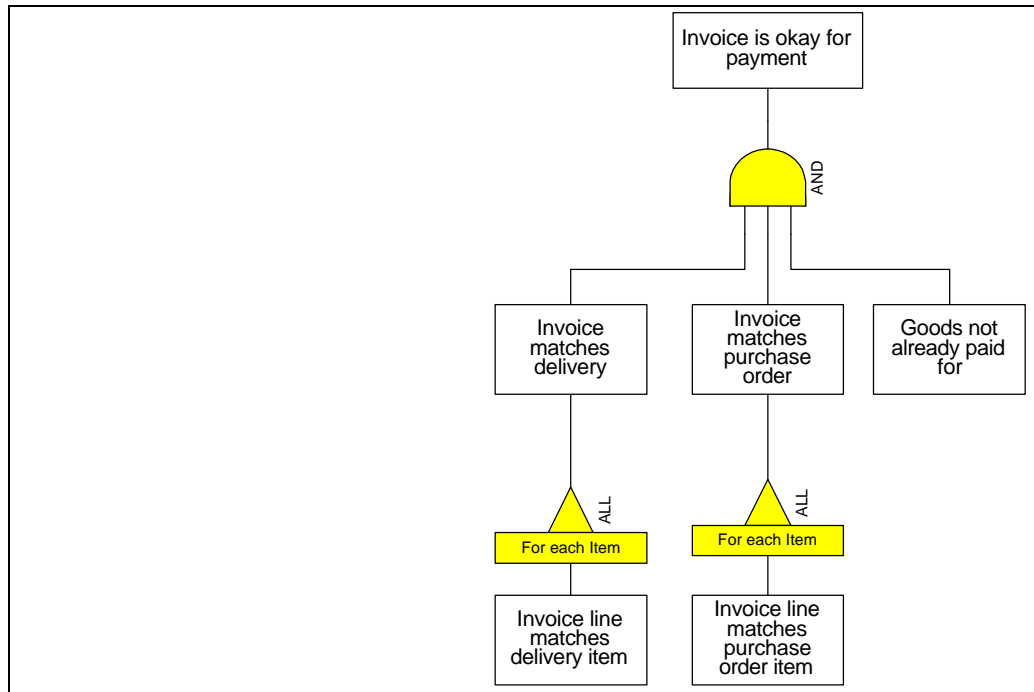
Put simply, if the above objectives were ignored, we could slash administrative costs in at least three different ways:

1. Pay every supplier invoice automatically, without carrying out any checks.
2. Pay for all supplies in advance, with the purchase order.
3. Never pay any suppliers until taken to court.

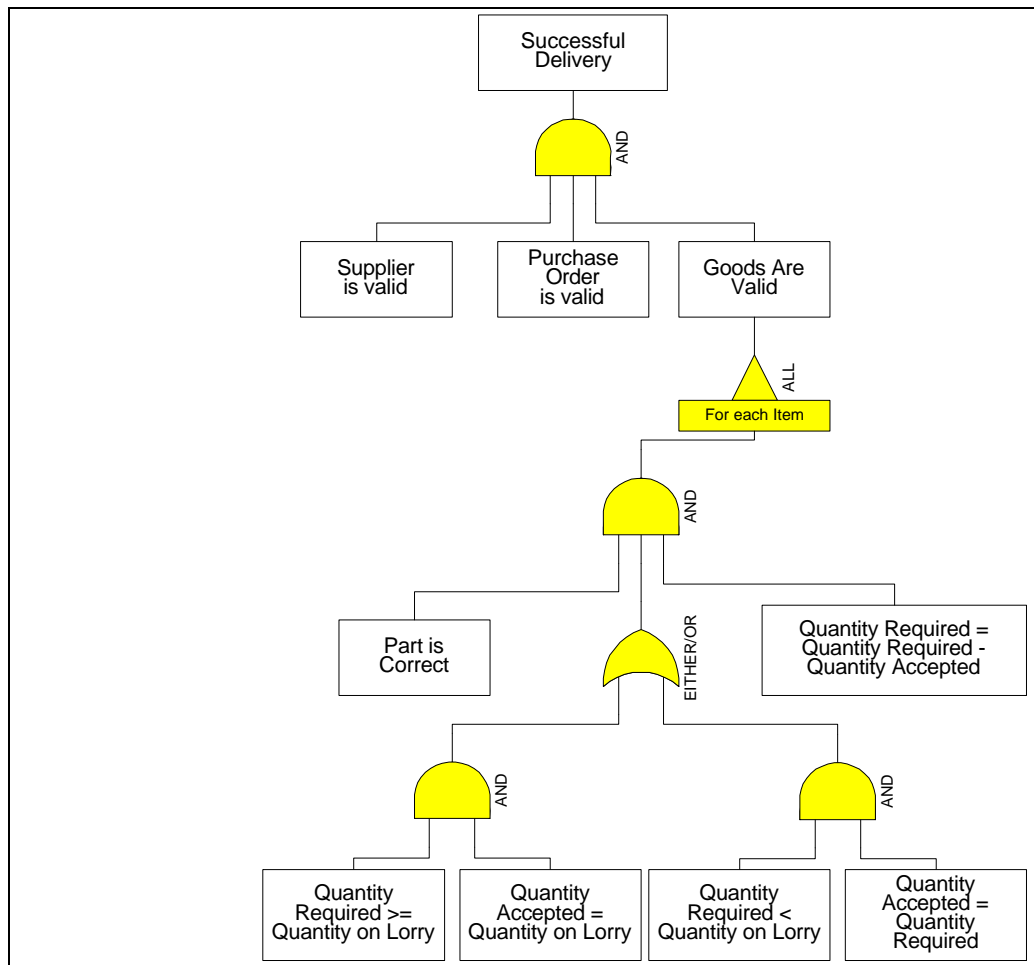
The invoice is a traditional mechanism for collaboration between a company and its suppliers, resulting in proper payment for goods received. This mechanism has been refined over centuries, to achieve a reasonably fair balance between the interests of the customer and the interests of the supplier. Thus before we get rid of the invoice, we have to understand exactly what the invoice achieves, in terms of Payment Correctness, Company Cash Flow and Supplier Cash Flow.

### Model existing rule structure

By analysing the behaviour of the current systems (both computerized and manual), we can infer the pre- and postconditions of each operation. Here is a pictorial view of the pre- and post-conditions of the Invoice Control and Delivery Control components (As-Is).



**Figure 4: Invoice Control Rules**



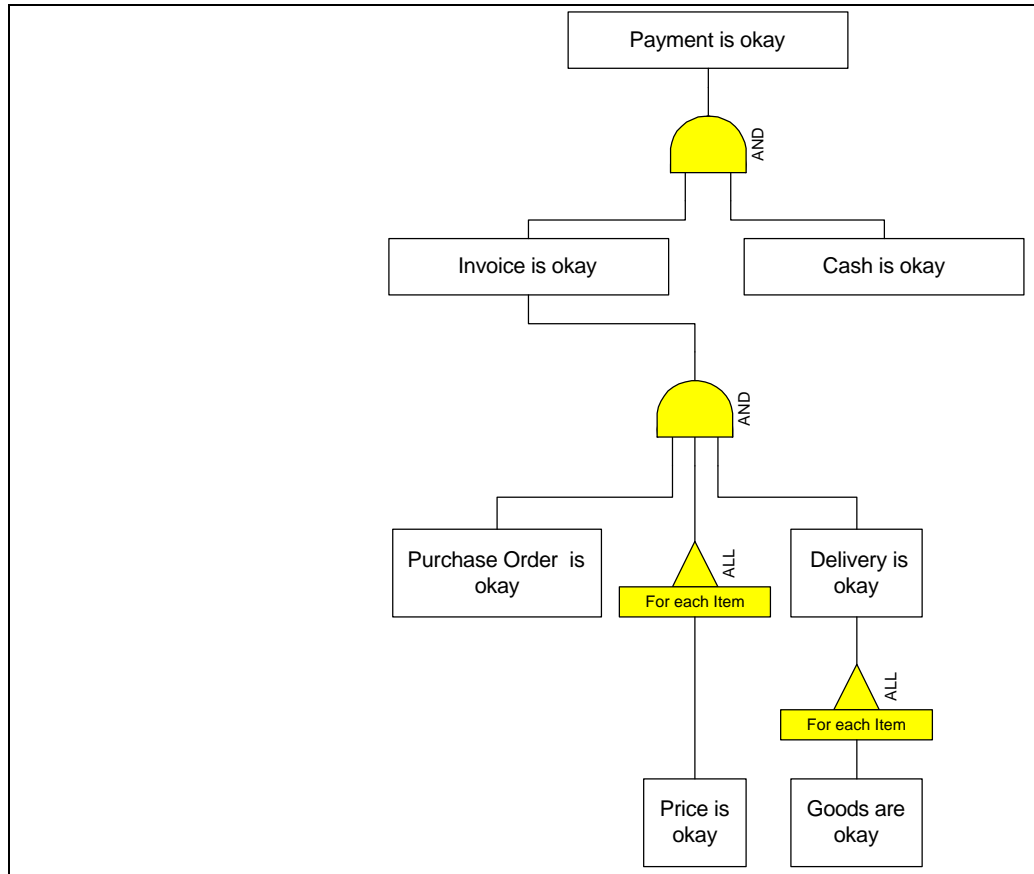
**Figure 5: Delivery Control Rules**

Note: Instead of the either/or clause, the business logic can be expressed mathematically using the MIN() operation, which returns the lower of two numerical values. The Quantity\_Accepted will be the minimum of Quantity\_on\_Lorry and Quantity\_Required. But such mathematical expressions often demand more effort to understand than does simple logic.

### Extract Business Rules

There are many checks and controls carried out within the Accounts Payable process.

If we analyze the purpose of these control mechanisms, we can derive the underlying rules that these mechanisms are intended to implement. These rules are essentially the preconditions for making a payment to a supplier. The rules can be expressed in a hierarchy.

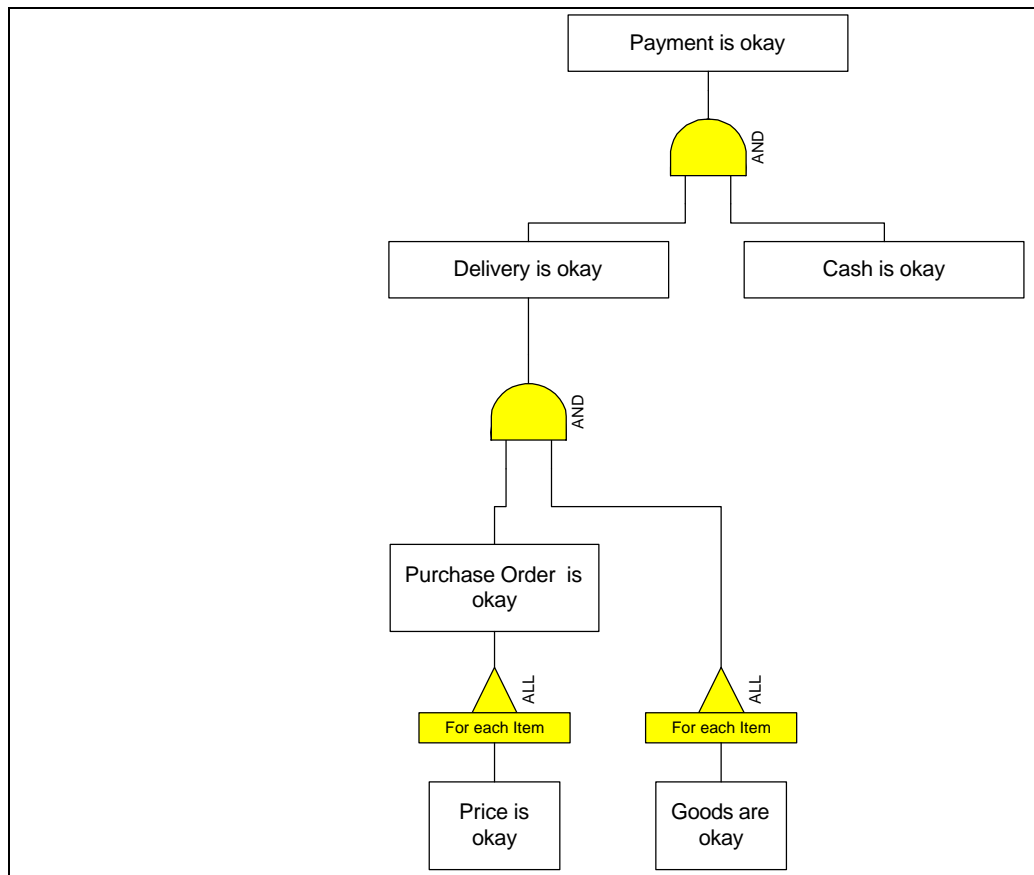


**Figure 6: Payment Rules**

### Simplify Business Rules

If we analyze the business rule hierarchy, we discover that the Business Rule 'Invoice is okay' can be entirely decomposed into rules that refer to other documents.

This indicates that, for the purposes of these business rules, the Invoice itself is logically redundant. We can draw another version of the rules hierarchy omitting the Supplier Invoice altogether.



**Figure 7: Simplified Payment Rules**

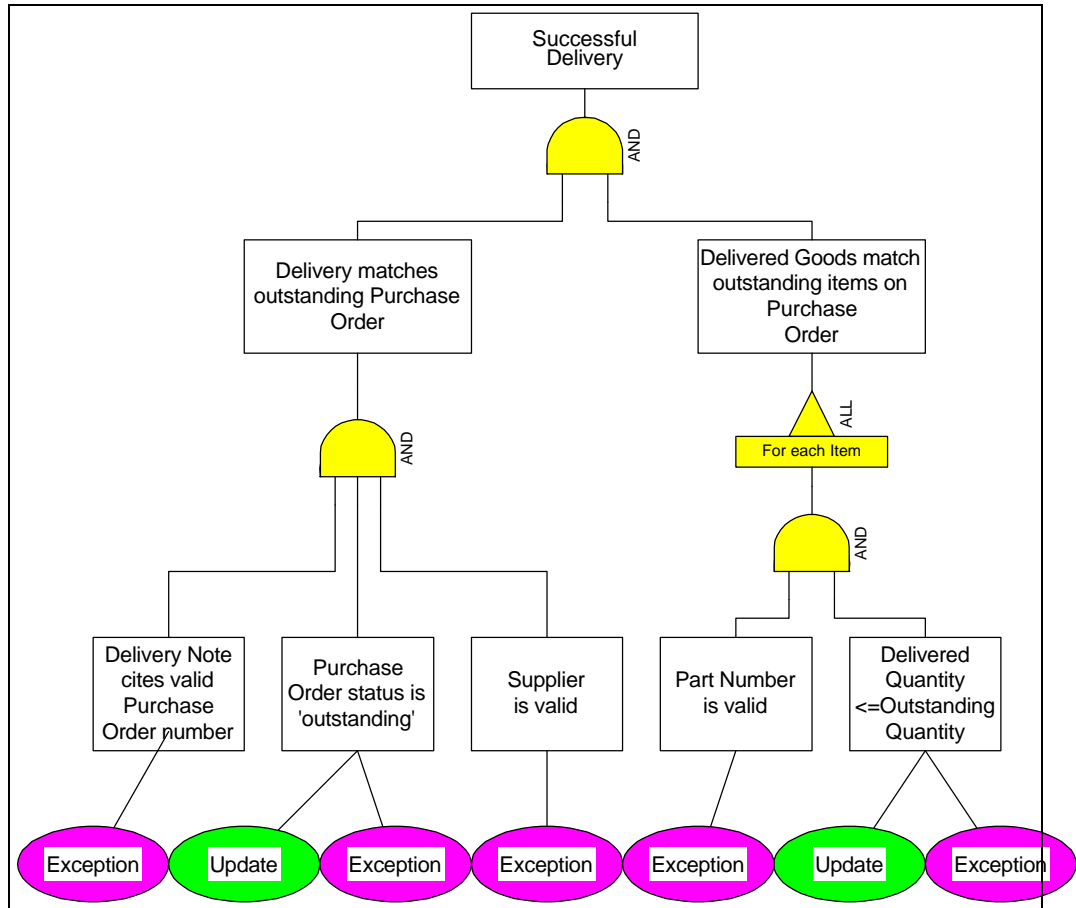
### Adjust System Responsibilities

What responsibilities can we displace onto the supplier, in return for taking on responsibility for triggering payment? It becomes the supplier’s responsibility (which may be delegated to the goods delivery agent) to match deliveries against purchase orders - otherwise the delivery will be turned away.

### Define Business Rules for Component

What counts as a successful delivery? We can specify the checks carried out by Goods Inwards in the form of a condition tree, which expresses the business rules associated with deliveries.

However in business virtually anything can happen. We therefore design Exception handling actions. If the exceptions are sufficiently frequent to require a designed solution then they become part of the process and are no longer exceptions. But in practice there will always want to be some flexibility which exception handling by an authorized expert can provide.



**Figure 8: Towards a Component Design**

# Investigating Rules

## Where do rules come from?

When analysing business rules, we are not just recording business material at face value. If we challenge the rules, we may be able to simplify them, or perhaps derive them from more general (and possibly more robust or more stable) rules.


There are typically three different sources of business rule or constraint. Although these may appear equivalent from the perspective of the computer system designer, their status is usually very different.

Demand-driven	This rule is determined by the nature of the customer or market demand, which the business process or service is intended to satisfy.
Supply- or capability-driven	This rule is determined by the limitations of the available capabilities, skills, resources, devices and platforms.
Regulation-driven	This rule is not constrained by demand or supply, but is imposed for the sake of standardization or simplification.  Regulations may be imposed by some external authority or agency, such as legislation or industry regulation. They may also be imposed within the organization, when they may be known as house rules. However, the distinction between external and internal is not always clear-cut.

The difference between these three types of rule becomes important when we are concerned with the ability of systems to adapt to possible changes in these rules.

Let us return to the invariants taken from the Catalysis book, and subject them to an imaginary critique.

- *for every CourseRun, its instructor qualifications must include the course*
- *for any Instructor, and for any date you can think of, the number of the instructor's outages on that date is never more than 1*

 What happens when the instructor is not qualified? Who (if anyone) cares?

Perhaps it is the students who care whether the instructor is qualified. Perhaps they are reluctant to attend or pay for courses with unqualified instructors. This would make it a demand-driven rule.

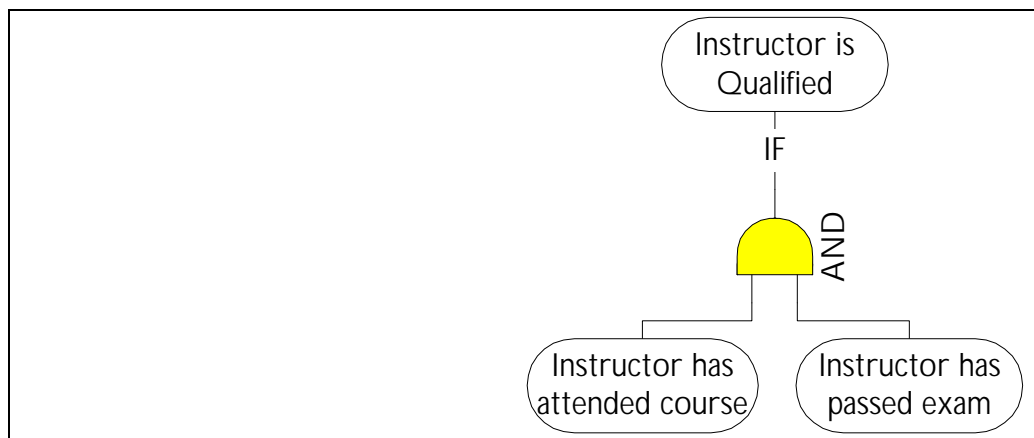
Perhaps it is the instructors themselves who refuse to teach courses for which they are not qualified. Perhaps they need the qualification to feel confident in front of a class, or to receive what they regard as a fair pay rate. This would make it a supply-driven rule.



Or perhaps it is part of maintaining some notion of professional standards. Perhaps the only people who care are the managers of the training company itself. This would make it simply a house rule.

- Q Suppose that this rule is required as a condition for membership of a voluntary professional training association. The managers of this training company have decided that they want to belong to this training association, and therefore to adhere to its rules. Furthermore, as members of the training association, the managers of our training company have some influence over future formulations of the association rules. Is this rule external or internal, or hybrid?
- Q In general, how useful is the distinction between internal and external rules?

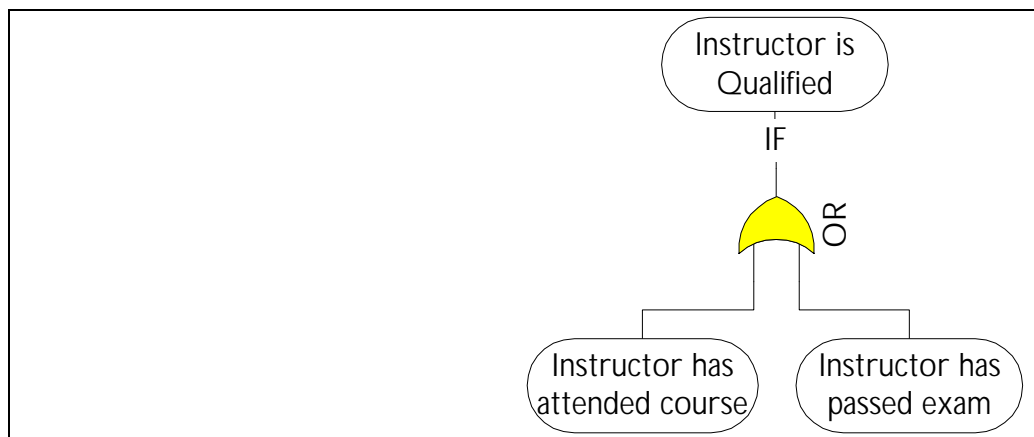
Let us suppose that the official qualification rule is as shown in Figure 9.



**Figure 9: Strong Qualification Rule**

- Q What happens if we want to allow exceptions to the instructor-qualification rule? How might exceptions be enabled? How might exceptions be managed?

One way of allowing exceptions is to weaken the condition.

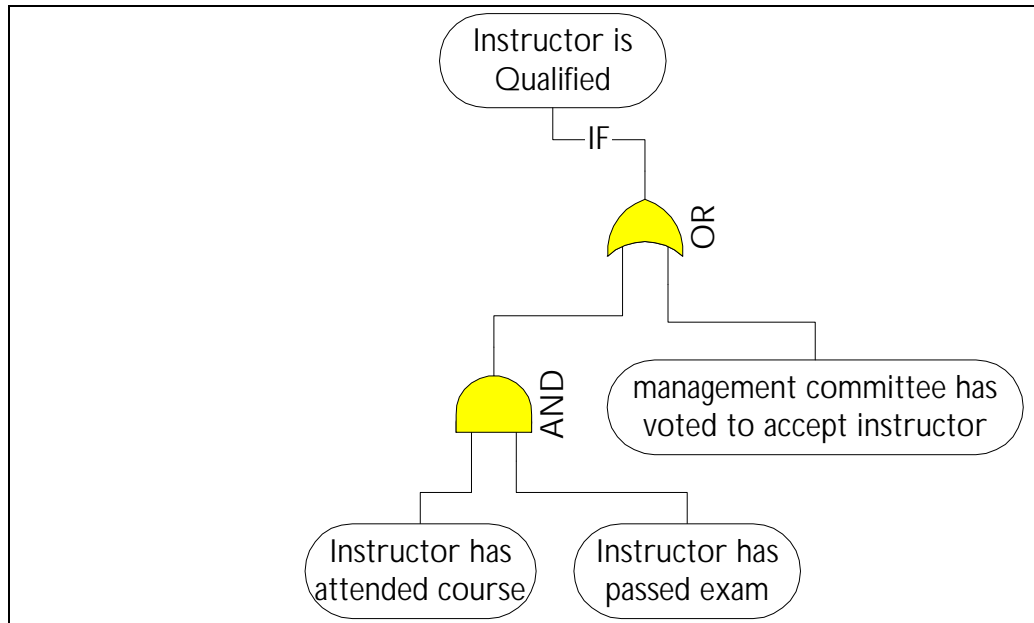


**Figure 10: Weak Qualification Rule**

**Q** In what other ways could the qualification rule be weakened?

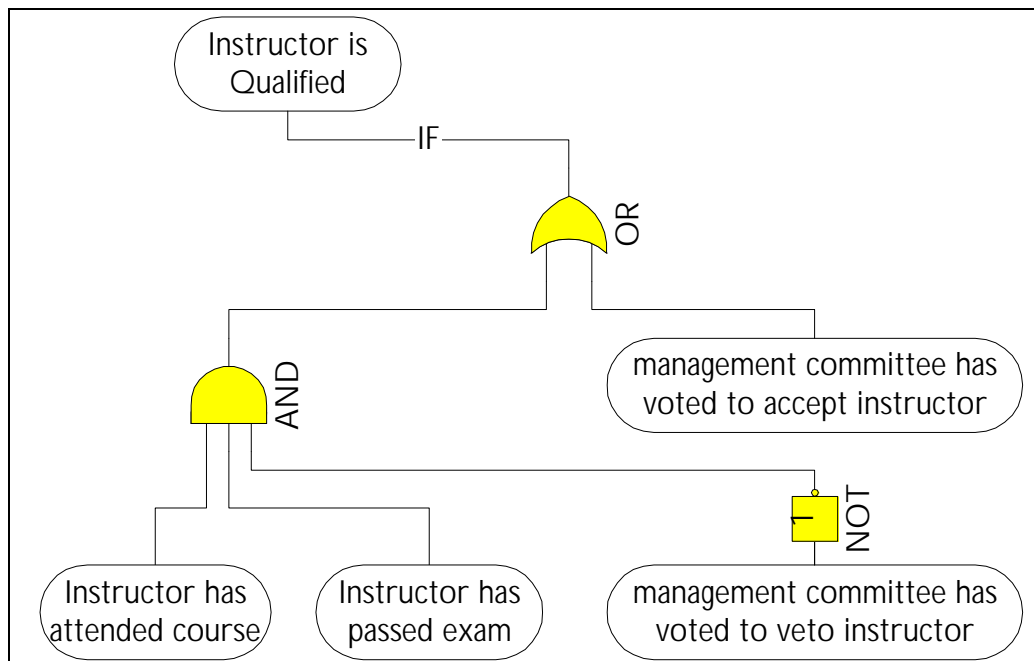
But if **anything** can count as a qualification, then the business rule that instructors should possess qualifications ceases to have any meaningful content.

Another way of allowing exceptions is to make the condition dependent upon some authorization process.




**Figure 11: Partially Authority-Based Qualification Rule**


In some cases, the authorization may swamp all other rules. “The decision of the management committee is final”.



**Figure 12: Fully Authority-Based Qualification Rule**


 If the management committee can overturn the results of the exam, is there any point in having the exam at all?

- *for any Instructor, and for any date you can think of, the number of the instructor's outages on that date is never more than 1*

 What prevents an instructor doing two things at once?

At first sight, this rule seems to be a supply-driven one - it is a consequence of the physical limitations of the instructor.

But with an appropriate communication system, perhaps an instructor could teach a course in London and another course in New York at the same time?

 How might we handle such exceptions to the rule?

## Exception handling

What happens if we want to alter a rule, or allow exceptions?

Before his untimely death, Imre Lakatos wrote a brilliant book called Proofs and Refutations. Among other things, this book identifies several different ways of preserving obsolete rules. Although his work was done for a different purpose than ours, we can apply his terminology to our purposes.

The key question here is: what happens when we confront a rule with an exception.

Surrender	Throw the rule away and start again.
Monster barring	Reject the exception as invalid, a "monster". This usually involves refining the definition of one or more constructs, to explicitly exclude the exception as a monster.  For example, we could say that teaching two courses simultaneously, one in New York and one in London, doesn't count as proper teaching.
Monster adjustment	Reframe the exception, so that it fits within the rule. This usually involves refining the definition of one or more constructs, so that the monster is "tamed", converted or decomposed into something that can be handled by the rule after all.  For example, we could say that, if the instructor is teaching in New York and London simultaneously, we define this, not as two courses, but as a single course with multiple locations.  Or perhaps we have two occurrences of INSTRUCTOR: one real and one virtual.

Lemma incorporation	<p>Reformulate the rule, to incorporate the exception into the condition.</p> <ul style="list-style-type: none"> <li>for any Instructor, <i>except those capable of teaching remotely</i>, and for any date you can think of, the number of the instructor’s outages on that date is never more than 1</li> </ul>
---------------------	---

- Q Do you recognize these ways of preserving rules? Can you think of any examples from your own work?
- Q What are the possible drawbacks or dangers of these ways of preserving rules?

### Proving and improving rules

In the context of evolutionary change, monsters can be of particular interest. Lakatos refers to the concept of “hopeful monsters”. In biology, this means mutations that could start a new evolutionary line if fitting into an empty environmental niche.

- Q Do you think the concept of “hopeful monster” is relevant to business modelling? How might you use it?

Sometimes the goal of rule modelling is to represent and implement the existing rules as accurately as possible. Sometimes the goal is to improve the rules, to make them more robust.

Where do the monsters come from?

Monster inclusion	Some people are particularly good at inventing adhoc freaks, which can be seen as exceptions to a given rule. These freaks are usually handled adhoc, and result in complications to the rule structure.
Error inclusion	Some monsters may originate in errors or misunderstandings of various kinds.
Concept stretching	Starting from the underlying concepts, we can broaden and generalize a given rule. This can sometimes result in simplifications to the rule structure.

Kevin Kelly tells us to honour our errors.

*A trick will only work for a while, until everyone else is doing it. To advance from the ordinary requires a new game, or a new territory. But the process of going outside the conventional method, game or territory is indistinguishable from error. Even the most brilliant act of human genius, in the final analysis, is an act of trial and error. “To be an Error and to be Cast out is a part of God’s Design,” wrote the visionary poet William Blake. Error, whether random or deliberate, must become an integral part of any process of creation. Evolution can be thought of as systematic error management.* [Kelly, p 470]

Sometimes a monster can become a brilliant innovation.

# Specifying Rules

## Rule interaction

In any reasonably complex business system, there will be unforeseen interactions between rules. In some cases, there may be an outright contradiction between two rules, resulting in an impasse.

Where rules are imposed by external agencies, there may be conflicts between these agencies, manifested in contradictions between the rules. Some years ago, a friend of mine was faced with conflicting instructions from the Inland Revenue (concerning income tax) and the Customs and Excise (concerning sales value-added tax). Such conflicts can only be resolved by higher authority or political action; the individual may decide to avoid certain otherwise legitimate areas of activity in order to escape being faced with these difficulties.

## Default rules

Some rules simply express a business preference for one condition or outcome over another, where no other rules are in force. These rules are called **defaults**.



Where services and repairs are entered together, services should be booked first, by default [Allen & Frost, p 80]. How should this rule be represented? How could it be implemented?

Default rules can also suffer from interaction effects. Where two rules express conflicting weak preferences, then they may each defer to the other, leaving the business with no preference at all.

Suppose there are three possible outcomes, {A, B, C}, and two rules. Rule 1 prefers outcome A, while Rule 2 prefers outcome B. Usually, only one of the two rules is applicable, so there is a simple default. But under some circumstances, both rules are applicable at the same time. We need some way of composing the two rules to ensure that we get either A or B as the default in these circumstances, and definitely not C.



How can such rule compositions be implemented?

There are some interesting analogies here with voting systems. Voting systems that are supposed to elect the most popular candidate often achieve the opposite: candidate C is elected, although most of the electorate would have preferred A or B. Perhaps we need mechanisms for rule implementation that correspond with transferable voting systems.

# Rationalizing Rules

## Normalize rule structure

Business rules may be derived top-down by an analysis of the goals of the business process, and bottom-up by an analysis of the behaviours of the current system. They may then be rationalized, and reassigned to the components of the future system.

In the ideal rule structure, every rule is traceable from high-level business goals, without redundancy or distortion.

## Compare actual rule structure with ideal rule structure

If we examine the current systems and working practices in a given business area, we can extract the rules that are enforced by them. For example, what are the actual invariants, postconditions and constraints of the system operations? These rules can be expressed as a rule model.

Where the systems are composed of identifiable subsystems or components, we can assemble a rule model for the whole system from the separate rule models of the component.

An analysis of this structure often reveals many problems. Some rules may be enforced in several places, redundantly or inconsistently.

When we compare this rule structure with the business requirements, some further problems are often revealed. Some business rules may not be properly implemented or enforced by the current systems. Conversely, some constraints in the current systems may not be required by the business.

# Implementing Rules

## Embodying rules

Business rules should be embodied in business objects.

*[D'Souza & Wills. p 650]*

Whenever business rules are susceptible to change, exploit packages to separate them out. ... Separating certain rules makes versioning and configuration management easier.

*[D'Souza & Wills. p 551]*

## Rule clustering

If the rule diagram is normalized, each branch of the rule hierarchy can be regarded as a separate cluster of conditional logic.

Each cluster is then implemented in a separate component.

## Rule interaction

Multiple exception conditions may arise, and this is handled as follows. Every time an exception condition is recognized, this creates a new instance of an object belonging to the generic type `EXCEPTION`. (We may of course define subtypes of this generic type, if this is useful.) These objects may collaborate: thus the behaviour of one exception may be influenced by the presence of other exceptions.

## Rule variation



A business works differently for permanent and temporary sales administrators, because of the different business rules involved. The Select Perspective handles this by allowing multiple business actors for one system actor [Allen & Frost, p 61] How else might this rule variation be implemented?

# Uses of Rule Diagram

An understanding of the structure of business rules (as expressed in rule diagrams) can be used in several additional ways.

## System evolution

Many changes to business requirements involve a change to the business rules. These can often be expressed in terms of the addition or removal of some case or exception condition.

Given a component-based system architecture, the ideal way of implementing such changes would be to plug in a new component that handles the new case or exception.

## System testing

Systems can be tested for their adherence to a given set of rules, under a range of circumstances.

## Rule management: enforcement and empowerment

If unnecessary constraints are removed from computer systems, they become more flexible.

If unnecessary constraints are removed from working practices, the people become more empowered.

Rule analysis provides an understanding of the available degrees of freedom in some task or project.

Managers with an intelligent understanding of the rules governing some working area can focus enforcement on the rules that matter to the business, while allowing an appropriate level of autonomy to staff.

## Processes that manipulate rules

In large organizations, some people, especially in central staff functions, may see their role as the maintenance of rules. Thus the personnel department may spend much of its time negotiating rules and guidelines for line management, and monitoring the application of these rules, rather than performing direct personnel operations. The planning department lays down general business rules and operating policies, while the accounts department maintains a fat volume of accounting rules, dictating how costs are to be allocated and distributed between accounts.

Thus the rule itself is created in one part of the organization, and communicated to other parts of the organization. The rule diagram provides a suitable notation for this.



# Beyond Predicate Calculus

## Introduction

All of the rules we have looked at so far can be expressed in first order predicate logic with simple arithmetic.

Some forms of business knowledge, however, demand more complex logical expression.

## Three-valued logic

Distributed systems often require some form of three-valued logic. This is because a question to a remote component is typically capable of three possible answers: “Yes”, “No” and “Please try later”.

For example, we may have a simple order processing application, in which the logic varies depending whether the customer is already on our database or not. Thus we execute a FIND operation on the database. In a “traditional” monolithic non-distributed implementation, there are two possible outcomes: CUSTOMER FOUND and CUSTOMER NOT FOUND. But in an open distributed system, we need to allow for a third outcome: CUSTOMER DATABASE NOT ACCESSIBLE. Three-valued logic provides an elegant way of handling these third outcomes, once the initial unfamiliarity is overcome, particularly when we want to compose complex systems containing many such outcomes.



How can such third outcomes be handled using two-valued logic and exception conditions? What are the drawbacks of this approach?

## Source preference

In many open information or business intelligence systems, there are multiple potential sources for a given item of information. We can then formulate business rules expressing preference for one source over another, because of speed, reliability, security or cost-effectiveness.

## Fuzzy logic

Fuzzy logic is a way of expressing a form of set theory where the boundaries of the sets are not clear-cut.

This is required in many business situations. If a commercial business wants to survey customers, or analyse the behaviour of competitors, it needs some way to identify potential customers and competitors. If a police force wants to investigate suspects for a particular crime, it needs some way to reduce the whole population down to a subset of likely candidates.

But there is no single attribute or Boolean condition that reliably separates customers, or competitors or suspects from the rest of the population. Instead, it may be possible to identify a number of characteristic features. We then focus our attention on those individuals possessing a suspicious number of these characteristics.

*For more discussion of these issues, see my earlier book: Information Modelling (pp 98 ff).*

## Modal logic

Modal logic includes additional operators, such as **necessarily** and **possibly**.

These may be appropriate for modelling higher forms of knowledge.

## References

Philip Boxer & Bernie Cohen. A series of papers describing the rules governing large organizations, presented at CASYS and other conferences, can be found on the Boxer Research Limited website: <http://www.brl.com/>

Catalysis. Desmond D'Souza & Alan Cameron Wills. Objects, Components and Frameworks with UML: The Catalysis Approach. Addison-Wesley, 1999. See also <http://www.catalysis.org/>

Kevin Kelly, Out of Control: The New Biology of Machines. UK edition, Fourth Estate, 1994. See also <http://www.well.com/user/kk/OutOfControl/ch24-a.html>

Imre Lakatos, Proofs and Refutations: The Logic of Mathematical Discovery. Edited by John Worrall & Elie Zahar. Cambridge University Press, 1976.

Select Perspective. Paul Allen & Stuart Frost. Component-Based Development for Enterprise Systems: Applying the Select Perspective. Cambridge University Press & SIGS Books, 1998.

Richard Veryard. Information Modelling: Practical Guidance. Prentice-Hall, 1992.