Going by the Rules

Faced with the inevitability of having to live with legacy systems, New Zealand Insurance decided to find a better way to maximise its investments in systems development, improve its agility and protect its intellectual property. PHIL BOWDEN, former strategy and planning manager for NZI, describes the company’s experience as it sought to build sustainable systems via the introduction of XML and a business rules-based approach.

Systems in the insurance industry have tended to become “legacy” from the day they were written. Their complexity mitigates against change, which suits an industry that has not changed its business principles much in the past 150 years. As a result, old technology has stayed around longer than is advisable and business knowledge has been locked away in computer applications — inaccessible to the business and often to the IT department, where staff turnover has left few people with an in-depth knowledge of the program coding.

Recognising this problem, New Zealand Insurance (NZI) decided some time ago to develop its systems by evolution rather than revolution in an attempt to retain its business knowledge and make it more accessible. Remembering the pain of its last revolutionary systems implementation exercise in 1984, NZI initiated a study in 1991 to determine how best to establish an evolutionary path while also preparing for Y2K. The latter had to be considered early because of the nature of the industry, where policies run for many years and involve dates far into the future. The systems were ported to HP/UX and Oracle in 1995 in order to provide a sustainable technology environment that allowed for the evolution to new systems.

By 1999 the use of the internet was being explored after a push from NZI’s UK parent, Aviva. NZI jumped at the opportunity, although studies in New Zealand had shown that the work required to develop an internet system would probably not be cost-justified for the number of policies likely to be sold via this channel. Project Quest (quotations and underwriting enabled sales tool) was set in motion, designed to service the company’s customers, agents and brokers and to ensure conformity with these two groups, including internal staff.

With quotations and new business coming through this online system, we planned that the work underpinning it would form the basis of our future development approach and the start of an evolutionary way to replace the heritage system over the next five to eight years. The first phase of Quest was implemented in 2002. At that time NZI was the only insurance company in New Zealand to allow the purchase and payment of domestic insurance directly over the internet.

Stop repeating yourself
The main driver for the path taken was to reduce the repetition of existing business rules each time the system was changed and eliminate the associated analysis. Statistics from Rational (RUP) show that 35% of a project’s cost is spent in gathering the requirements. In insurance, the basic products and processes remain virtually unchanged over a long period. Consequently, each time the systems are changed to accommodate new technology all the information has to be gathered again for incorporation in the new applications, although only a small proportion of the business rules are actually changed at this time. NZI sold about 84 different types of products, each with different information and designs, and it was taking between three to 12 months to rewrite a product from scratch.

A process that separated the business rules from the application would remove this middle step and clearly be beneficial. In theory, if this was achieved, a large proportion of the 35% of project costs spent on specification would no longer be required. To do this, the unchanging rules had to be extracted from the hard code in the programs and from the heads of many staff interacting with the current processes. The question, then, was “How do you do this?”

So how do you do it?
To answer this question a new study was begun, justified by the savings that such an approach promised by reducing the cost...
Another contentious issue is the dumbing down of users — the trend in the last 30 years to put more into the system and, as a result, have users come to rely on it and be dictated to by it. NZI had been spending around $5 million a year on software development and redeveloping. Reducing the effort on replicating unchanged business rules at the specification stage, say from 11% of the total to 1%, would represent a considerable saving.

As we developed, the decision was made that the IT infrastructure project management, and outsourcing be classified as the areas of intellectual property. All other facets of IT could be considered commodity skills and services not unique to insurance and would be progressively outsourced.

In the wake of this study the entire helpdesk and operations function were successfully outsourced and one employee, with the support of the accountancy department, managed the services. As IT evolved away from having to maintain and develop the heritage system, we reduced the amount of internal development work and moved to a primarily outsourced approach for development work. Internal staff were used for integration activities.

Throughout the study we maintained a focus on ensuring that outcomes supported business alignment, flexibility and ZTP (zero-touch processing, or the minimisation of human intervention). This led to XML becoming a key component. By that stage XML had reached a level of maturity, with supporting tools and standards becoming available, where it was felt it could be relied on to provide communication with third parties and would operate on any hardware and still be cost-effective. The company committed to XML as its transaction vehicle, with XML schemas providing both the design and understanding of the information involved. A major effort was put into the design of the data side of our business.

It was also important to look after processes, and so a search was made for process engines and business rules engines. While there was little product available to support standards-based processes, a number of business rules engines were on the market. NZI did not have a documented process model, so the study was made more difficult by the fact that many processes were run by the existing heritage system — a system the business did not know about, could not explain how to rewrite and did not even understand how complex the business had become. In other words there was often a disconnect between the rules in a manager’s brain and the rules in the procedural system. It was necessary for technicians to investigate the Cobol code to work out what was going on in the system as well as documenting the manual processes. One outcome was the generation of some statistical evidence highlighting the complexity of the business and allowing IT to better demonstrate this to the business managers.

Issues encountered

Some key questions were raised in considering system replacement. First, the intellectual property of the business consists of its rules, processes and data, and NZI wanted to retain that information within the business. However, the purchase of a software package meant the vendor and the people who implemented the package often ended up knowing more about how the business runs than NZI. As a result, the vendor retains ownership of the intellectual property. I had already experienced this working for Paxus. It can have a significant impact on an organisation’s agility and have real bottom-line implications. The question is: how do you protect the business against this? In the insurance world there are few packages with the configurable depth to allow for retention of ownership of the intellectual property.

Another contentious issue is the dumbing down of users — the trend in the last 30 years to put more into the system and, as a result, have users come to rely on it and be dictated to by it. If we were to start afresh, what kind of system would we build? And should we encourage staff to be better thinkers by giving them different types of systems where they have an individual choice on how they process?

Finally, there is the extent to which a new system would be able to adapt to future changes. Or whether, in redeveloping the system, another legacy system — another white elephant — would be created. How could this be avoided? History has shown that insurance systems tend to last 10 to 30 years but computer innovation occurs at least every 18 months.

Systems agility

During the past 20 years most companies have come to understand and manage their data reasonably well and have mastered the business rules their systems dictate. However, these rules are still being built into the code associated with each form, each screen, each back-end process, etc. This does not make the systems agile or easy to manage and maintain, and certainly does not make it possible to turn the code off at large. Systems agility in response to the need for process change is uncommon. Agility requires business rules and systems to be able to develop independently from each other.

NZI wanted to have different processes for different people, using mainly the same rules and the same technology. Although this is still a challenge, process manage-ment and process engines are starting to become available now at an affordable price and help in the way business process management languages standards coming into effect. Most process engines have been unique, allowing no interchange with other intellectual property, business processes, rules and data. In the IT domain would sit intellectual property, business processes, rules and data. In the IT domain would sit intellectual property, business processes, rules and data. In the IT domain would sit intellectual property, business processes, rules and data. In the IT domain would sit intellectual property, business processes, rules and data.
process engines. But BPM, driven by the demand for web services, allows one to do business-to-business process across multiple systems and process engines.

Why a rules engine?

A rules engine sat at the heart of NZI’s approach. Consolidating the business rules into one place allows the business to take ownership and enables accountability for the management of rules. The use of a business rules engine means changes can be made more efficiently by enabling the business to talk directly to someone capable of making the changes without major system integration testing, thus saving considerable time.

It also enables knowledge retention as business staff change and “re-use” — the ability to change one rule without changing the logic to the new system when there is a system change. It is common for the same information to be written in different languages in different systems — for example when Java is used in a web-based front end and Cobol in the back-end processing. By moving to more of a component view of the world and isolating things into various functional components, NZI’s intention was to separate the rules from the process and have only one place where an activity was performed. Many processes can re-use a function rather than using in-line code by the use of the rules engine. From a systems perspective, agility — the ability to change processes more quickly — is gained. Improved maintainability results from visibility and having everything in one place. The result is cost savings.

Buying criteria for a rules engine

The ability to deliver these outcomes formed the basis of NZI’s rules-based product requirements. The company also sought “date-driven rules” as a fundamental requirement. In insurance a customer may be buying a new policy, effective today, or a new policy, effective tomorrow, at the same time as a backend process is renewing a policy for the same type of product due in a month’s time. Consequently the rules relating to this policy must reflect such different situations at any point in time. However, NZI could not find an engine that incorporated date-driven rules in their fundamental design, so business managers needed to specify the date rules every time they required a rule to be applied or changed. This was too difficult to manage — hence the buying criteria to have such a feature implicit in the product.

Other requirements included:
- The ability for the data to be useable by any application system under any operating system.
- Cost-effectiveness, including:
  - free onward distribution of rules processing.
  - use of structured English rather than code so that the rules would be readable by users.
  - auditable change control.
  - use of open standards to ensure sustainability.

Unfortunately NZI found a lot of these requirements were not available in the commercial offerings.

Implementation:

“I wouldn’t start from here”

In evolving from the heritage system, NZI faced the problem that the only interface was the keyboard. But it was not possible simply to build a new front end that would deliver messages to the old processing system. Initially the path to the future was well ingrained with the phrase “I wouldn’t start from here”, so changes were made to shift the starting line by developing a common non-keyboard transactional interface. Once this problem was recognised and analysed, the necessary work to resolve it turned out to be a lot easier than had been expected.

The problem lay in shifting the ingrained mindset on how the heritage application worked and the standards used for making changes. But the solution dropped out quite easily.

XML was adopted as the transactional interface because it was a standard anyone could use anywhere with no hardware, database or application lock-in. It was seen as providing the most flexibility for future requirements and therefore became a mandatory requirement. By this means not only was the system opened up for added value processing, but it also separated the message and presentation layers in a way that had not been found in 4GLs.

The application was developed from that starting point. A problem encountered in researching the big systems delivering 4GLs with built-in rules engines was that they linked their rules to the controls in HTML documents. This meant that the rules were duplicated if the presentation layer needed to be different for various users but the data was generally the same. At that time, none of the large established systems could run rules against a native XML document. Generally, components for rules engines and process engines that were available from the market leaders were considered unsatisfactory in terms of cost or usability. Certainly, they had no low-cost means of distributing the rules as an executable to brokers or agents.

With the architecture established but without having actually found the rules engine to support it, we started on a Java-based development of Quest using J2EE based on WebSphere with MQ Series for delivering the data to be processed by the engine.

A partnership was formed with a software house that had developed a prototype rules engine, and the features NZI required were added into it to deliver a commercial rules engine. While work proceeded on building the Quest system, creating the forms and XML documents and coding the rules, developers added the rules into those forms — a parallel activity — added the desired functionality into the rules engine as it developed. At about the halfway stage the parallel rules coding ceased and a commitment was made only to build the rules into the rules engine system. It had to work then.

The approach delivered a cost-effective solution with low entry cost and set of rules. The engine was fully functional, and the outcome was the Idiom product.

Benefits achieved

The approach delivered a cost-effective solution with low entry cost and set of rules. Visibility to the business managers was achieved and some managers came to understand and support the project, although many others were reluctant to recognise the complexity of their rules. An agility objective had been set that required the build of a new insurance underwriting system. The approach delivered on both count — in a week and a rate change in 24 hours. The stage achieved before the takeover of NZI by IAG (January 2003)

A rate change could be effected in...
Phil Bowden originally presented the contents of the accompanying feature at an IDC/CIO Intrep session. Here are some of the questions they asked:

Case Study

How much in the original system (Pollay) was migrated to Quest?
Not a lot. Policy mostly involved interaction with humans, who knew most of the business rules. Therefore the system only had the high-level rules and not the low-level rules. It also had no rules to enable people with different levels of authority to accept certain types of claims other than in the cheque-writing process. The aim of the new system is to reduce human involvement by building any human decision-making into the business rules program and ensuring the rules are acceptable to the business.

Managing rules: there are atomic rules and then there are supersets of rules. How do you manage these?
In the structure that was used, there were two parts: the data must first be structured in a sensible manner and you need to design the XML schemas in managed chunks that you can relate to. Then you can start applying the rules to that data.

There are then multiple parts to this application: you must have authority at various levels (controlling who is allowed to manage the various rules), rules must know who they are for and who is allowed to use them. For example, if a broker wants to confine his or her scope to a particular market segment, even if NZI has a wider scope for sales, there can be intubated authority levels to confine only that broker. The higher-level rule can still be kept for NZI at large and the rest of that broker’s business. Eventually the broker or his sales representative could even put in this rule.

Secondly, there must also be version control associated with the rules, and a rollout management system needs to be part of what is being delivered. Change control is not as bad as when you are dealing with an entire system since you are only dealing with a segment. If you do not change the parameters of the interface, the only thing you need to test and roll out is that particular chunk since it will interface and work correctly. NZI took this further and extracted the business rules out of the chunk so that was the only change to be tested. Simple rule changes could be authorised from the English without actually doing program testing.

How many atomic rules are involved?
We asked the business how many rules they thought would be applied to underwrite a motor vehicle? They estimated that they would be about 80. The end result for the three broad exposures of new business in home, contents and domestic motor insurance was just under 3,200 atomic rules. The business people have probably not read and signed off on all these rules as yet. However, from the subsets that the team have already gone through with the business people, I believe we are on the right track.

Who has been responsible for the documentation?
It has been responsible for most of it to this point. However, the business has been closely involved. For example, the forms used as part of the development of Quest were developed through the marketing department. Marketing and the business product department came together to decide what business rules they actually wanted to do, and then IT came in and told them they were becoming too ambitious and tried to cut them back. But these departments hired external people to generate the forms they wanted, and from that point it was up to business analysts and the IT group to assemble the rest of the documentation to make it happen. This was not necessarily how we intended things to happen, but it was a vehicle to move the people through to taking more ownership of the business.

CIO Q&A

Can you describe the difference between a rules engine and a process engine?
The rules engine is separate and is dealing only with the application of decision-making criteria to data. The process engine deals with the manipulation, assembly and movement through the system of data, including sending the XML document to the rules engine for this level of processing and retrieving the answer. As a result of the answer the data can be moved into the back-end system or cause the printing of a document. Or, if there is an error, it can send a message back requesting further refinement of the information. In short, the process action does the job of moving the data around.
NZI did not find many good process engines on the market and had to write one itself.

24 hours, as could a product change (in considerably less time than the two-week target).
A new product could probably be developed in around two weeks unless at the back-end needed significant change, as that was still in Cobol.
The ultimate objective was to remove IT from the critical path of what the business wanted to do, and the new methods could now be seen to deliver these kinds of results. Specialised printing and formatting came later than anything else.
Business IP in terms of data (in the form of XML schemas) and rules (the business rules being applied to the data) were now captured and accurately reflected the many rules being executed. In other words, WYSIWYG rules — the rules printed out from the system in English, and as read by the business user — were the true rules being executed by the system. This was a huge advance over requiring users to read Cobol code, as had happened in the past. The approach had put a huge stake in the ground for NZI insofar as future development was concerned. Business rules could now be used for anything that could be defined in an XML document and not just for interactive processing. The same rules engine could be used to make the decisions for workflow, B2B, routing, etc. Web services were within sight.

New disciplines were required to control usage and change control processes had to be established. Statistics supplied by the system enabled a simplification of business processes, and the rules became self-contained as a general improvement in processing would evolve. The system would not be dictated by IT but using its services for business improvement.

Types of rules engines
NZI looked at basically two types of rules engines. Reference engines are used by most artificial intelligence systems and are triggered by any change of any element in those rules. Such a change will trigger a reprocessing of the rules to see if there is any change to the actions triggered. If one has a system that needs that type of process and if the decision-making requires fuzzy logic to determine issues that are not black and white, then one of the large, expensive engines is required.
However, the majority of commercial systems, those for which Cobol was originally developed, work on a hierarchy of clear procedural rules and can make good use of procedural engines that are simpler and easier to manage.

Against the original assumption that what had been written would be underwritten without human intervention, the next development was to look at commercial policies, and broker interfaces. The rules as applied to these processes but with more complexity and options and probably differing presentation layers, especially if you class a B2B XML document as a presentation layer.

We have been responsible for the business rules engine. There are two parts: the data must first be structured in a sensible manner and you need to design the XML schemas in managed chunks that you can relate to. Then you can start applying the rules to that data.

There are then multiple parts to this application: you must have authority at various levels (controlling who is allowed to manage the various rules), rules must know who they are for and who is allowed to use them. For example, if a broker wants to confine his or her scope to a particular market segment, even if NZI has a wider scope for sales, there can be intubated authority levels to confine only that broker. The higher-level rule can still be kept for NZI at large and the rest of that broker’s business. Eventually the broker or his sales representative could even put in this rule.

Secondly, there must also be version control associated with the rules, and a rollout management system needs to be part of what is being delivered. Change control is not as bad as when you are dealing with an entire system since you are only dealing with a segment. If you do not change the parameters of the interface, the only thing you need to test and roll out is that particular chunk since it will interface and work correctly. NZI took this further and extracted the business rules out of the chunk so that was the only change to be tested. Simple rule changes could be authorised from the English without actually doing program testing.

How many atomic rules are involved?
We asked the business how many rules they thought would be applied to underwrite a motor vehicle? They estimated that they would be about 80. The end result for the three broad exposures of new business in home, contents and domestic motor insurance was just under 3,200 atomic rules. The business people have probably not read and signed off on all these rules as yet. However, from the subsets that the team have already gone through with the business people, I believe we are on the right track.

Who has been responsible for the documentation?
It has been responsible for most of it to this point. However, the business has been closely involved. For example, the forms used as part of the development of Quest were developed through the marketing department. Marketing and the business product department came together to decide what business rules they actually wanted to do, and then IT came in and told them they were becoming too ambitious and tried to cut them back. But these departments hired external people to generate the forms they wanted, and from that point it was up to business analysts and the IT group to assemble the rest of the documentation to make it happen. This was not necessarily how we intended things to happen, but it was a vehicle to move the people through to taking more ownership of the business.

How do you prevent the rules based engine from becoming a legacy system?
As the information is completely stored in XML documents, it is always transferrable to something. If a standard for storing rules in XML is developed in the future, the managers of this engine will be able to adopt it — and so it should not become a legacy system. It is desirable to have the business rules remaining as a legacy because the less the business changes its rules the more stable the system will be. The process can now be changed more easily, allowing the business rules to stay the same. The data structures can also be changed to a different type of data with the business rules staying the same. I believe we can now attack our problems and evolve in a more manageable way, and by continually evolving each part of the system as required, we are able to avoid having it become a legacy system.

What is the cost?
For a large-scale package, the cheapest at that time was around $600,000. The cost to NZI was around one fifth to one tenth of this amount. There are licensing conditions on the product to suit different kinds of products and it is substantially cheaper than the big inference engines that are being sold, although it is more restrictive. What you are buying is a “developer’s licence” to construct a Java engine. Once you have this, you can distribute this engine to execute on as many machines in as many places as you like so long as it is doing your business rules engine.
Software developers can also get a special licence to embed the engine into software development and then sell the software.

9 OCTOBER 2004 29