The IDIOM approach to business rules and the development process

Robert Love

Object Oriented Ltd. • Steinen • CH-6048 Horw • Switzerland • Tel. +41 (0)41 340 5640 • Fax +41 (0)41 340 5641
www.object-oriented.com – info@object-oriented.com

Summary

In a recent book, *Business Rules and Information Systems*, Tony Morgan attributes major shortcomings in today's software development processes to an over-emphasis on programming and a failure to recognise the importance of rigorous requirements specification. In Morgan's opinion we should express requirements in the form of a richly structured business model that supports two kinds of transformation: into natural language understandable by the business owners, and, ultimately, into a machine generated information system. While this second goal is not reachable today, beginning immediately to adopt these techniques will enable us to realise substantial improvements in the development process immediately as well as setting us on the right path for the future.

Business rules are a case in point. They are a slice of the requirements which we can treat in isolation to prove Morgan's agenda: that is, to create a complete and precise rules model which business owners can certify as accurate and from which the rules component of the system can be generated. This approach had not been realised at the time Morgan described it in 2002. IDIOM has since realised it fully. The benefits include flexibility and accuracy in the completed application, and reduced time and cost from the elimination of programming steps.

IDIOM is an independently conceived product for developing business rules, whose approach is strongly aligned with Morgan's. We describe how IDIOM is applied in the rules development process described by Morgan, finding that it more than meets his requirements for a tool effectively supporting rules development wholly under the control of the business owners. We outline typical work-products and activities in business rules development when IDIOM is used, and additionally relate these to the relevant sections of the Zachman framework.

For best results when viewing this document with Adobe Acrobat, please turn on the *Smooth Line Art* option in *Edit* → *Preferences* → *General*.

Introduction

IDIOM is a tool and an approach for the development of business rules by business people. To be successful, businesses need automated systems that accurately and reliably implement the day-to-day decisions that embody their business strategy and on which they depend for competitive advantage. Business decisions are the focal points in the business process where the business acts to realise its goals by applying its business rules. In the IDIOM view, effective management of business rules is a crucial to building automated systems with the characteristics essential for business success: in particular, systems that accurately meet business requirements and adapt quickly and flexibly to changing business conditions and opportunities. Business people, not software developers, are the people who make business rules and understand them best. A fundamental principle of the IDIOM approach is that successful systems are built by providing business people with the means to develop and manage the business rules throughout the development process, and also during the subsequent life of the system.

Similar ideas were developed independently by Tony Morgan in his recent book, *Business Rules and Information Systems: Aligning IT with Business Goals*[1]. As Mr Morgan's two-part title suggests, he too sees focus on business rules as an essential ingredient of information systems that serve the business more effectively. The remainder of this paper is an appreciation of Mr Morgan's book, with a discussion of the substantial points in which the Morgan and IDIOM approaches coincide, and a demonstration of the ways in which IDIOM meets or exceeds Mr Morgan's specifications for a business rules tool of the future.

This paper is intended for anyone interested in software development processes and the growing emphasis in the IT world on business rules. Readers are assumed to be familiar with software development processes: for example, with the activities such as analysis, design and implementation through which an automated system is built, and the typical products of these activities. Prior acquaintance with IDIOM is an advantage but not essential: IDIOM familiarisation materials are readily available from www.idiomsoftware.com.

[1] [Morgan 2002]. References are listed at the end of the paper.

The way we build software

No technical book is any the worse for getting our attention with some initial polemics, and Tony Morgan begins by remarking that many people in the IT world today, along with many outside it, have the perception that organisations' attempts to build information systems are often, and sometimes spectacularly, unsuccessful. He observes that "the levels of performance accepted as normal in IT would be considered utterly appalling by the standards of any other mature industry"[2]. By conservative estimates, 10 percent of an average IT budget is spent on projects that never deliver measurable benefits, and 25 percent on correcting errors in previously delivered software. Worldwide, he estimates, this amounts to $250 billion spent each year on "various kinds of failure".

Reflecting on this situation, he observes: "Our current development culture puts a huge emphasis on programming, and the popular perception is that 90 percent of building an information system depends on the cleverness of the programmers. … The reality is different. Many research projects have shown that the vast majority of software problems originate from specification error, not from the code as such." In other words, multiple attempts over the years to improve programming techniques have been misguided because the problem lies elsewhere, in the way we treat requirements, or, more accurately, in a flawed development process that loses sight of requirements.

In the conventionally accepted software development process, other worrying features are apparent:

- The development process relies heavily on a series of steps in which requirements are successively converted from one form to another by analysts, designers, and developers. The business owners of the software are kept remote from the process because the further the process goes, the more likely it is that the generated artefacts are not in a form that business people can readily understand.
- The multiple steps also introduce many opportunities for misunderstandings and errors, and these are often not detected until late in the process. In fact, "the process contains an implicit acceptance of frailty. It's taken for granted that the code produced will contain errors" and that a significant amount of effort will be expended on the inevitable cycle of fixing these plus further errors introduced whenever the code is changed.
- There is inadequate separation of concerns. Different needs are tangled together in code in a way that makes it impossible to reuse parts of either a completed or incomplete system in a subsequent system. Business logic may be polluted with irrelevant implementation constraints, for example, and business rules are seldom separately identified as such and implemented in a reusable form.

There are, in summary, two serious problems with the conventional development process. First, it has a wrong focus. It has a preoccupation with programming, which inevitably distorts requirements and as an end result produces software in which there is no obvious link to the requirements, and the correspondence between software and requirements is not open to examination or verification. Secondly, it is also very labour-intensive: each step requires manual labour from expensive specialists, with no significant help from automation. The process is consequently slow. Some large projects are so slow that the needs of the organisation change faster than the software can be developed, and the projects are abandoned at huge cost.

[2] Throughout this paper, all quotations in double quotation marks are from [Morgan 2002].

How could it be better?

In Tony Morgan's view, the core problem in the way we build systems today is that requirements are weakly structured and only tenuously linked with the implementation. His vision of a better future – in which he is far from alone[3] – is one in which structured descriptions of the business requirements drive system development. This is a vision in which humans define the system and machines implement it.

> "If we have a complete definition of the functionality required and a complete description of the processing environment, the creation of the software to do a specific job in a specific environment can be reduced to a mindless automated process."

Such an approach is based on a complete and structured description of the business and its requirements. 'Structured' means that it is organised and expressed using techniques which make it logically consistent and allow machines to analyse it. Once such a description is available, two steps follow.

- First a human-readable view of the structured description is generated, which the business owners verify and correct. This view acts as a kind of contract between the business people and the developers.
- Then the equivalent machine-readable view of the same description is generated. This is sent as input, together with a suitably complete definition of the technology that's to be used, to the system generator, which produces specific software components that together realise an appropriate information system. This is a very far from simple step, but it is in principle possible – "no technology barrier stands in the way".

The crucial idea here is that the humans defining the system and the machine implementing it are sharing the *same* description of the business requirements, each using it in the way that best suits them. This makes it impossible for the generated system to diverge from the requirements.

A practical way forward

The goal of machine-generated systems is still a long way from realisation, but it leads us in a critical direction and we can immediately derive significant benefits from beginning to approach it incrementally.

First, we need to recognise the central importance of Morgan's approach for structured requirements. Regardless of the extent to which we can realise automated development, it is of immediate benefit to be able to describe the needs of the business in a complete, accurate, and richly structured form that allows the business needs to be expressed or converted into a form that the business owner can easily understand, and is capable of being checked for coverage and consistency and navigated by automation tools. Even if the software is developed manually, this approach is a valuable goal in itself, and "will allow us to eliminate a large proportion of the problems that plague current software development".

Secondly, we need to seek ways to allow our enhanced business description to *drive* the development process, because this is the way to maximise accuracy and traceability. This implies principally an emphasis on code generation. The tools of the future will refine and extend techniques in which our enhanced business description is converted into generated system components. Mr Morgan predicted in 2002 that tools that do this would soon be available. IDIOM is one of them.

[3] Many writers have proposed software utopias; a recent example with a focus on business rules is found in [Bevington 2004] and its precursor [Bevington 2004].

IDIOM's solution

Business rules are one part of the business requirements. Looking at the role of business rules in today's development process, IDIOM's designers simultaneously perceived the same weaknesses in the business rules microcosm that Tony Morgan detected globally. They noticed, for example, that even more so that other types of requirements, business rules are not effectively separated and managed in the conventional development process: they are inextricably embedded in the developed code in a way that prevents them from being verified by business users or shared with other systems. Business rules also (along with other requirements) pass through multiple steps in the development cycle, being converted into a new form at each step. No step yields an output which the provider of the input can verify, and the process is error-prone, slow, and expensive.

The IDIOM designers' response to these concerns was a tool for business rules development that happens to exactly coincide Tony Morgan's requirements:

- Business rules are captured – by a business expert or a business analyst using IDIOM – in a simple and intuitively clear graphical formula language. This is a complete and rigorous structured specification of the rules.
- IDIOM converts this structured specification into near-natural language for verification by business people.
- When the rules are fully verified and tested, IDIOM also converts the structured specification into a generated system component. Incorporated in the built system, this component encapsulates the rules and is used by other components to gain access to them.
- This entire process encapsulates the rules in a form that makes them easily maintainable and verifiable by their business owners, and easily distributed to and reused by multiple applications.

Focus on business rules

Having identified a general problem and outlined a general approach to tackle it, Tony Morgan focuses in the rest of his book on business rules in particular. This is appropriate because business rules have to date been a neglected part of requirements specifications. As a consequence, they define a field where system builders are in need of practical advice and effective tools, and where improvements in the development process have the greatest potential to deliver substantial benefits. These are, of course, exactly the reasons that IDIOM was developed. In the rest of this paper we look in more detail at Mr Morgan's recommendations and techniques, focussing on those parts where IDIOM realises and extends his approach.

The business model – the context for modelling business rules

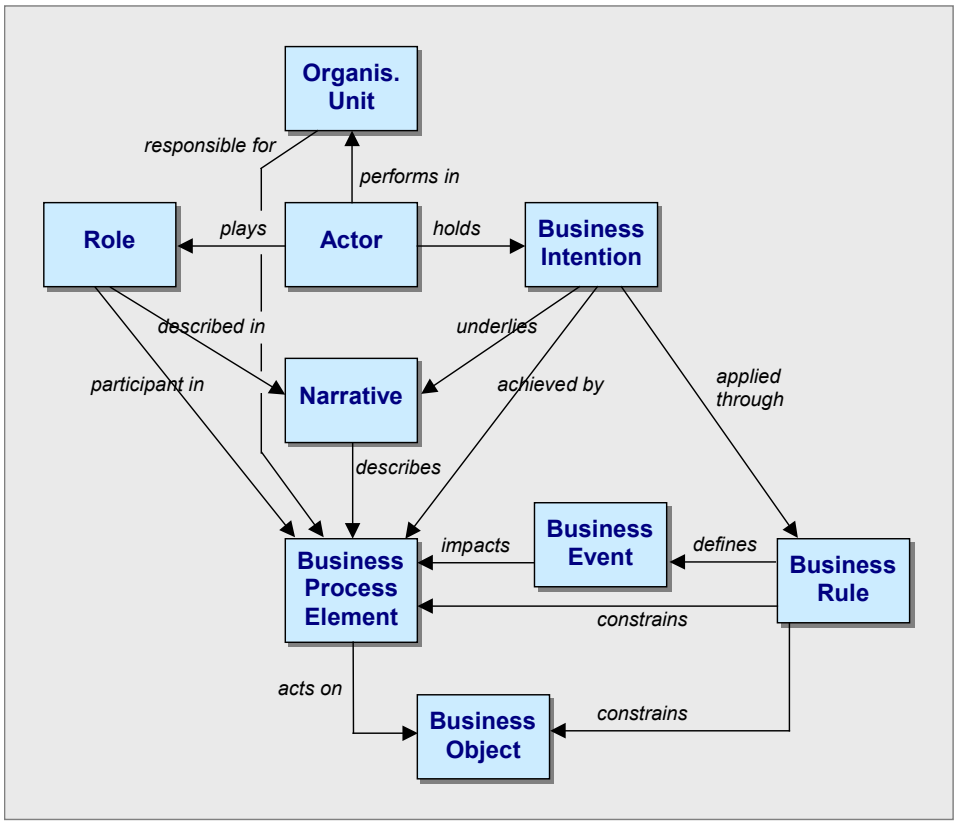
In this section we briefly survey the business model described by Tony Morgan. Business modelling is the bigger picture which we need to properly understand the role of business rules.

The business model

The richly structured requirements specification envisaged in the previous section is, technically, a *business model*. Morgan describes a business model as a "multifaceted information structure that captures business requirements. ... It covers several interlocking viewpoints, each of which highlights a particular aspect of 'how we do business'". One of its most important characteristics is that it should be built entirely from business terminology and remain completely divorced from technology concerns.

Business rules are one essential element of a business model. There are some other elements that usually essential, but the complete set of elements and techniques is expected to vary from one organisation to another, and to be precisely specified in the organisation's *business architecture*. The purpose of a business architecture is to define the vocabulary and syntax – the conventions and methods – of the organisation's business modelling. A particular business model can be thought of as an 'instance' of the architecture. The sample business architecture described by Morgan includes the following elements[4]:

Figure 1: A sample business architecture (adapted from Morgan)



One of things meant by "richly structured" is that all the elements of such a scheme are expected to have their own internal structuring: using, for example, hierarchical decompositions, flow or sequence diagrams, UML-style modelling, and so on, according to the organisation's methodological preferences.

[4] 'Element' is not always the best word, especially since it is reused at a different level in the term 'business process element'. Where it seems clearer we will sometimes refer to *sections* rather than elements of the business model.

'Business objects' – or fact model

What Morgan calls a business model coincides reasonably closely with the second layer of the Zachman framework,[5] after which it is presumably named. Some elements of a Morganesque architecture can be correlated with Zachman cells in this layer, though Morgan's conception is more detailed.

The architectural element called *business object* in Figure 1 corresponds with the cell in Zachman's *Data* column known as the *semantic model* or *fact model*. Following Morgan, we will use the terms 'fact model' and 'business objects' more or less interchangeably, according to context. The attraction of the term 'business objects' is that it better conveys that the purpose of this section of the business model is to define the real things or concepts that participate in or are dealt with by the business, such as customers, orders, contracts, penalties, inducements, and so on. This is the sense intended in the figure, where such alternatives as 'business fact' or 'semantic object' do not obviously commend themselves. In other contexts – for example, when describing the controlled vocabulary necessary for the precise expression of business rules – the term 'fact model' better describes the methodological intent.

In current development practice the fact model is often simply a glossary. In using the term 'business objects' Morgan indicates that it should be more than this: his intent is to enrich the structure of the conventional glossary or fact model with more rigorous definitional techniques – for example, by deploying a notation such as UML in order to model classes, attributes, associations, and state.

The notion of 'business objects' introduced here is not be confused with the logical data model or physical data model at levels 3 and 4 of Zachman respectively. These other artefacts will undoubtedly represent some of the same objects, possibly using very similar techniques, but they do so at different levels of detail, from different perspectives, and for different purposes.

Other elements of the business model

Here we briefly survey other elements of Morgan's sample business model which have a significant relationship with business rules. These are narratives, business processes (or business process elements), and business intentions.

Narratives

> "A business narrative is a description of a fragment of business life." The most universally adopted kind of narrative is, though it takes a variety of forms, the use case, and we will use this term more or less interchangeably in subsequent discussion. What all narratives have in common is that they describe the way the business is expected to operate in a given situation. They do this by describing the interactions of an agent with 'the system', and in doing so define the scope of the system: "they describe a boundary around the system and interactions taking place across that boundary".
>
> Narratives deal with things and concepts in the business world, using the vocabulary precisely defined in the fact model. They describe workflow – i.e. elements of business process, – in the course of which they arrive at critical points where *business decisions* are made, requiring the existence of business rules. We discuss business decisions and business rules in more detail later; here we merely note that narratives, or use cases, are usually an important starting point for the discovery of business rules.

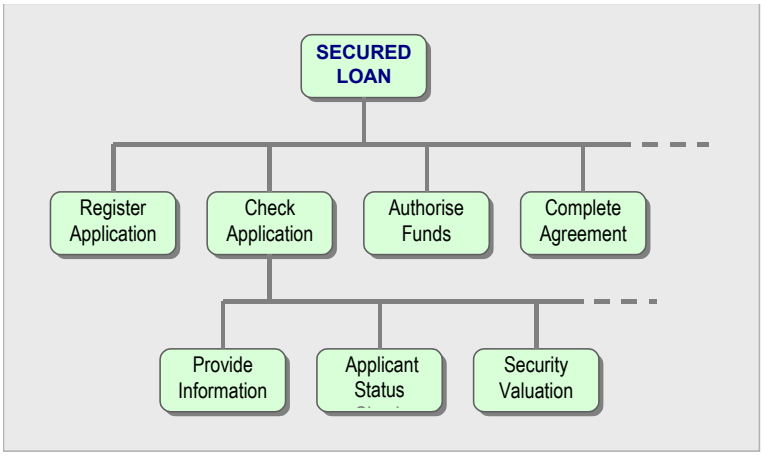
[5] A brief explanation of the Zachman framework is given as an Appendix.

Business processes and their elements

"A business process is a sequence of activities aimed at producing something of value to the business." Typically a business process has at least one clearly identifiable start and end point, has an objective or purpose, is enacted over an interval of time that may be short or long, often spans organisational units, and handles 'things' (physical objects or information) that may be transformed or augmented and handed on. Business processes can also be broken down into smaller elements, which may participate in more than one process.

Morgan notes that business processes are neglected by some methodologists (particularly proponents of UML) but describes two techniques for structured breakdown that are in common use: hierarchical decomposition, and flow diagrams. A typical hierarchical decomposition, using Morgan's example of a bank loan application, is pictured below. Whatever the modelling approach, it is necessary to identify and depict the *elementary business processes* which make up the lowest useful level of decomposition. Criteria for identifying these elementary units vary, but one common guideline is "one person, one place, one time". It is the job of use cases to specify the content of each elementary business process in detail.

Figure 2: A sample process decomposition (after Morgan)



Business intentions

This section of the business model falls roughly in the *Motivation* or *Why* column of the Zachman framework[6]. It describes business goals and objectives and core business values. It is the "underpinning that should enable us to answer questions about our system" like 'Why do we need to include this feature?' or 'Why do we constrain values in this way?'. Morgan suggests one possible structuring technique which involves the modelling of layers of intentions, enablers, and risks.

There is a connection between intentions and with business rules which is important when business rules are being formulated. Business intentions provide the motivation for business rules, or, as Figure 1 sees it, business rules are the means of applying the intentions. Morgan does not suggest that it is necessary to model these connections explicitly, and we don't at present see this as necessary to the rules development process.

[6] At level 2, of course, since the whole of the business model we are discussing lies at this level. This Zachman cell is variously described as a *business plan* or *policy charter*.

Business rules

Having made a brief survey of what else a business model contains, we arrive finally at business rules. Business rules gravitate towards the end of business model development, because they are logically dependent on most other elements of the model. In Tony Morgan's words, the importance of rules in relation to the rest of the business architecture is that they provide an excellent means of encapsulating knowledge about the intentions of the business which can then be applied to define and control other parts of the system. To be developed effectively, "rules can't stand in isolation but need to be grounded in a rich representation that captures the many facets of the business that are needed to provide a balanced picture."

The high level purpose of business rules is to realise intentions of the business which might be quite generally expressed, such as reducing risk (in the case of loans or insurance policies, for example), managing customer relationships, or controlling workflow and approval processes. The rules themselves can have various types and purposes, such as:[7]

Purpose	***Examples***
Entity relationships	Relationships that must be enforced between business entities (e.g. *an Order must have exactly one Customer*), and specific conditions for these (e.g. *unless the Order is of type X, in which case...*).
Supporting business decisions	Recognising business scenarios and conditions, allowing standardised, predictable, well-managed decisions to be made.
Computation	For example, premiums, interest rates, finance allocations, workflow steps, approval status, etc.
Validation and notification	Checks on correctness of values, descriptions of incorrect states, notification of automatic corrections made, etc.

Rules, then, represent constraints that define conditions that the business requires to exist, or calculations that must be applied in particular business contexts. Morgan emphasises:

> "Business rules are not descriptions of a process or processing. ... They define *what* must be the case rather than *how* it comes to be."

While Morgan is here referring to processing in its most general sense, this is a correct description of the relationship between rules and process in the business model. Use cases describe business process, by narrating how exactly how the business is carried out. Business process elements also describe business process, by analysing and decomposing its structure and tasks. Use cases describe process elements, and rules are applied in the context of use cases to impose conditions and specify calculations.[8]

Lastly, there is the question of what is the appropriate form for business rules to take in the context of the kind of business model we would like to construct. In keeping with our general principle of verifiability, business rule statements must be available in a form that the business owner can immediately accept as valid or reject as invalid. Morgan's view is that normal business language is entirely adequate for expressing business rules (it has managed quite well for centuries so far), and that the form shown to the business owner can be the original form rather than a re-presentation of it: "esoteric notations, technical languages, and so on, are not essential." There must of course also be an alternative form of the rules "better suited for automation". Morgan is driven here by a lack of tools

[7] Adapted from [Morgan 2002], section 3.1.2.
[8] The term 'process' is a fairly rich source of confusion. The execution order of rules, discussed later on page 20, is sometimes called their 'processing order', but this turns out to be a detail of implementation beneath the notice of the business model.

to compromise his own general principle, with the awkward proposal that normal language is the primary form of the rules from which the automation-oriented form is somehow generated. As is evident from what we said earlier, IDIOM is now at hand to rescue Morgan's original agenda, which is preferable. With IDIOM the primary representation of rules is a fully and precisely modelled (but far from esoteric) graphical representation, and the automatically generated near-natural language is a secondary representation (the other being the generated rules component). The mechanics of this approach are described in more detail in the next part of this paper.

Business decisions

The concept that IDIOM contributes to rules modelling is the *business decision*. As defined in the IDIOM lexicon, a business decision determines the action to be taken in a particular 'business situation'. It represents a point in the business process where we need to say exactly how the organisation – through its information system, if this is an automated decision – will behave, in response to the particular circumstances. The details of this behaviour are spelt out by business rules. In a use case for a bank loan application, for example, the typical business decision to be made is 'Should this loan application be accepted, and if so at what rate and with what conditions?'. Or in a medical insurance claim: 'What items in this claim should be paid, at what level, or with what explanation if rejected?'.

Business decisions in this sense do not immediately seem to require us to add a new section to Morgan's business architecture, because they are already present in a slightly different form. Looking at the example process decomposition in Figure 2, it's clear that business decisions coincide with Morgan's elementary units of business process. 'Applicant status check' and 'Security valuation', for example, are pieces of process whose purpose is to arrive at a business decision. The underlying business decision is the goal of the business process element, and it is noticeable that the business process element usually takes its name from the decision it makes.

Business decisions, then, more or less coincide with a concept already present in the business model. However, as we integrate IDIOM into the development process, we will find that they have a useful role to play in defining how business rules are invoked. Briefly stated, a use case describes a piece of process whose purpose is, usually, to make a business decision. Making the decision consists of applying of a particular group of business rules in the particular context of the use case. The business decision plays an important definitional role: it describes the purpose of applying the rules, which rules should be applied, and the facts (or data, assembled by the use case) on which they operate.

These relationships are reflected in Figure 1, if we regard a business decision as being roughly synonymous with a Business Process Element. A use case 'describes', or provides context for, the decision; business rules constrain the decision; and business intentions are realised by the decision.

Developing business rules – activities and artefacts

In the previous section we have described business rules as one of the essential elements of a business model which accurately and completely captures the business requirements. It is now reasonable to ask, what kind of development process is needed to develop such a model? What are the development activities, in what sequence are they undertaken, and what are their products?

It is outside the scope of this paper (and Morgan's book) to describe the entire development process. The different elements contained in Morgan's business model all merit separate discussion of the benefits which they individually bring to the development process, but our purpose here is to focus on business rules. We will outline the overall development process sufficiently to explain where business decisions and business rules fit it, and then trace the specific thread of rules development in more detail. The emphasis is on IDIOM's contribution to rules development.

One conclusion from this discussion will be that a focus on business rules, particularly with support from a tool like IDIOM, can be introduced to an existing development process with minimal upheaval. It should not be necessary to completely re-engineer the development process in order to introduce and benefit from a more rules-centric approach.

The 'typical' development process

Every organisation evolves its own development process. As a basis for discussion we'll assume that this process typically proceeds through the activities of analysis, design, and implementation. These can succeed each other either iteratively (the spiral model) or sequentially (the waterfall model), but that is not an issue with any significant bearing on our discussion.

The business model that we have discussed so far describes pure business requirements and is a product of the activity we'll call 'analysis' for brevity, but which is understood to include the capture, analysis, and notation of requirements. This activity is carried out by the business owners and business analysts. As we have said earlier, this activity and its results fit roughly within the second layer of the Zachman framework.

Design activity produces such artefacts as the logical and physical models described in the third and fourth Zachman layers, and implementation activity, which is largely programming, produces the technological artefacts of the fifth layer.

As a general rule, the Zachman layers describe phases of development of a piece or component of the system that must occur in proper sequence. Iterative life cycles obey this rule too: they work by completing sections of the system at each pass. *Within* a Zachman layer, it is common, even inevitable, for the constituents elements of major work products to be developed in parallel.

Working on the business model

Morgan says nothing about the order in which he expects the various sections of a business model (apart from business rules) to be produced. The implication is that this is not something that can be usefully specified in advance, or is perhaps too obvious to need mentioning. At the risk of stating the obvious, we'll state what we regard as typical steps leading up to the development of the business model.

First, before the development process proper can begin, there is some kind of project inception phase whose purpose is to make a preliminary determination of scope and to plan the development activities. A functional breakdown of the system is usually the only sensible basis for this, meaning that an initial analysis of the business process elements is produced, leading to a list of principal use cases which can be used as a framework for planning.

After this development process gets under way, beginning with requirements capture through interviews and other forms of information gathering. The information collected is recorded, after analysis, in the individual model sections for actors, roles, organisational units, intentions, and so on, as well as the use cases themselves. The sections are typically developed piecemeal and in parallel, according to the order in which facts are discovered and need to be written down. There is nonetheless a logical order, or a set of logical dependencies, that must be observed by finished sections before they can be regarded as complete and well-formed. In the sample business architecture illustrated in Figure 1, for example, actors, roles, and business process elements are prerequisites for use cases, and use cases and a fact model are prerequisites for business rules.[9]

Where do business rules fit in?

The development process we are picturing is a progressive elaboration of the various sections of the business model in an order driven both by the logical relationships between them and the fortuitous order in which facts are discovered. The project planners need to organise the work, and a common planning basis is to tackle the use cases in prioritised order. Use cases tend to be the backbone of the plan because, as we mentioned earlier, they are a vehicle through which the detailed scope of the system is discovered, and this has repercussions throughout the business model and for the project as a whole.

Through this process there is a thread concerned with business rules which we trace in the following sections. In the first two of these we focus on business decisions, which are discovered in individual use cases, and imply sets of rules. After that, we look at the logically separate and subsequent development of the rules themselves.

Locating and recording the business decisions

In our business model, the job of use cases is to describe (one or more) elementary pieces of business process, and each elementary business process revolves around the making of one or more specific business decisions. The job of a business decision is to realise some particular intentions of the business by orchestrating the application of a set of business rules to data assembled in the use case.

What this means in practical terms is that as a use case is developed the business decision (or decisions) that it makes are identified. A common pattern is that the first part of use case is concerned with assembling the data needed by the decision (through various interactions of actors and the system), and the second part executes the decision, using the collected data as input and receiving as the result of the decision some significant business output (such as a price or status or instruction, or a combination of these). The analytical task here is to identify the name and purpose of the decision, and as far as possible the pieces of data that are involved as inputs and outputs.

Given that this step is embedded in development of a use case, the first priority is to properly understand the operation of the business and the nature of the business decisions involved. It can be a distraction at this stage to attempt to capture the underlying details of the decisions, which are the business rules. It is usually recommended, particularly when the rules are significantly complex, to plan for a separate rules discovery process at a later stage: Tony Morgan's case study for rules discovery[10] is a case in point. Good reasons for this approach are that business rules may require different business specialists from those that contributed the body of the use cases, and that rules can turn out to be shared by a number of use cases, in which case it is obviously better to have

[9] Note that the arrows in Figure 1 run in the direction of the verbs with which they are labelled. The direction of the prerequisite relationship may or may not be the same as the arrow, depending on the meaning of the verb. The intention of the verbs is to suggest rich relationships between the model elements, in which prerequisite-ness or dependency would be just one constituent (and not necessarily in every case).

[10] [Morgan 2002], section 4.4.

found these various use cases before attempting to formulate their rules.

The decision inventory

Business decisions, as we discussed earlier on page 11, in virtue of their role as the genesis and superstructure of business rules, have a claim to being regarded as a business model section in their own right. Particularly when IDIOM is used, business decisions are significant analytical entities which provide a framework for rules development and are traceable throughout all subsequent phases of the development process. We will call the artefact in which they are described the *decision inventory*.

The purpose of the decision inventory is to provide a place, outside of the use cases, to collect information about decisions which directly supports development of the business rules. Initially, when a decision is first identified, this is likely to include the decision's name, a brief description of the decision's business purpose and effect, the name of the use case making the decision, and a list of the data values that, from analysis of the use case, appear to be required as input and expected as output. Inputs and outputs, in particular, are expected to be incomplete at this stage, pending full discovery later during the rules development process. Information given for each value is likely to include a name or definition, probably authorised by the fact model, plus data type, constraints, etc.

Other facts about the decisions could be recorded: for example, references to business intentions, similarities between decisions (implying shared rules), references to government regulations that rules must implement, and any advance information about rules' implementation that happens to come up in interviews and needs to be captured somewhere. The intent is not to copy any information in the use case, but to organise the information in the way that best supports rules development. The use cases will probably not be completely supplanted as source material for rules development, nor is this a goal.

The decision inventory is an artefact intended to support rules development but not to contain the final expression of the developed rules (for that a specially designed tool such as IDIOM is required). Learning from Tony Morgan, we should ask what form it is likely to take: will a simple word-processing document suffice, or should some elements of it – for example, the data content of each decision, or the links between use cases and decisions and between decisions and rules – be more precisely modelled? The answer depends on the tool used for expressing the rules. If the tool is comprehensive enough, it will model all these elements – in a rules repository, for example, – and the inventory will revert to the status of a secondary source, no longer containing any information that is vital for production of the information system and is not modelled elsewhere.[11]

The steps for rules development

Business rules tend to be one of the last sections of the business model to be developed because of their logical dependency on other sections. As we mentioned earlier, use cases and the decision inventory are essential prerequisites. The fact model is also a prerequisite – or co-requisite – of rules development: the terms referred to in the rules must have a consistent and well-defined interpretation, and it is the job of the fact model to provide this. As also happens elsewhere in the development process, 'prerequisites' are not necessarily definitively complete at the outset: we expect to correct and extend them in the light of discoveries made during rules analysis.

There is a further reason why rules development should gravitate towards the end of business model development. In many businesses, competitive advantage comes from being able to change business rules quickly and easily after the system has been put into production. The business demands not just a business architecture in which rules are modelled effectively, but a system

[11] IDIOM models most of the elements mentioned here. It models decision content (as schemas) and the linkages between decisions and rules. It is not cognisant of use cases and does not enable decisions or rules to be traced back to use cases.

architecture, tools, and general development approach that allow rule changes originated by business owners to rapidly pass through the development steps and (via code generation) into the running system. The result of such a capability, demonstrated by IDIOM, is that not all rules development has to be finished before design and build activities can proceed; it can be done in parallel, with corresponding acceleration of the project schedule. As is shown graphically later in Figure 10, we think of business rules development as beginning towards the end of business model development, and continuing thereafter through the remaining development steps and through the life of the deployed system.

The development of business rules is a process with three steps: discovery, definition, and testing. Tony Morgan has chapters on each which provide excellent supporting material. In the following sections we review these steps and describe how they are carried out under an IDIOM approach.

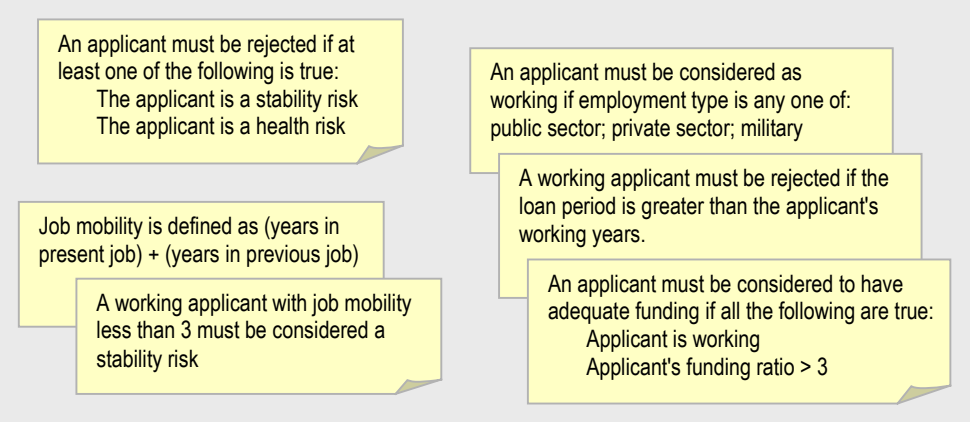
Rules discovery

At a high level, the task of discovering business is not a lot different from the way the rest of the requirements are captured. It involves collecting and analysing information from various sources, including documents and business people, and presenting the results in the form required by the business architecture for input into the next step of the process. The details are, of course, different, and it is not our purpose to describe them here. Tony Morgan provides a comprehensive chapter on this subject, covering information sources, indicators for rules, techniques for analysing documents, interactive sessions, and workshop techniques. He includes a valuable case study dealing with a loan approval example, on which the illustrations in this section are based.

Seen from an IDIOM perspective, rules discovery is the step which brings rules analysis far enough for IDIOM to be able to take over the remaining work of precisely defining and testing the rules. Our interest in the rules discovery step is therefore mainly limited to describing its results. The goal of rule discovery is to find and specify all rules that underlie each business decision identified in the decision inventory. The results consist of a decent first attempt at the natural language texts of all rules underlying the decision, plus a tabulation of the input data on which the rules operate and the output data finally returned by the decision. The data tabulation belongs in the decision inventory; it is a completion of the rough list that was assembled during use case analysis. The rule texts can be in whatever form it was convenient to produce during the analysis process: they are a temporary and incomplete realisation that will soon be converted into a precise and permanent form.

Typical rule texts might look something like the following:

Figure 3: Sample rule texts produced during rules discovery



The concepts and objects, or *facts*, that the rule texts refer to are of two types. Some, like the datum 'job mobility' are *transient* results produced by the rules themselves, having definitional value to the business, but existing only briefly in

the operational information system. These may disappear when rules execution is finished, or may be returned as part of the result of the decision for further use in the business process. By contrast, other facts are part of the *persistent* records of the business, likely to found in a database, for example. Rules refer to these using the terminology and definitions established in the fact model, and in later phases of system development we expect to appear in artefacts such as the logical and physical data models. The terms 'applicant' and 'employment type' above look as if they will turn out to have some persistent quality: an Applicant is probably modelled in the fact model as part of hierarchy of Person objects, possible with a close similarity to a Customer. (Temporary facts can also be described in the fact model, if it's desired to use the fact model as a complete glossary of business terms; but if the rule is sufficiently specific and particularly if the fact is not exported from rules execution, then a duplicate definition in the fact model is likely to be more of a maintenance overhead than a help.)

The sample rules shown above are a few from a considerably larger set required by the loan approval decision. The following tabulation is likewise only a sample of some of the input and output data that the complete decision might be expected to require:

Figure 4: Sample input and output data required by the rules belonging to a decision

	Datum	*Possible Values*
INPUT	Present age in years	Integer
	Marital status	Single; married; divorced; widowed; other
	Number of children	Integer
	Residence type	Owner occupier; living with parents; tenant; other
	Years at current address	Integer
	Years at previous address	Integer
	Health	Good; poor; unknown
	Smoking habit	Heavy; light; non
	Average weekly income	Integer
	Average hours worked per week	Integer
	Average weekly outgoings	Integer
	Employment type	Public sector; private sector; military; homemaker; student; unemployed; other
	Years in current job	Integer
	Years in previous job	Integer
	Amount of loan requested	Integer
OUTPUT	Amount of loan approved	Integer
	Loan interest rate	Decimal
	Approval status	Approved; Rejected; Referred
	Approval reason	String
	Next processing step	Contract; Supervisor; Resubmit

Rules definition

Rules definition is the task in which we convert the results of rule discovery into a form that allows us to ensure:

- ❒ The rules are unambiguous, precise, and complete.
- ❒ The rules are correct, because they can be verified by the business owner and fully tested.
- ❒ The rules can be converted by automated means into a component that can be directly deployed into the built system, obviating the need for any further human transformations of the rules.
- ❒ The entire rules development process is carried out by business people and under the control of the business owner.

At the time Tony Morgan was writing his book this agenda was not achievable. There was no rules language in existence that enabled all these criteria to be met. Natural language is understandable by the business owner, but impossible to make precise and error-free. The cautionary example below (Figure 5) shows a natural-language attempt composed with a word-processor: because the tool provides no intrinsic support for a precise expression of rules, much effort is expended trying to work round this deficiency with manual formatting, and the results are laborious without achieving the desired mathematical precision (see clauses 1.d, 1.e.iv and 3.b, for example). On the other hand, various formal or technical languages have been proposed, but these are rejected because none of them are understandable by business people. Morgan's conclusion at the time was that current technology was not adequate, and that achievement of the above goals was a long-term objective for the future.

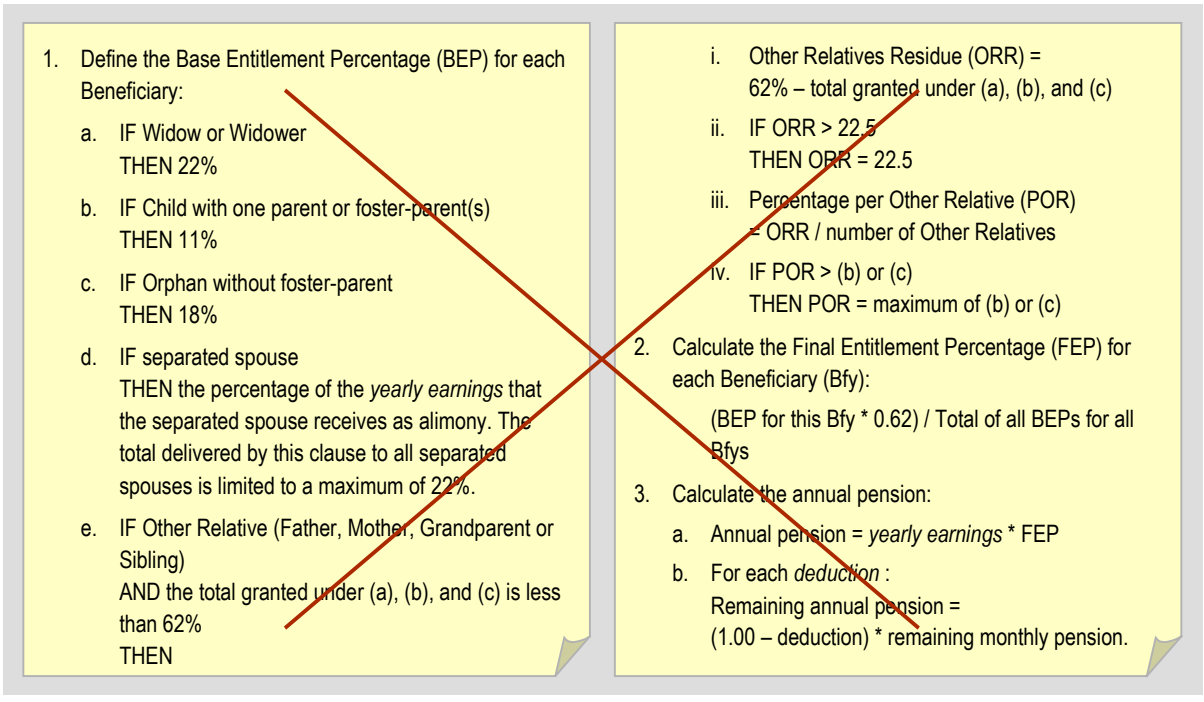


Figure 5: Trying to express rules precisely with a word-processor is time-consuming and unsatisfactory

IDIOM achieves Tony Morgan's long-term objective. It supports the complete rules definition process with an approach that satisfies all the above criteria. It provides an intuitively clear graphical language for expressing rules precisely, plus facilities for testing the rules fully, for converting them into near-natural language for review, and for deploying them in the form of generated program components.

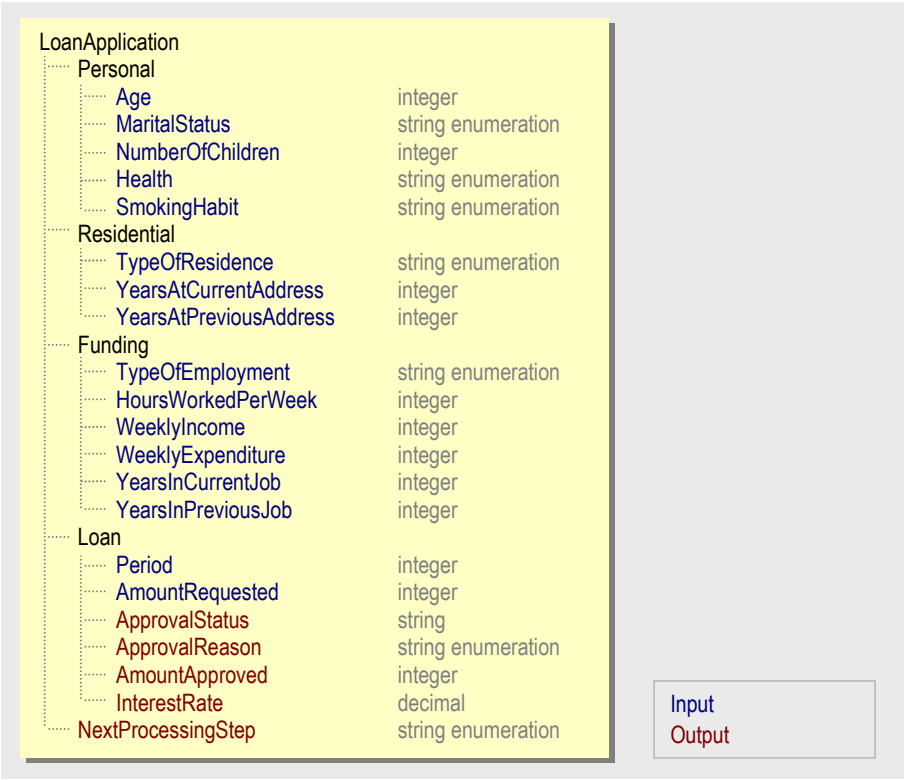
The IDIOM graphical formula language is the base technology on which these features rest. Technically it is a functional programming language. Functional programming languages – Microsoft Excel is a well-known example – are recognised as well suited to non-technical users, because they allow potentially complex expressions to be correctly constructed with simple means in an environment that eliminate many common problems and errors that arise from 'real programming'. With IDIOM, there is no programming in the usual sense: rules (or *formulas*) are assembled by selecting and dragging, which ensures that most kinds of invalid expression simply cannot be constructed. Data types cannot be wrongly combined, for example, and incomplete rules cannot be deployed. Satisfying the grammar of the formula language automatically ensures that rules have been completely and unambiguously expressed.

The rules definition process with IDIOM centres round a rules repository in which analysts recapture the results of rule discovery – business decisions and business rules – in precise and final form. The main elements of this activity are briefly described under the next four sub-headings.

Describing the data content of business decisions

Business rules act on data. Before rules can be formulated some kind of execution environment for them needs to be defined. Business decisions serve this purpose. The complete set of input and output data for a decision are organised into a formally defined data structure called a *schema*. Once the schema has been created, rules belonging to the decision have the complete vocabulary of facts on which they need to operate.

Figure 6: Schematic representation of a decision schema



IDIOM schemas are developed in their primary form externally to the rules repository as XML schemas[12], and then copied into the rules repository for rules development. This is because they have a wider role in the system architecture: they specify the public interface of the rules

[12] More precisely, text files whose contents conform to the syntax of the W3C XMLSchema specification.

subsystem which other system modules must use to invoke the rules. In their XML form the schemas are an artefact of design, not analysis, and are produced by the system's designers or architects, who are technical specialists. They represent a small part of system design that is done early – i.e. in the last stages of requirement analysis – to support rules development.[13]

Expressing rules as IDIOM formulas

Analysts express rules in IDIOM by recasting the natural-language business rule drafts as formulas. Formulas are functions that operate on inputs that are either data values from a schema, or the results of other formulas, or values extracted from rule-defined tables. Formula builders are free to factor the formulas in any way that seems appropriate: a complex rule may be factored into several formulas, particularly if some of resulting pieces are reusable. The 'transient facts' that we mentioned earlier – significant intermediate results generated by rules for use by other rules – are of course expressed as formulas intended for use by other formulas.

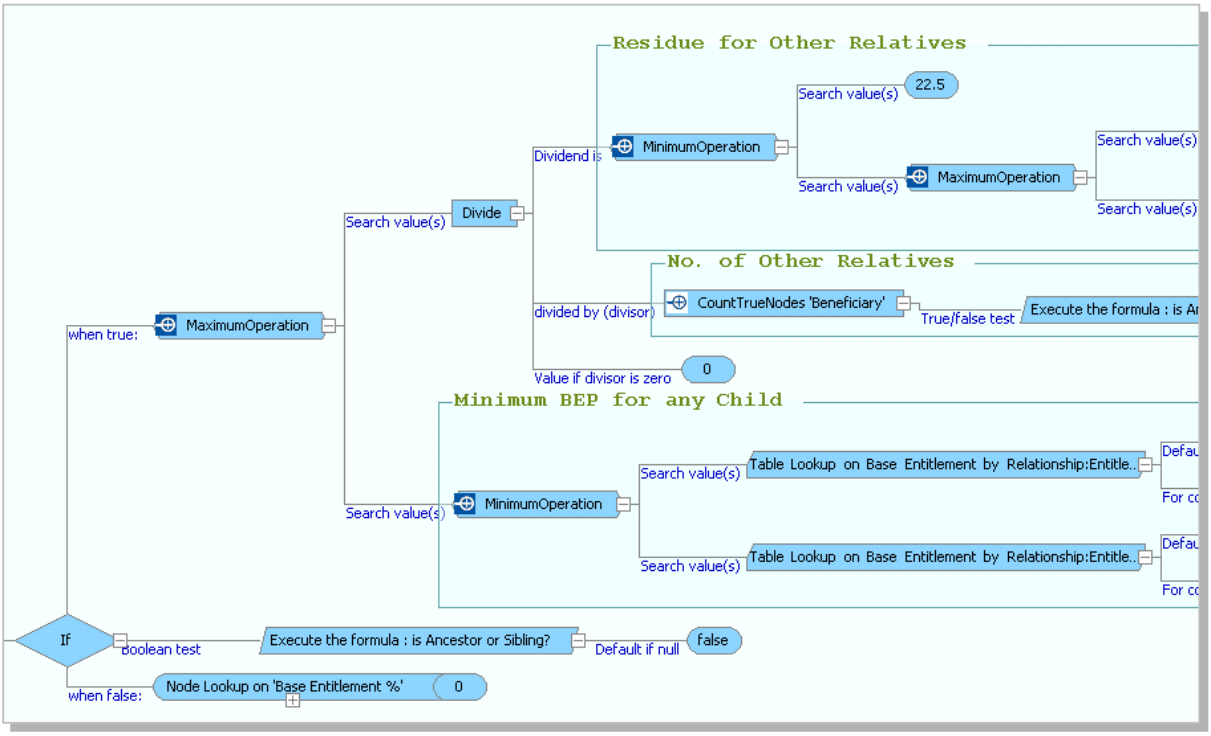


Figure 7: Extract from an IDIOM formula (graphical view)

The work of recasting the draft rule texts into precise formulas naturally exposes ambiguities and gaps that were not evident in the drafts. Business experts and analysts work together to resolve these flaws and bring the rules them to a final correct form. Corrections may of course have repercussions for other work-products: use cases, decision inventory, fact model, and schemas can all potentially be involved.

DRAFT

[13] The sequence of artefacts and activities in the entire development process is summarised in Figure 10 and discussed in more detail in the Appendix on the Zachman framework.

Rule execution order

A business decision specifies, in addition to input and output data, a group of business rules. Business rules are declarative in nature. This means (as should be evident from the examples in Figure 3) that they independently declare desired states of affairs. The business requirement is just that all the rules in the decision be correctly and consistently applied: no *order* of execution is specified. True, there are dependencies between rules where one rule establishes a fact used by another: but this does not mean temporal ordering; it simply says more about what "correct and consistent" means.[14]

At execution time, some order or other must occur. In rules systems that use various kinds of inference engine, the order of rule execution may not be readily predictable and may not be under the control of rules developers. With IDIOM (whose rules execution engine is not an inference engine), an appropriate order of execution is specified during rules development, and always applied at execution time. Deterministic rules execution is essential to IDIOM's approach for rules development by business users. It allows complex rules sets to be built by simple means, and is particularly important in enabling rules authors to develop and debug their own tests.

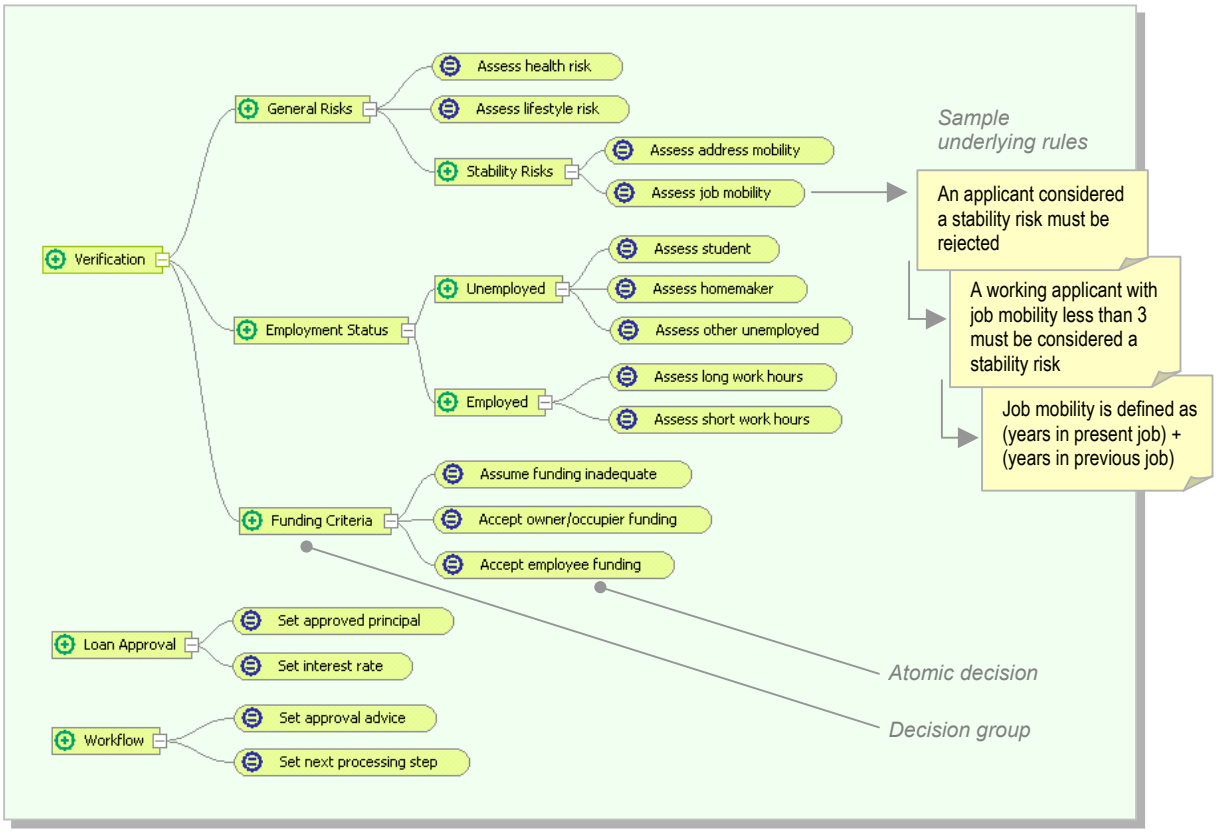


Figure 8: In IDIOM, hierarchical groupings can be created to help organise rule sets. The leaf level shapes (blue icons) stand for the individual rules, arranged in execution order (top to bottom).

Rule execution order is specified in IDIOM using the graphical tool illustrated in Figure 8. Rules in the decision are arranged in a hierarchical structure created out of linked groups. Square shapes with green icons are groups, and rounded shapes with blue icons represent rules (or, to be

[14] Material relevant to this section occurs in [Morgan 2002], section 5.5.2 and figure 5-6.

precise, root-level rules, which in general lead to the execution of other dependent rules). The figure shows a possible way (one of many) of organising the 30-odd rules in Morgan's loan application case study (a few of which were quoted in Figure 3).

The hierarchical organisation is multi-purpose. The desired rule execution order is a straightforward traversal of the leaf nodes (shapes with blue icons, in top-to-bottom order). The remaining nodes (green icons) specify no rules themselves, but form groups containing rules and other groups: they allow labelling and classification of rules for management purposes, and they are used as control structures during execution.[15]

Technically, the kind of specification being done here is procedural programming, conducted by simple graphical means. The IDIOM rules developer transforms declarative anarchy at the requirements level into an precise structure in which rules execute in predictable and correct order. Interdependencies between rules, for example, must be taken into account, perhaps by letting each rule simply call its dependants whenever necessary, or perhaps by strategies for generating and storing intermediate results in the appropriate order.

Lastly, it is worth re-emphasising that because rules are declarative, the rule execution order we specify with IDIOM is a implementation contrivance (with definite practical advantages) and not a reflection of business requirements. With its hierarchical representation it is sometimes mistaken for a 'process decomposition', but in fact it has no significance at a business level and nothing to do with the sections of the business model that analyse 'business process'.

Further explanation of Figure 8

The figure illustrates the IDIOM *scope pane*. A scope corresponds roughly to a business decision. In IDIOM terminology the shapes with green icons are called *decision groups* and the shapes with blue icons are called *decisions*. In IDIOM a 'decision' is an atomic element in what this paper calls a business decision: it corresponds to one value computed by rules for return to the business process. For example, the last four atomic decisions in the figure compute the AmountApproved, InterestRate, ApprovalReason, and NextProcessingStep output fields shown in Figure 6. Multiple atomic decisions may be associated with the same output value: in the large Verification group in the top part of the diagram, all atomic decisions are concerned with calculated the ApprovalStatus: whether the loan is approved, rejected, or referred.

Broadly speaking, each atomic decision is a place where a rule (in IDIOM, a formula) is executed which returns the output value associated with the decision. The rule may call other supporting rules. The figure illustrates this for one case: the atomic decision 'Assess job mobility' executes a series of rules that calculate and assess the transient fact called 'job mobility'. The figure illustrates this with three of the sample rule texts that appeared in Figure 3; in reality more rules than this are involved, and in IDIOM they are expressed precisely using formulas.

Testing and verifying formulas

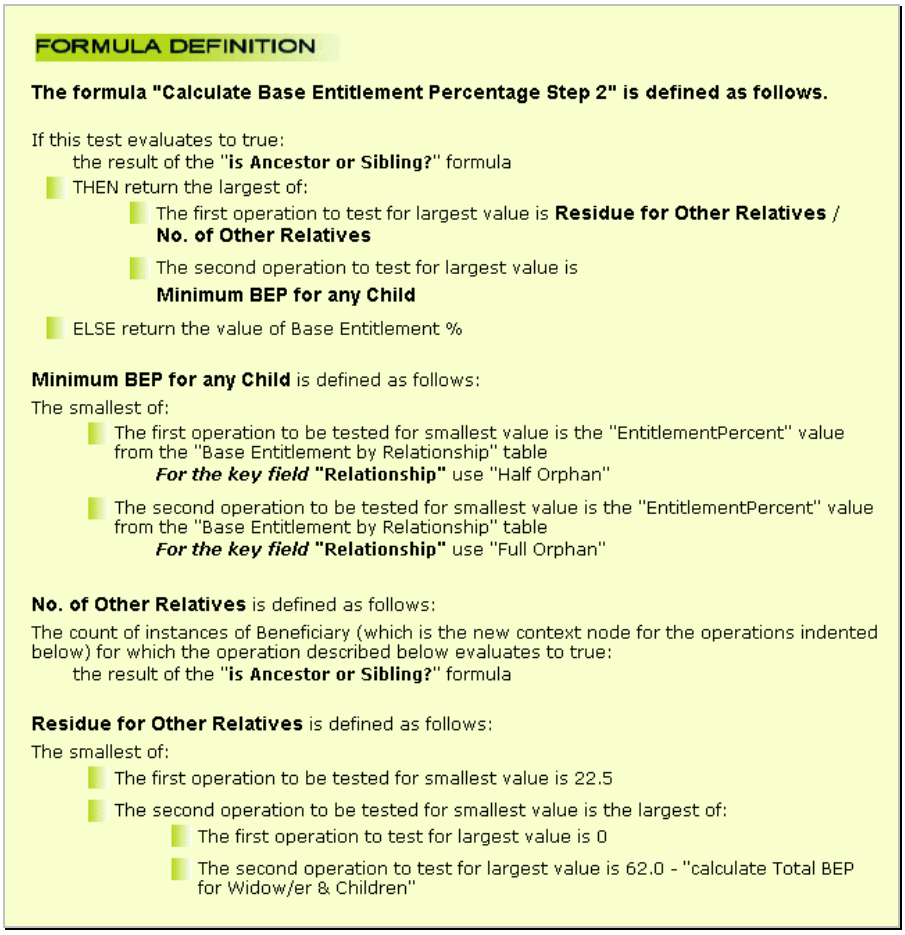
Testing is an integral part of developing rules. In the IDIOM testing approach, described in more detail in the next section, testing of the rules to be done at the same time and by the same people as develop the rules. This goals of this approach are to produce correct rules more quickly and cheaply, first by removing reliance on programmers, and secondly by ensuring that only properly tested rules are released to developers of other system components and to the domain experts and business owners

[15] For example, formulas can trigger abort or exit operations which prevent one or more current groups from completing execution. In Figure 8, for example, it's intended that at any time any rule encounters a condition that requires the loan to be rejected, the all active groups up to and including the Verification group are terminated. Execution would then continue with the Loan Approval group (presumably setting zero values) and the Workflow group (setting values appropriate for Rejection).

who review them.

Domain experts and business owners need to review business rules in normal business language or something close to it. For this purpose, IDIOM converts business rules expressed as IDIOM formulas into near-natural language. The sample shown below in Figure 9 is an excerpt from the rendering of one formula (a heading section including contextual details and comment has been omitted).

Figure 9: Example of IDIOM formula converted into near-natural language



Rules testing

A fundamental principle of the IDIOM approach to business rules development is we make information systems more effective and flexible when business rules are independently modelled and, through code generation, can be rapidly deployed or replaced in the running system. For this to be achieved, the development, testing, and release of rules must be part of an integrated approach entirely under the control of business people.

Tony Morgan[16] classifies testing as one of three kinds of quality assurance, along with walkthroughs and inspections. This implicitly makes the same point: that testing must be closely integrated into the rules development process, alongside other kinds of review. Reviewing tools available in 2002, and finding none suitable, Morgan correctly observes: "The ideal tool [is] one that's designed specifically for rule-set testing."

In the absence of a properly focused tool, it is hard for business people to do effective testing without reliance on programmers. Such a reliance has

[16] [Morgan 2002], chapter 5.

disadvantages of time, expense, and accuracy. Programming a custom tool is many times slower and costlier than reusing a good existing one. That being the case, a common alternative is that a business analyst (the rule author) first defines a suitable set of test cases that exercise all the rules, and then programmers are responsible for testing the rules in the testing environment. This has not only the disadvantages of time and expense incurred by increased programming. It is time-consuming for the business analyst to specify the tests in sufficient detail for the programmers to implement them accurately and understand what the correct results might be. The process by which the programmers convert the test specifications into executable form introduces opportunities for error. And programmers are less able than the analyst to improve or correct the tests as they are being developed and executed. In short, the division of labour introduces an unproductive impedance.

An integrated approach to testing removes the impedance by allowing the rule author to develop and run complete tests as part of rule development. With IDIOM, the rule author, as rules are completed is able to immediately create and run tests that exercise the rules, using an integrated test tool. Each test is simple to create: it takes the form of an XML file conforming to the decision's schema, and containing sample input data for one decision. The author runs the tests as often as required, interleaved with any necessary corrections to the rules. The results are viewed after each test as a parallel view of the input and output states of the decision data; in the latter, output fields are expected to be filled with the rules' outputs. If the result is good it can be saved for comparison with future results from the same test. Tests can easily be restricted to particular data fields or rules of interest: it is not necessary to always execute all rules in the decision. Suites of tests with comprehensive coverage can be quickly built up. The amount of XML syntax the rules author must understand is trivial.

The greatest advantages of this approach arise from the fact that the same person writes the rules and carries out the testing. When this is the case the tester fully understands the logic being tested, is instantly able to spot errors and correct them, and is able to implement additional tests when unexpected bugs or results suggest them or gaps in coverage are noticed. The hands-on approach achieves substantially better coverage and accuracy than the approach of thinking out test cases and results for someone else to implement and execute. It is, of course, also faster and cheaper because other workers or departments do not have to be involved.

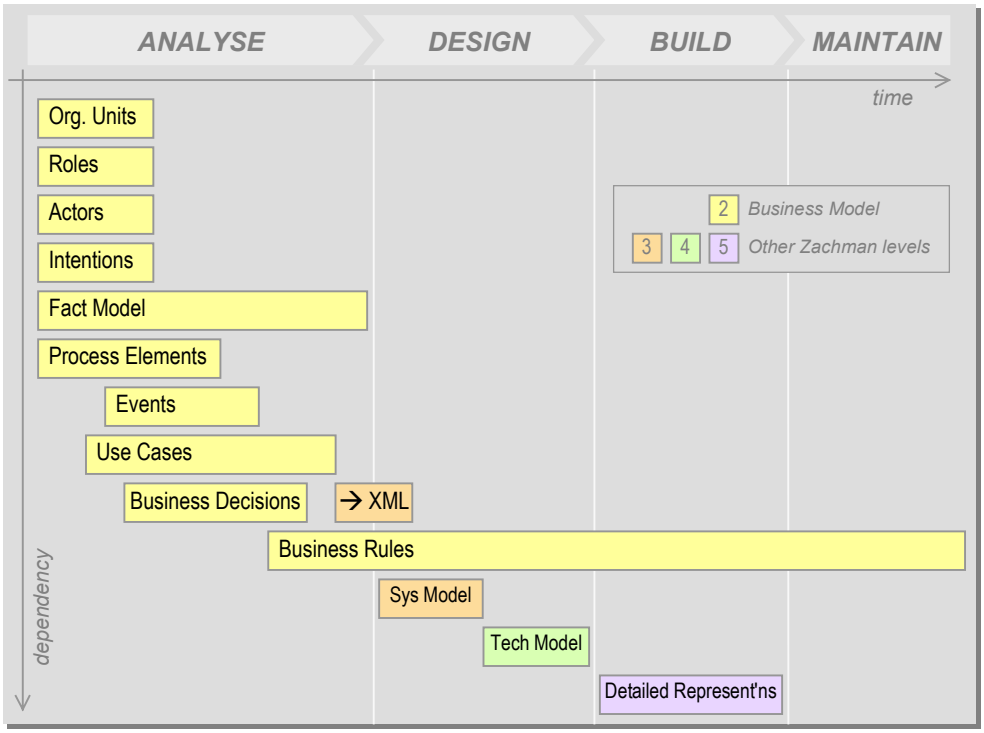
The IDIOM test tool emulates exactly what occurs in the executing information system. It guarantees that the rule behaviour observed during testing will be exactly reproduced in the running system. When the rule authors' test suites are complete enough – and this should be verified under the organisation's quality assurance procedures – the entire testing of business rules is confined to the one place where it can be done most effectively, and there is only a minimal need for further system testing when a new generated rules component is released.

Incremental system testing is also readily supported. At any time the application programmers are ready to test components that use business rules, completed decisions in the rules repository can be converted into their generated form for trial use.

Summary of the overall development process

The following diagram gives an rough overview of the development process we have been discussing. The coloured rectangles represent artefacts that are produced, and the columns the activities which produce them. The horizontal axis represents time; first three columns describe the steps from analysis to implementation of the system, and the rightmost column represents future maintenance of the built system.

Figure 10: Rough sequencing in Morgan-IDIOM development process



The colour conventions used here are the same as in the diagrams of the Zachman framework presented in the Appendix. The yellow rectangles represent the various elements of our proposed business model; details of the Design and Build artefacts (orange, green, mauve) are outside our scope and not spelt out. The width of each rectangle suggests the span of time over the bulk of the artefact's content is likely to be developed.

The vertical axis roughly indicates logical dependencies between artefacts: boxes lower down are dependent on those higher up.

The business model artefacts depicted here (yellow) are based on the same sample business architecture as was presented in Figure 1, with the following IDIOM-related additions:

- The 'business decisions' box indicates the decision inventory. It has a similar time-span to the use cases, because the two are developed together. The orange box captioned 'XML' represents the step in which the decision definitions in the inventory are formally expressed as XML schemas in order to create an IDIOM rules repository.
- The 'business rules' box begins late in life of the business model, after use cases and the decision inventory are well advanced. The bulk of the rules will be built at this time, but IDIOM provides the freedom to continue adding and changing rules for the remaining lifetime of the system.

Impact of introducing business rules

Any reader of this paper working with an already established development process may be wondering what is involved in introducing the IDIOM-based approach for business rules that we have been describing. In broadly answering this question, we assume that use cases or narratives in some form are produced under the existing process, but business rules are not explicitly analysed or modelled.

The modelling side

It should be possible to begin modelling business rules with minimal upheaval. The use cases should be the only existing requirements artefacts which are impacted, assuming that these are in full enough form for it to be possible, with some re-analysis of their content, to identify the places where business decisions are made and the input and output data for each decision. A decision inventory needs to be started, containing the usual outline description of each decision. This may mean moving a small amount of material from the use case into the inventory: for example, descriptions of the decision's data elements, or anything that will eventually become the contents of business rules. For use cases which follow the common pattern of first assembling data for a decision, and then executing the decision, the modifications should be easy to make and well localised.

After that, the next step, which of course is not trivial, is to discover and analyse the business rules and then define them in IDIOM in the manner described in the earlier sections on rules discovery and definition. The places where rules are discovered will vary according to whether an existing system (or part of an existing system) is being redeveloped or modified, or whether a wholly new system or module is being developed. Rules may be found in existing source code, or in existing requirements documents, or may need to be discovered afresh from interviews and business documentation. In all cases, rule discovery produces, as usual, rules drafted in normal language, an updated (or new) fact model, and descriptions (in the decision inventory) of rules' input and output data. In the following stage of rules definition, also as usual, rule drafts are expressed as IDIOM formulas, tested, and verified by business owners. These activities are carried out by business analysts and other business people, with brief involvement from a technologist at the start of rules definition, when business decisions are formally described as XML schemas in order to announce the proposed public interface of the rules subsystem. After these activities are finished the rules are ready for deployment as a generated component.

The technology side

On the technology side, considerably less effort is needed than on the analysis side. IDIOM is explicitly designed to make minimal technical demands: it generates code in standard languages in a form that can easily be called by application components, and does not require any changes to the technology of the database or other system components. To use the business rules, program modules make calls on the IDIOM decision service. Each such call represents the execution of a particular business decision identified in the decision inventory. In new code with good traceability back to the use cases, this should be a considerable simplification: instead of containing explicitly coded business rules, the application has a small amount of code which creates the data structure required by the decision, fills this with all required input data, sends it to the decision service. The results of rules execution are returned by the service in the structure's output fields. Calls on the decision service always follow this same simple pattern.

There are obviously implications for existing program modules that

contain coded rules which are being moved to IDIOM. In these, the coded rules need to be found and excised, and the modules brought into conformity with the protocol just described. This ease with which this can be done depends on traceability from the code back to the use cases: it is straightforward if individual business decisions are well localised in the code.

Programming work can usefully begin as soon as business analysts have published one or more of the XML schemas which describe the business decisions. Provided the analysts can undertake not to change these too frequently during rules definition, programming and rules definition can proceed in parallel. At agreed checkpoints, analysts can provide programmers with a new release of the generated rules component: this could be, for example, whenever a new business decision is available or an existing one is significantly changed.

The two sides compared

We have just outlined the two major conceptual steps of the business rules development process with IDIOM. The second step is entirely concerned with using the product of the first step to generate the rules component of the system. This step involves relatively little human effort, and disproportionately large benefits.

The first step contains the bulk of the human effort connected with business rules. It begins with an analytical effort which we call 'rules discovery', and then continues with the IDIOM supported activities of precise definition, testing, and owner verification of the rules. The result is a rigorous, complete, and accurate model of the rules that is a valuable artefact in its own right, not currently achievable with other tools. Even if the technology step were never completed or the rules were programmed by hand, this would remain the best way to model rules requirements.

Rules development with IDIOM – some practical considerations

This section discusses some practical details of the IDIOM rules development process that are peripheral to the theme of IDIOM's role business modelling. The material here will be of most benefit to readers who have seen or used IDIOM, or are embarking on a detailed evaluation.

The IDIOM application architecture; location and types of rules

IDIOM presupposes an application architecture in which business rules are incorporated into the built system in the form of a generated component which, seen from the perspective of other system components needing to invoke the rules, is the provider of a *decision service*. Units of interaction with this service technically known as *decision requests*, each of which corresponds to a business decision described in the business model.

Business decisions and decision requests

Business decisions are the backbone of the IDIOM architecture, traceable at all levels, from the business model down through the logical and physical models and the implemented programs. At the requirements level, they comprise an independent section of the business model (the decision inventory), where they describe the context in which a set of business rules is executed for a particular purpose. At the logical level they are manifest as XML schemas which describe the input and output data associated with each decision and together serve as the generated rule component's interface specification. At the physical and programming levels, the schemas describe the decision requests which programs send to the decision service, and are also the units of organisation and execution of the rules inside the IDIOM decision engine which constitutes the core of the decision service.

At the program level, the purpose of a decision request is to request execution of the rules underlying a particular business decision and to acquire the business results. The medium of exchange between the program and the decision service is a data structure conforming to the decision's schema; it comprises the input and output fields required by the decision. The program fills the input fields and sends the request to the service; the service executes the rules, which use the input data to create results and store these in the output fields. At the conclusion of the call, the program has an updated data structure in which it finds the business results it requires.

Location of rules

The decision service may be local or remote – a wide range of technical solutions are possible – but in the IDIOM architecture rules always belong in application layer (Morgan's 'middle tier'). They do not belong in the client layer, or in the data services tier.

Types of rules

Of the several types of business rules described on page 10, not all are regarded as within IDIOM's scope. Rules concerned with entity relationships are not appropriate for IDIOM because they usually belong in the data services tier.

The others – those supporting business decisions, computation, and validation – are handled by IDIOM and are the kinds we have had in mind in describing the rules development process, with the proviso that two kinds of validation need to be distinguished. Rule inputs in a decision request are assumed to have undergone some previous validation: for example, to ensure that correct data types are supplied, and values lie in

ranges regarded as sane at a business level. This kind of validation is not handled with IDIOM. On the other hand, IDIOM is designed to handle the kind of validation where incoming values are inspected according to rules and are rejected or adjusted: for example, a policy start date is rejected because it is not within 90 days of today. In such cases, decision requests typically include output fields intended specifically for error messages and warning notifications, since these are legitimate business results of rules execution.

Designing decision requests

In the rules development process, there comes a point where rules discovery is complete enough for the business analysts to want to start converting draft business rules into a precise form in an IDIOM rules repository. In the IDIOM approach, rules execute in the context of a business decision part of whose specification is the list of facts – inputs and outputs – on which the rules operate. These lists of facts need to be present in the repository before the IDIOM rules can be created. As we have already seen, each list is at this stage expressed as an XML schema, which is loaded into the repository, and also published to designers and programmers as being the intended interface supported by the rules subsystem.

Productivity vs. stability

Once a decision schema is made public, business analysts and programmers can work in parallel on the details of the decision (i.e. the individual rules) and the program modules using the decision, respectively. This accelerates development provided the rules discovery has been reasonably complete and the analysts do not subsequently need to modify the schema numerous times. Where possible, analysts should try to reduce this risk by including in the schema fields that seem potentially useful in future.

What exactly does a schema describe?

In technical language, a schema is an abstract description of an object that *conforms* to the schema. Any conforming object is an *instance* of the schema. It may be real-world object, or an object in a business model, or an object known only to programmers.

If a schema describes an object, it is reasonable to ask: what kind of object, exactly, is this a description of? Take the sample schema shown in Figure 6, for example: this schema describes an object whose characteristics include number of children, weekly hours worked, requested loan amount, and loan interest rate. What kind of object has all these characteristics? Not the loan applicant, who has children and works, but cannot be said to have an interest rate. Not the loan itself, which has a requested amount and an interest rate, but no children.

There are two reasonable answers. A business analyst would say the object represents a business decision: this is not a real-world object, but it is a real-world concept. The schema lists the facts that the decision is concerned with, and that its rules operate on. A designer or programmer sees the schema as a description of a message sent to the decision service: it is a purpose-built object containing the data accompanying a specific request for rules execution, somewhat like a real-world application form.

This may seem to be labouring the point, but it is important to be clear that the schema *does not describe a business object*. The contents of the object described by the schema are often assembled from attributes of various business objects, because it is in the nature of significant business decisions to represent the confluence of several different business objects for a specific business purpose. But in principle, the schema describes a decision concerning business objects, and not one or more business

objects per se. This is perhaps clearer when one considers the lifetime of the object described by the schema. Business objects are typically persistent objects of business record; they have a substantial lifetime. Decision request objects are transitory: they are never stored in a database, and indeed, live only for the brief span of time in which they are built, sent, returned, mined for results, and then discarded.

Factoring, reusing and adapting schemas

Considering a range of decisions made by a business – for example, house, car and personal loans – one can easily imagine that a series of decision schemas could arise with a section in common (for example, the applicant's family, health or employment circumstances) to which the same or similar rules applied in all cases. In such cases IDIOM supports decomposition of the schemas into pieces, some of which are intended to be shared by multiple decisions. As a result, each individual decision participating in the sharing is described by several schemas, either shared or unshared. Rules are always anchored to a particular schema; the real goal of shared schemas is to avoid having to code the same or similar rules in more than one place. A decision's rule set is the union of the rules associated with each of the schemas it uses. For further flexibility, IDIOM provides ways for the rules anchored to a shared schema to be subject to variations in the context of different decisions.

When schemas are subdivided in this way, some of the partial schemas that are created may seem to resemble the business objects with which the decision is concerned. For example, a schema piece containing all personal details of the applicant could be seem to be much the same thing as the Person or LoanApplicant business object present in the running application or its database. Further, some organisations already have XML schemas describing their business objects, often adopted from accepted standards for data exchange among businesses. This leads, naturally enough, to an inclination to substitute combinations of pre-existing business object schemas for the original 'pure' decision schemas derived from a careful rules analysis process.

Reusing business object schemas in this way has advantages and disadvantages. An advantage, particularly when reusing industry standard schemas for data exchange, is that all fields that might be needed in future are probably already included, so there is a promise of future stability in the schemas even during business change. But by the same token, these standard schemas are large and generalised, and inevitably include many fields that are never needed, adding clutter to the rules repository.

On the whole, reusing business object schemas is well justified if it also enables common rules to be shared among multiple decisions. If it is done not to share rules but to accommodate requirements arising outside the realm of rules, then it brings no benefit to the development of the rules themselves and may compromise the rules model. Whenever the schemas finally used are not those derived from pure rules analysis, it is a sensible precaution to ensure that the decision inventory documents both, by preserving its focus on the true business requirements and describing pragmatic deviations from these with annotations.

A related point is that business analysts and schema designers should aim to ensure that schemas present as faithful a representation of the business requirements as possible. That is, it is important to resist requests to 'pollute' schemas to accommodate technical limitations or special conventions of lower layers of the architecture. The schemas have an important role to play in publicising an accurate expression of business requirements for reference by all layers of the architecture. It is valuable to the organisation if it is known that schemas accurately track

requirements, and that a change to a schema can only mean that the requirements have changed.

Developing rules as IDIOM formulas

Rules development in IDIOM can begin as soon as one business decision, together with its data requirements and rules, has been adequately specified by analysts and its corresponding XML schema has been produced by the designers. The IDIOM repository administrator, who is typically someone with some technical experience, loads the schema into the repository and defines some minimal infrastructure for rules development (e.g. users and workgroups). The repository is now ready for business analysts to begin capturing the decision's rules as IDIOM formulas.

There are two parts to this. Given their essentially declarative nature, the rules can at first be translated into formulas, using the graphical tool illustrated in Figure 7, without much regard for the hierarchical organisation of the complete rule set that will eventually be specified with the tool illustrated in Figure 8. Logically, the formulas come first, and ordering and classifying them is secondary. There are, however, practical reasons why the hierarchical organisation normally needs to be developed simultaneously with the formulas. First, when the rule set is large, organisation is an essential aid to thinking. Second, a formula can't be tested until an atomic decision has been created to invoke it, because the test tool works by executing either all or selected atomic decisions and/or decision groups in the scope[17]. Thirdly, some decision groups are essential to flow control: for example, they can define units of procedure than can be aborted or exited, and can control iteration over repeated elements in the schema.

[17] As mentioned in the note on page 21, a scope corresponds roughly to a business decision. The tool shown in Figure 8 is the scope pane, and 'scope' is the term usually used to refer briefly to the complete set of all atomic decisions and decision groups specified in a scope pane.

Conclusion

We have outlined the approach to software development recommended by Tony Morgan, in which the key to improvements in the way we develop software lies in better capture and modelling of business requirements. Morgan's recommendations for more rigorous techniques in creating structured business models are the ideal platform from which we make another advance: recognising that business rules are a vital component of business requirements and incorporating them as a first-class element of the structured business model. This is a philosophy with which IDIOM is completely aligned, and takes a step further. Business rules are not just a primary element of the business model: they lead an important life of their own throughout the development process and the subsequent life of the system. IDIOM promotes an approach to rules development which is controlled by business people through development and production, with the aim of both streamlining the rules development process, and producing an information system that provides more accurate and flexible support for the business's essential goals.

In the development process, we have described how IDIOM now provides the tool support that Morgan, writing in 2002, hoped for. In particular, IDIOM allows analysts to model business rules in a precise an accurate graphical formula language which can be converted mechanically into the two essential forms: near-natural language for sign-off by business owners, and a generated rules component for the implemented system. We divide the rules development process into two stages, the first of which we call 'rules discovery'. This is an analysis task as conventionally understood, in which IDIOM, which is not an analysis tool, does not take part. Use of IDIOM begins when discovered rules are ready to be converted from natural language into a more precise and accurate form; this is the step we call 'rules definition'. IDIOM provides complete management of the rules from this point on: for definition, testing, and deployment. Under IDIOM, an integral part of rules definition is complete testing of the rules by rules authors: this approach is quicker and more effective than testing involving programmers, and makes it possible to guarantee that the generated rules component delivered to the system builders is complete and accurate, requiring minimal re-testing.

Appendix – the Zachman framework

	WHAT DATA	HOW FUNCTION	WHERE NETWORK	WHO PEOPLE	WHEN TIME	WHY MOTIVATION
SCOPE (contextual) *Planner*	List of Things	List of Processes	List of Locations	List of Organiza-tions	List of Events/ Cycles	List of Goals/ Strategies
BUSINESS MODEL (conceptual) *Owner*	Fact Model	Business Process Model	Business Logistics System	Work Flow Model	Master Schedule	Business Plan
SYSTEM MODEL (logical) *Designer*	Logical Data Model	Application Architecture	Distributed System Architecture	Human Interface Architecture	Processing Structure	Business Rule Model
TECHNOLOGY MODEL (physical) *Builder*	Physical Data Model	System Design	Technology Architecture	Presentation Architecture	Control Structure	Rule Design
DETAILED REPRESENTATIONS (out-of-context) *Subcontractor*	Data Definition	Program	Network Architecture	Security Architecture	Timing Definition	Rule Specification
FUNCTIONING ENTERPRISE	DATA	FUNCTION	NETWORK	ORGANIZ'N	SCHEDULE	STRATEGY

Figure 11: Zachman's Enterprise Architecture Framework

Figure 11 is a simplified depiction of the Enterprise Development Framework produced in 1987 by John Zachman[18].

The rows of the framework represent different complementary views of the system seen by the planner, business owner, designer, and so on. These views are all equally valid and comprise artefacts suggested in the individual cells (it is important to note that these are just examples: in the original diagram all cells except those in the first row are qualified with "e.g."). The rows are also known as levels, because their downward order is the order in which the views are typically developed during the growth of the system, and in a crude sense descending to the next row means adding another level of detail to the system.

The columns of the framework represent various aspects of the system that need to be considered at each level. Every cell represents an aspect of the system seen from a specific viewpoint.

The framework is intended as an aid to thinking and not as a list of components that should be bolted together to create a system. It may not be necessary to create exactly the thirty different artefacts implied in the first five rows: depending on the tools and techniques used, cells may in practice be joined or repartitioned. But it is important to *consider* every cell to avoid the risk that

[18] [Zachman 1987]

some important facet of the system may be overlooked.

Business rules are a case in point. Under the approach to the development of business rules described in this paper, we need to assign different artefacts to the cells in the *Motivation (Why)* column, and not all cells in this column remain equally important. This is discussed in the next section.

Reconciling Zachman with Morgan and IDIOM

In the body of this paper we discussed an approach to system development originated by Tony Morgan and extended by IDIOM. A guiding principle for Morgan is that requirements should be captured with the aid of a richly structured business model, one of whose advantages is that it promises in future to provide a basis for the machine-generation of system components. Business rules are an important focus of the discussion because, having first recognised rules as an essential element of any business model, we can also demonstrate, using IDIOM, that for rules at least Morgan's vision is already realisable. IDIOM creates a model of business rules from which the rules component of the system is directly generated. In this light, some questions about Zachman's original cell designations arise:

- First, business rules are now a fully-fledged element of the business model, something that Zachman seemed not to envisage when he labelled the 'Business plan' cell. A business rules model is now the dominating artefact in this cell. The element of the business model that Morgan calls 'business intentions' also belongs here, and includes the idea of a business plan.
- Second, when the artefacts of level 5 are completely machine-generated from the business model at level 2, the question arises, what happens to Zachman's levels 3 and 4? Are they bypassed, or vestigially present, or also machine-generated?

These two questions converge in the *Motivation* column. With IDIOM, all the principal artefacts at levels 2, 3, 4 and 5 of this column are IDIOM-produced or IDIOM-related, as summarised in the following diagram:

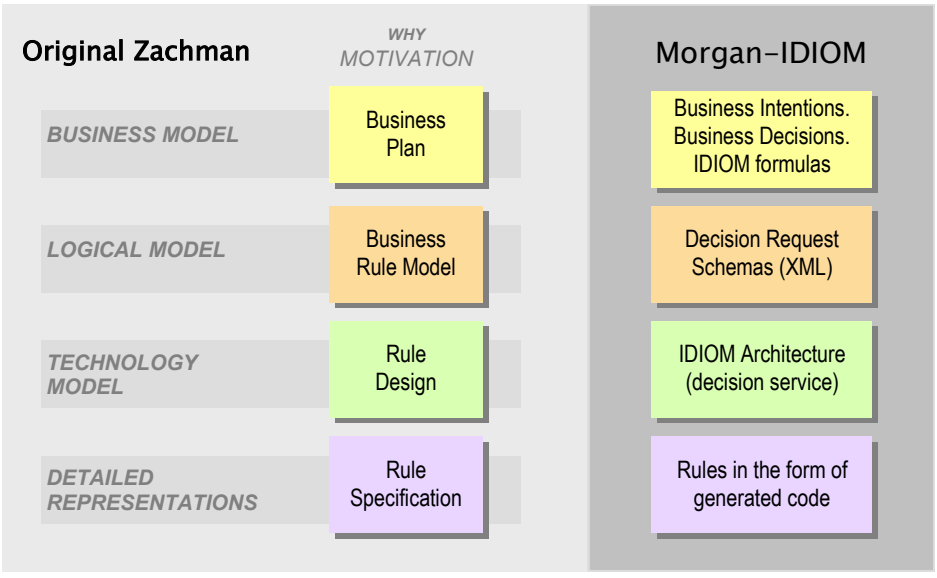


Figure 12: Reinterpretation of Zachman under a Morgan-IDIOM approach

Level 2 (Business model)

Business Intentions refers to the business model element described by Morgan. *Business Decisions* are an IDIOM-influenced clarification of Morgan: as we explained in the body of the paper, they link use cases and business intentions to business rules; they supply motivation and context. A business decision has input and output data and executes a defined set

of rules; the rules use the input data to produce the output data (or, to put it another way, a decision provides rules with their execution environment).

In practice, the main physical artefact constructed at this level is an IDIOM rules repository. It contains a rich representation of business decisions and business rules, the latter expressed as IDIOM formulas which can be displayed mechanically in various forms, including the 'logical English' form in which they are reviewed by business owners. Business decisions are expressed by IDIOM in two parts: groupings of formulas which specify rule execution order, and schemas which describe a decision's input and output data and therefore the data environment available to the formulas. The schema component of the decisions is actually consists of the 'decision request schemas' belonging to level 3, which are developed by a designer, loaded into the repository, and thereafter presented by IDIOM to business modellers in an appropriately simplified form.

Level 3 (Logical model)

IDIOM enforces an approach in which business rules are encapsulated as an independent component of the application architecture, in order to promote flexibility, maintainability, and sharing of rules between applications. This approach becomes visible from level 3 downwards. At level 3 we are aware that there is an interface over which other components of the system call on the business rules component. We picture this (anticipating level 4 perhaps) as a service which allows other components to send requests for business rule execution; requests correspond to the business decisions identified at level 2. The interface is specified in the form of XML *decision request schemas*, each of which precisely describes the input and output data for a particular business decision. The schema-based interface specification co-ordinates the separate development of the rules repository and the program modules that use the rules.

Zachman calls this cell a 'business rules model', but that term would seem to better apply to level 2. We assign decision request schemas to level 3 because they are developed by a designer in advance of programming, but they could arguably be assigned to level 4. There are no other artefacts we can assign to this cell that are part of the specific information system being built: parts of IDIOM's own design could perhaps be claimed.

Level 4 (Technology model)

At this level, as at level 3, it is hard to identify relevant parts of the specific information system being built. The technology model is largely provided by IDIOM's own architecture for the decision service. System-specific additions might include specifications for how exactly the decision service is connected to the Data and Function aspects at this level, though these would arguably seem more closely connected with Function than Motivation.

Level 5 (Detailed representations)

This level is realised by IDIOM's code generation function. Business rules are converted into a body of generated code which is integrated into the built system by being installed in the IDIOM-supplied runtime module which provides the decision service. The generated code is the principal artefact at this level; again, it can be asked whether parts of IDIOM itself qualify.

References

Morgan 2002 — Tony Morgan. *Business Rules and Information Systems: Aligning IT with Business Goals.* Addison-Wesley, 2002

Bevington 2000 — David Bevington. *Technical note – Business function specification of commercial applications*. IBM Systems Journal, vol. 39, no. 2, May 2000.
http://www.research.ibm.com/journal/sj/392/bevington.html

Bevington 2004 — David Bevington. *ASL – A formal language for specifying a complete logical system model (Zachman Row 3) including business rules.* Business Rules Journal vol. 5 no. 1, Jan. 2004.
http://www.BRCommunity.com/a2004/b167.html

Zachman 1987 — J. A. Zachman. *A framework for information systems architecture.* IBM Systems Journal, vol. 26, no. 3, 1987.
Framework diagram available at http://www.zifa.com