

Business process architecture and the Workflow Reference Model

© Chris Lawrence 2006

INTRODUCTION

This paper is a response to David Hollingsworth's 'The Workflow Reference Model 10 Years On'.¹ He refers to a lack of 'agreement on a common meta-methodology for modeling the business process'. That is what this paper offers: a process meta-model. The meta-model is derived not by abstracting business process architecture from workflow or Business Process Management (BPM) implementation technology, but by analysing 'the business process in terms of its core characteristics'.²

Business rules play an important part in the meta-model, and the relationship between process and rule is compared with that of Ronald Ross's 'Business Rule Approach'.

Implications of the meta-model are explored to see to what extent they illuminate some of the process issues Hollingsworth raises in his paper.

BUSINESS PROCESS ARCHITECTURE

Hollingsworth claims the Workflow Reference Model (hereafter abbreviated to 'WfRM') was an attempt to

*construct an abstract view of the business process in terms of its core characteristics, separated from the technologies that could be used to deliver its functionality in a real world situation.*³

I am not sure it has done quite this. What it might have done instead is abstract general principles from workflow technology applications, in pursuit of effective interoperability standards for the industry. But this is different from constructing an 'abstract view of the business process in terms of its core characteristics'. The latter would call for a conceptual meta-model articulating what it is to be a business process, not just what it is to be a workflow system. Neither the WfRM nor the equivalent Business Process Management (BPM) reference model Hollingsworth proposes provides an analysis of what it is to be a business process. They both seem to take it for granted that we know what a business process is.

¹ David Hollingsworth: 'The Workflow Reference Model 10 Years On', included in the *WfMC Handbook 2004*, Future Strategies Inc, Lighthouse Point, FL, 2004.

² David Hollingsworth: *ibid.*

³ David Hollingsworth: *ibid.*

This paper offers something which I think is more like ‘an abstract view of the business process’, and one which in particular seems to fit the kind of rule-intensive contexts where workflow technology predominates, like ‘insurance, banking, legal and general administration’ as well as ‘some classes of industrial and manufacturing applications’.⁴

I would argue that the WfRM is an abstraction from workflow technology, not a conceptual analysis of the process/workflow space in business terms. Either that problem space was deliberately seen as out of scope – which is fine; or it was assumed that abstracting from workflow technology offerings is the same as conceptually analysing the process/workflow problem space – which is not so fine. By extension the BPM reference model could end up as an abstraction from BPM technology offerings rather than the conceptual analysis of the process space in business terms which it needs to be.

Hollingsworth’s paper⁵ displays other clues to what seems a generally (deliberately?) un-analytical approach to ‘process’ – as if assuming that conceptual analysis may not get us far. ‘Process fragment’ is a case in point:

...more emphasis is required on the decomposition of processes into fragments and their consolidation in various ways to support more dynamic operational business processes. This stems from the vast increase in co-operating e-businesses brought about through the Worldwide Web... The original model identified various ways in which process fragments could interact – hierarchic subprocess, parallel synchronised processes, etc and did develop runtime models for binding them in execution terms. However, it did not attempt to develop anything beyond a primitive choreography capability in the area of process definition support for interactions between process fragments.

Later on the ‘overall process’ is seen as a

combination of process “fragments” which can be recombined in various ways to deliver new or modified business capability.

But what is a ‘fragment’? We approach an answer in the discussion of its ‘internal’ and ‘external’ aspects, which suggests it may be more a *functionality* component than a *process* component:

Some form of choreography is required to identify the valid sequences of messages and responses across and between the participating

⁴ David Hollingsworth: The Workflow Management Coalition Specification: The Workflow Reference Model, Document Number TC00-1003, Issue 1.1, 19 January 1995.

⁵ David Hollingsworth (2004), *op cit.*

process fragments. The choreography requires each process fragment to exhibit a set of prescribed external behaviours in response to such message sequences.

I do not expect my suggestion above to go unchallenged – but will explain later what I mean by *process* as opposed to *functionality* component. For now I will merely remark that a ‘process fragment’ seems like a piece of broken pottery, defined by what it is a fragment of. Conversely a ‘process’ seems little more than a set of process fragments linked by messaging technology. ‘Process fragment’ is something smaller than a process, but that is about it. It is not an analytical component.

Another apparently equally arbitrary concept, but of something bigger than a process, appears in the later discussion of the conceptual model phase, which

needs to focus on the position of the process within an end-to-end process delivery chain, involving interfaces with existing or planned processes within other organizational domains.

So a ‘process’ can itself be part of ‘an end-to-end process delivery chain’. But why should a ‘process’ itself not *be* an ‘end-to-end process delivery chain’, or vice versa?

We hear that in both the WfRM and the equivalent BPM reference model the

abstraction of the business process from the implementation technology remains an important goal, because organizations tend to think differently, using different skills and methodologies, in the two domains.

I agree the WfRM abstracts something from *implementation technology*, but I am unconvinced that what it abstracts is the business process. The WfRM offers little conceptual analysis of the business process itself, because it sees it in terms of given system functionality. Interestingly these are the very ‘two domains’ in which organisations ‘think differently’. Why should there be two domains? Is it because organisations see their business processes in terms of the systems they happen to have? – because they are continually encouraged to? But the problem is that, with rare exceptions, systems are so fragmented from a process perspective – collections of point solutions. Could this be why the ‘business process’ is seen as something in a separate ‘domain’ linking those systems and components together?

A more fruitful paradigm might be to see the business process as something existing logically *prior* to those systems, but as *implemented* in them – with varying degrees of success. In which case there would not be two domains at all: there just seem to be two when the degree of success is low.

A currently favoured implementation technology is web services. Hollingsworth is concerned that

emphasis on web services architecture...[could] constrain the development of the business process architecture. ... [T]his raises the danger of ignoring the organizational and human aspects of the business process, in favour of a resource model wholly based on web services.

This could miss the point. The danger is not so much of 'ignoring the organizational and human aspects of the business process', but of overlooking the business process itself. The 'natural technology choice' of web services represents yet another *solution* architecture capable of obscuring *logical* architecture.

What do I mean by 'logical architecture'? What is it, what is special about it, why is it needed? Is it the same as the 'conceptual model'?

Hollingsworth sees the 'conceptual model' as

the formulation of the business process in terms of business component objects and their interactions.

But if 'business component objects' are things like Customer, Supplier – maybe also Department, Role-player, Product etc – the snag is that some of these are necessary to the process while others are contingent. Customer may be part of the *what* (logical architecture) but Department will ultimately be part of the *how* (solution architecture). The business process may be implemented in terms of 'business component objects and their interactions', but we could miss a valuable step if we see the *conceptual* model (ie the highest and most abstract level of model) as something formulated in terms which include components of solution architecture, broadly conceived. This missing step is important for automation and integration, both of which call for the right representation in the right generic structures. We need a meta-model which promotes this kind of rigour.

Hollingsworth provides a clue as the discussion turns to translating the conceptual model into an executable model:

It has proved relatively difficult to develop a fully standardised framework model to support this function within the BPM model. Not only do different BPM/workflow products have different internal representation structures but a large amount of local detail needs to be defined to develop the complete internal representation of the executable model.

Not surprising, with so little basis for shared understanding. We can string robotic components together with messages conforming to particular standards. But by what criteria do we recognise one string as a 'business process' but not another string?

Hollingsworth continues to talk in solution architecture terms:

*XPDL attempts to step around this problem through the use of extended attributes for bi-lateral agreement. In practice many BPM products have process design tools that are closely coupled to the execution capabilities of the BPMS. **Generating a common approach across multiple product environments may depend more on agreement on a common meta-methodology for modeling the business process.*** [My emphasis.]

That is what I offer in this paper – a candidate meta-methodology for modelling business processes at logical level. But we will see it starts somewhere different from the solution-architectural level of 'Service Interactions and Choreography':

the choreography could be likened to a very high level process definition that links together process fragments by providing a set of high level business rules and identifiers for the locations of the resource implementing each process fragment.

There are two different things which should not be conflated. One is conceptual analysis into what it is to be a business process – leading to an agreed shared meta-model. The other is a technology framework and standard protocols to allow business processes to be implemented in a variety of ways – including (where appropriate) participation by parties and services not identified or known at the time the process was designed and built. The meta-model should guide, but should not be, that technology framework.

Hollingsworth asks:

WHAT BPM STANDARDS ARE REQUIRED?

At the heart of any BPM reference architecture lie the methodology and standards for representing the business process... [which needs] ...to be considered at two levels:

(i) a lower level, internal view of each process fragment that is similar to the traditional workflow process model

(ii) a higher level view concentrating on modeling the overall process flow, linking re-usable process fragments together via a choreography. This is a view of the external behaviors of the process fragments...

But both of these are solution level – *how*, not *what*. If we want a ‘common meta-methodology for modeling the business process’ we should consider the business process as first of all a *what*, as something which can be implemented in a *how*. That way we can represent the business process as a business process, not as a piece of automated machinery – even though that may be how we want to implement it.

Hollingsworth surveys a number of existing approaches to ‘Internal Process Definition’ – ie (i) above – which

all provide a means of representing process flow, events and decision points, and the classification of various process context data associated with executing processes. Some standards also provide a means of specifying the work resources associated with the process work items, or activities.

The purpose of these tools is to enable the integration of different process design products with different execution products or to allow simple migration of existing process definitions to a different design/execution product.

[But a] particular problem has been that different vendor products in the BPM space tend to use different process design paradigms.

Again, not surprising, without agreement on what a business process is. The following design paradigms are discussed:

Transition Based Representation:

...typically derived from Petri Net methodology; a process is represented graphically as a network of activity nodes and explicit transitions between them.

A powerful and familiar paradigm, sharing some features with the meta-model I will introduce later. But clarity is important as to what level the transitions are at, what sorts of things are making the transitions, what the transitions are between, and what the ‘activity nodes’ actually are.

A significant practical distinction is whether the transition logic allows [backward] transition to “earlier” [preceding] activities, allowing cycling through a set of nodes, or constrains transition flow to acyclic paths only.

This is one of the reasons proprietary generic workflow can struggle to achieve real process control. In many contexts the question is whether or not the business logic and logistics at any particular point make a return to a prior state sensible. This in turn depends on what has happened before and what is happening now to the real business entities. There are circumstances where looping back is inappropriate because it would mean

rewriting history – eg pretending we had not written to the customer first time round. Terms like ‘transition’, ‘cycling’ and ‘acyclic paths’ beg the question as to what exactly is going through the process. This will be a key piece of the meta-model. If there is a ‘thing’ going through the process, the question of whether or not looping makes sense at a particular point typically depends on what is happening to the ‘thing’ at that point.

Block Structured Decomposition:

Any single node in a process model may be decomposed to a lower level of underlying process (a paradigm based upon the hierarchic sub-process model).

This appears to refer to the familiar ‘nesting’ construct in which (say) a process can be a subprocess or activity in another process, creating opportunities for flexible reuse. But is this nesting a feature of the logical model or a desirable feature of any solution architecture? The difference will be explored later.

Activity Pre- & Post-conditions:

In this approach no explicit transitions between activities are declared. The process is defined as a set of activities each having entry (pre-) and exit (post-) conditions; parallelism is implicit and when pre-conditions are met the activity is initiated, independently of the status of other activities within the process. To provide sequential operation, pre-conditions may relate to the completion of a particular prior activity...

This, like the transition-based representation, may inform a successful solution architecture. But neither seems to illuminate what a business process is at a logical level, and what is going through it. They could both apply to a natural process or an automated control function just as well as to a business process.

Role Activity Diagrams

RADs focus on defining the process through the actions that are taken within abstract roles associated with the organization, and the interactions between roles. In this way the process flow is modeled as the combined actions associated with the cooperating roles and their interactions. ... Modeling of data and documents is not normally handled.

But what is passing through the process, if the process itself is the ‘combined actions associated with the cooperating roles and their interactions’? Organisational roles (however ‘abstract’) are *how* not *what*. They are there because processes are implemented as they are: with a different implementation the roles might be different, or may not even exist.

Modelling a process in terms of roles and role interactions could be putting the cart before the horse.

Having summarised these different paradigms, Hollingsworth concludes:

The problem for the systems integrator is that it is not easy to transfer process information between design tools and/or workflow control software based upon the different design paradigms. A very large amount of work has been undertaken by both industry and academia in attempting to define a common representation of a business process which may be translated between these different paradigms, and, by extension, the different process modeling and definition tools associated with the paradigm.

...The recent work on BPMN represents an interesting approach... By encouraging adoption of a common modeling notation to express the core components of process structure in a standard manner, it reduces some of the variety that constrains the production of a common machine interpretable process definition.

But how far does agreement on notation take us, without agreeing on what the notation expresses? It is as if we have all agreed what the 'core components of process structure' are, as we are all supposed to have agreed what a business process is.

BPMN most readily depicts an as-is or to-be implemented process (*how*), rather than the *what* of the business process at logical level. For example the use of pools and lanes to frame and contextualise the process is most relevant once the physical implementation (*how*) has been decided. But BPMN is methodologically agnostic:

*BPMN is independent of any specific process modeling methodology.*⁶

And there is thankfully little to prevent it being used to depict business processes at logical level, as the diagrams supporting my presentation below should demonstrate.

It is no surprise that transferring process information between different paradigms is a struggle. The attempts to 'define a common representation of a business process' appear to be abstractions from different technology approaches, not different logical analyses of what a business process is. It is like doing data conversion and integration in the days before the relational model, or like transferring accounting information without considering double entry.

⁶ Stephen A. White, Introduction to BPMN (Introduction to BPMN.pdf), IBM Corporation, 2004.

Having surveyed existing standards for internal process definition Hollingsworth turns to 'Choreography & External Process Interactions'. Focus has mostly been on extending internal process definition approaches to external business-to-business (B2B) and business-to-customer (B2C) contexts; or, as e-business gets increasingly sophisticated, on

structured sequences of messages and the process implications behind such sequences... [all the time] ...remembering that the prime requirement is to support processes intermeshed across organizational boundaries.

A revealing reminder. It is one thing to imagine, design and implement a process 'intermeshed across organizational boundaries'. But what is the business-architectural status of a process like that? Who owns it, who governs it, who is accountable for what aspect of it? More fundamentally, what is it there for? To get to an effective logical model we need to steer clear of the solution level: Petri Nets, organisational roles, functional decomposition, rules engines, interoperability, choreography, message exchange, process fragments. We must start further back.

A PROCESS META-MODEL

Request and outcome

The model I want to present has already been outlined elsewhere.^{7, 8}

We start with a familiar schema:

input → process → output

Process re-engineering and quality initiatives often assume a model like this –implicitly or explicitly. It is not false, but it is too generic. It can apply to a natural process like photosynthesis or a computer system process like computation. It says nothing specific about a business process.

Derek Miers alludes to a key feature of a business process by the concept of 'process as purpose'.⁹ A thread of intentionality runs through it.

We can express this by making the first and last components more specific:

request → process → *outcome*

⁷ Chris Lawrence, *Make work make sense*, Future Managers, Cape Town, 2005.

⁸ Chris Lawrence, *Integrated function and workflow*, in: Fischer, L. (Ed.). (2005). *Workflow handbook 2005*. Lighthouse Point, Florida: Future Strategies Inc.

⁹ Derek Miers, *Issues and Best Practices for the BPM and SOA Journey*, Enix Consulting Limited, 2006.

There may be other inputs and outputs. But what makes a process a business process is that it is initiated by an implicit or explicit request. In the paradigm case the request is explicit, from an external customer. But the request could be implicit, and the customer could be internal rather than external. Where goods are manufactured and/or distributed for stock, the request itself is pre-empted. But even if the request lies in the future, it is still the reason the process exists.

The 'customer' could even be a supplier who, say, asks a customer or potential customer to 'supply' a new contract, a transaction breakdown, or payment of an overdue account. We need to construe 'customer' very broadly, as any entity likely to 'request' something the 'supplier' might agree to provide. Ultimately the initiating event is described as a 'request' to reflect the intentionality which characterises a business process. The initiating event is, or is seen as, a request which the 'supplier' intentionally responds to as a request.

Specifying 'outcome' rather than the more generic 'output' highlights the continuity of this thread of intentionality. In the input-process-output model there is no necessary identity or connection between input and output: the process takes in, transforms and/or uses the inputs, and generates outputs which may be very different from the inputs. But in the request-process-outcome model the request and outcome are in an important sense the same. The request is for the outcome; the outcome is the thing requested.

This may seem an over-complex way of stating the obvious. But it has architectural implications. A business process does not just happen. Nor is it just a series of causally related events like photosynthesis. It is a structured and intentional response triggered by something recognised as a request for a particular outcome. The request initiates the process, and part of the way through the process it is still there. Some of the steps have taken place to meet the request, but not all. The request is at a particular status. At the end of the process is the request's last status transition, which realises the outcome. The request is a structural element of the process.

This structure is apparent in administration or service processes like 'insurance, banking, legal and general administration'¹⁰ – but it is also obvious in (say) purchase order processing. An 'order' is a paradigm case of 'request', and in a purchase order process the order initiates the process and travels all the way through it. This 'order' is not a paper document, or the digitised image of a paper document. It is the request entity itself – a dataset of request parameters.

In a life assurance new business process the request entity is the proposal or application – which may or may not be on a physical form. In an

¹⁰ David Hollingsworth (1995), *op cit*.

insurance claim process the request entity is the claim. Again this may or may not be on a form, and if there is a form it may or may not be digitised.

Rules

Having identified the request entity the next step is straightforward. This is to define the rules at logical level, including rules specifying the sequence in which the other rules should be applied. A typical sequence might be:

First: rules about what is and is not a well-formed request. Until these rules have been met it may not be clear what type of request it is, and therefore what type of process should start.

Second: rules about authorisation. Once we know what kind of request it is, there may be rules about who can authorise it and what proof is needed. For example an order may need to be on an order form signed by a known representative of a customer organisation.

Third: rules about carrying out the request.

Rules like these define the process at logical level. At this level the process is the rules. A process diagram could show six steps in (eg) BPMN symbols:

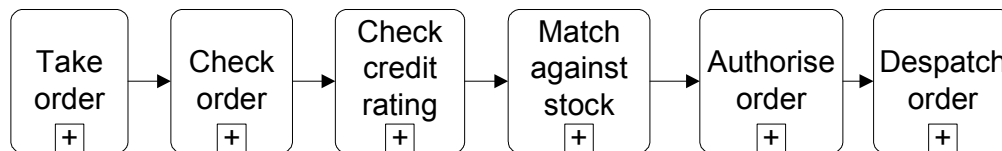


Figure 1

But this is only another way of specifying rules which could also be stated in words:

First, take the order.

Then, check the order.

Then, check credit rating.

...etc.

Within each step will be rules prescribing how to 'check the order', how to 'check credit rating' etc. For example to qualify as a well-formed order it may need to be either from a known customer or, if from a new customer, it may need certain customer details. 'Check credit rating' may also include 'IF...THEN...' rules, eg:

IF
Cash with order

```
THEN
  Pass to 'Match against stock'
ELSE
  IF
    Order amount <= available credit
  THEN
    Pass to 'Match against stock'
  ELSE
    ...etc.
```

The relationship between the rules and the request entity is key. The rules must allow for every possible value of the relevant attributes of the extended dataset defined by the request entity. For example an order will have a date, a customer, details of what is ordered and so on. The customer will have a current available credit and maybe a discount calculation rule. The products ordered will have a price, and so on. If the rules do not cover every possible value of every attribute, the process will not cover every possible request. There could be manual and/or discretionary catch-alls, eg:

```
ELSE
  Refer to credit manager for approval
  ...etc.
```

This is a perfectly sound rule, but it will need to be unpacked:

How is a 'credit manager' identified?

What happens if approval is not granted?

...etc.

Eventually the rules will be complete, when they cover every possible contingency. There will be rules within rules – rules inside the steps depicted as boxes 'Take order', 'Check order' etc. The steps may change when the rules are defined at a more granular level. For example the *Figure 1* process flow may turn out to be not quite correct, and that *Figure 2* below is how the rules really apply:

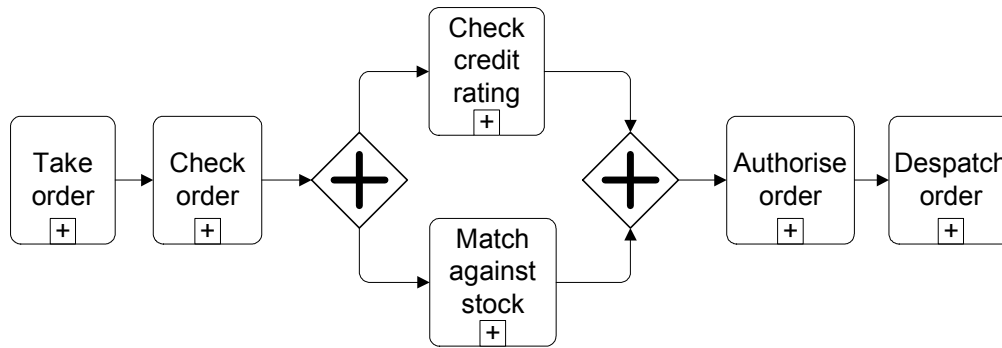


Figure 2

Here the steps 'Check credit rating' and 'Match against stock' occur in parallel, and both must complete before 'Authorise order'. It is not that one process flow is better than the other – it is purely what the organisation decides its rules are.

I must acknowledge that some of what is being said here is at odds with current 'business rules' orthodoxy. For example to quote Ronald Ross:

*Processes are processes, and rules are rules. They are not the same. A fundamental principle of the business rule approach is that each is a primitive. Formally, one primitive can never decompose to another primitive. So processes never decompose into rules, and rules never decompose into processes...*¹¹

And:

*...business rules are not "process" in any sense of the word. Roger Burlton recently expressed the business rule message this way: "Separate the know from the flow." Business rules represent the "know" part of that -- the stuff that guides the "flow." Guidance means following rules, of course -- hence the name "business rules."*¹²

It is worth digging a bit. First of all, what idea of 'business process' does this assume? Ross says the best definition he has come across is this:

*Business process: the tasks required for an enterprise to satisfy a planned response to a business event from beginning to end with a focus on the roles of actors, rather than the actors' day-to-day job.*¹³

¹¹ Ronald G. Ross, "Do Rules Decompose to Processes or Vice Versa?", Business Rules Journal, Vol. 4, No. 12 (Dec. 2003), URL: <http://www.BRCommunity.com/a2003/b155.html>

¹² Ronald G. Ross, WHAT IS A 'BUSINESS RULE'? March 2000, URL: <http://www.brcommunity.com/b005.php>

¹³ Ronald G. Ross, "How Rules and Processes Relate ~ Part 2. Business Processes", Business Rules Journal, Vol. 6, No. 11 (Nov. 2005), URL:

Rules and business processes interact (Ross quoting Roger Burlton again):

...business processes "...transform inputs into outputs according to guidance -- policies, standards, rules, etc..."¹⁴

The process is the 'flow' – a script – and the rules are the 'know' which guide the flow. So for example in making a cake, the script might be:

- 1. Combine flour, water, milk, and eggs in a large bowl.*
- 2. Mix until batter is consistent but not entirely free of lumps.*

This recipe represents a perfectly acceptable (albeit very simple) procedure or script. ... Now let's ask, what rules do we need?

Potential rules to provide appropriate control might include the following:

Milk must be fresh.

Bowl must be large enough so that contents do not spill out when stirred.

Batter may be considered "entirely free of lumps" only if there are no visible masses of congealed batter larger than 2 cm in diameter.

These rules represent business knowledge that must be present when the procedure or script is performed. ...I want both a script to follow ... and rules to guide me in doing the work. But most importantly, I want the script and the rules to be separate.¹⁵

There is also the concept of 'surrogates':

How does any model of the business (including its processes) differ from any model for the design of an information/knowledge system (including its processes)?

John Zachman¹⁶ describes the crucial difference this way. A business model "... is about real-world things." A system model, in contrast "... involves surrogates for the real-world things so that the real-world

<http://www.BRCommunity.com/a2005/b256.html>. The quotation is from Janey Conkey Frazier.

¹⁴ Ronald G. Ross: *ibid.*

¹⁵ Ronald G. Ross, *Principles of the Business Rule Approach*, Addison Wesley Professional, Boston, MA, 2003.

¹⁶ John A. Zachman, *The Zachman Framework: A Primer for Enterprise Engineering and Manufacturing* (electronic book). Zachman International (2002).

things can be managed on a scale and at a distance that is not possible in the real world. These surrogates are the things making up ... systems...." [emphasis added]. The most obvious kind of surrogate for real world things is data. A system process includes actions that manipulate data in various ways...

...A process in an information/knowledge system ... can manipulate other kinds of surrogates as well, for example:

The supervisor's work queue is actually a surrogate for a face-to-face interaction between a supervisor and an order clerk each time a special order is received.

The supervisor's GUI for displaying orders in the queue is actually a surrogate for the flesh-and-blood order clerk.

*A system process then is all about manipulating surrogates standing in for real-world things. A business process, in contrast, should never include tasks or steps for manipulating surrogates. That's a **big** difference...¹⁷*

I have no problem with the idea of surrogate as representation – in the sense that (say) a customer data record is a surrogate for a flesh-and-blood customer. But it does not follow that a 'supervisor's work queue' is 'a surrogate for a face-to-face interaction between a supervisor and an order clerk each time a special order is received' or that a 'supervisor's GUI for displaying orders in the queue' is 'a surrogate for the flesh-and-blood order clerk'. The analogy does not hold. The flesh-and-blood customer exists, and the customer data record represents the real-world entity. But the customer record does not replace the real customer, or provide an alternative implementation of the real customer. To see system functionality as primarily representing concrete entities and their relationships pertaining to a specific (and perhaps historically prior) process implementation is to fall into the trap of automating the 'as-is' which has bedevilled IT since its early days.

A subset of the rules for the supplier's order process will be criteria for deciding whether an order is 'special'. At a particular point in its history the supplier may have implemented its order process by having a flesh-and-blood order clerk and a flesh-and-blood supervisor. Part of the order clerk's job would have been to apply those criteria and refer 'special orders' to the supervisor via 'face-to-face interaction'. But consider an alternative where the order clerk only captures the order and stored system rules decide whether an order is 'special' and, if so, route it to the supervisor's work queue. There is no 'face-to-face interaction' for the supervisor's work queue

¹⁷ Ronald G. Ross, "How Rules and Processes Relate ~ Part 4. Business Processes vs. System Processes," Business Rules Journal, Vol. 7, No. 1 (Jan. 2006), URL: <http://www.BRCommunity.com/a2006/b265.html>

to be a surrogate for. Or another implementation where customers capture their own orders, there is no flesh-and-blood order clerk, but there is a flesh-and-blood supervisor complete with work queue. Or where there is no flesh-and-blood order clerk and no flesh-and-blood supervisor, and instead of routing to the supervisor's work queue, special orders invoke special processing requesting references direct from banks and/or special credit checks direct from credit agencies.

Sequencing the scenarios like this may suggest an evolution of 'improvement'. History may have been like this, but it may not have been. There may never have been an order clerk or a supervisor. These are just different implementations with different features, different costs and different technical feasibilities in different contexts. Even if things had happened in the order the scenarios were presented, a later scenario is not a surrogate for an earlier one. Something cannot be a surrogate for something which may never have existed.

The false analogy of 'process surrogates' is of a piece with the 'best definition' of business process quoted above in terms of

tasks required for an enterprise to satisfy a planned response to a business event from beginning to end with a focus on the roles of actors.

Why should there be actors? There *may* be actors, and if so they are likely to have roles. But to assume actors and roles are necessary components of a business process is as unwarranted as to assume automated system functionality is necessarily a 'surrogate' for a more concrete (or flesh-and-blood!) implementation which must have preceded it.

It is possible that assumptions like these are behind the curious statement that a business process 'should never include tasks or steps for manipulating surrogates'. (Is there an echo here of Hollingsworth's 'two domains'?)

What do seem to be necessary components of business processes on the other hand are rules. To that extent I agree with Ross and Burlton that business processes 'transform inputs into outputs according to guidance -- policies, standards, rules, etc.' But I would go further: I see no residue of 'script' which is not rules. 'Flow' is also 'know'. It is arbitrary to see 'Milk must be fresh' as a 'rule' but 'Combine flour, water, milk, and eggs in a large bowl' as part of a 'script', and therefore not a rule – just because 'I want both a script to follow ... and rules to guide me in doing the work' and 'I want the script and the rules to be separate'.

At an empirical level, there may be contexts where ‘The really rapid change is in the rules ... **not** in the business processes’¹⁸, where the steps themselves and their sequence do not change much, but things like authorisation limits and role assignments do. But equally, and particularly in pursuit of automation efficiency, the process implementation could change significantly (eg withdrawing cash from an ATM rather than cashing a cheque) while the fundamental rules stay the same (account and amount must be specified; withdrawal must be by a person authorised to withdraw from that account; withdrawal amount subject to limits; etc).

Far from separating ‘process’ from ‘rules’ a more helpful paradigm would be to see process completely in terms of rules. In terms of strict logical typing it *may* be that ‘process’ and ‘rule’ are different primitives, as a ‘computation’ is not a ‘number’ and vice versa. But it does not follow that the steps and their sequence which make up the entire content of the process are not rules. (In fact I lean to the view that perhaps a business process *itself* is a business rule at a logical level. Structurally there can be rules within rules. And for an organisation to see the data set an event represents as a ‘request’ which it then intentionally honours is effectively to follow a rule it has been given or has given itself.)

A more helpful distinction would be between the process itself at logical (rule) level and any particular implementation of it – in terms of systems, people, roles, procedure manuals, rule books etc. It is not that the ‘process’ stays the same and the ‘rules’ change. Some rules are volatile; some hardly change at all. Sometimes the physical implementation of a process stays unchanged for a long time: because it works; because it is too expensive or disruptive to change it; because no one thought of changing it or that it could change. Sometimes – particularly for cost and/or quality (and therefore competition) reasons – the physical implementation has to change.

To illustrate how artificial it is to separate ‘process’ from ‘rule’ consider a familiar type of administration process. There could be a rule that a particular document or data item must be available. What if it is missing? We do not immediately move out of a ‘rule’ domain (‘x is needed’) into a ‘process’ domain – which would then detail how to get the missing information. The stipulation that the information is required is a rule; and the statement of where to get it from is also a rule. We could imagine an implementation where on finding the information is missing the requirement rule immediately leads to generating correspondence to request the outstanding information from the intended recipient. This is process, but it is also rules.

¹⁸ Roger T Burlton, quoted in: Ronald G. Ross, "How Rules and Processes Relate ~ Part 2. Business Processes," Business Rules Journal, Vol. 6, No. 11 (Nov. 2005), URL: <http://www.BRCommunity.com/a2005/b256.html>

To return to the process meta-model, talk of rules not satisfied does mean we should qualify the statement that the outcome is precisely the thing requested. Yes the request is for the outcome in the sense that, for example, an *application* (request) for a passport is an application for a *passport* (outcome). But not every passport application leads to a passport. There are eligibility rules, and if the applicant cannot prove eligibility (eg produce a birth certificate proving place of birth) the passport will not be granted. Strictly speaking the outcome will not necessarily be a passport, but it will be a well-formed outcome in line with passport application rules. The principal or paradigm 'well-formed outcome' will be a passport. But because there are rules, and it must be possible for rules to be not satisfied, the requested outcome may not necessarily be the actual outcome.

Far from refuting the model this is actually a crucial implication of it. Exceptions and error conditions are the bread and butter of business process, because rules are.

So what are rules? We have said a lot about them but not yet attempted to define them. This was deliberate. The only definition the meta-model needs is an operational one: something is a rule if it is used as a rule. This is almost circular but not quite. It reflects the same thread of intentionality as 'request' does.

This may be heresy to some business rule theorists. I have no objection to the view that there is a significant subset of business rules which are best expressed declaratively in business language; which can more formally be expressed in predicate calculus format; which can be categorised as 'rejectors', 'projectors' and 'producers'; and for which it is true that 'rules build on facts, and facts build on concepts as expressed by terms'.¹⁹ But I am unconvinced that there is anything about 'first, take the order, then check the order, then check credit rating...' which disqualifies it as a statement of rules. The development of information systems and therefore the implementation of business processes may well be facilitated by a specifically designed 'logicbase' (consisting of a 'factbase' plus a 'rulebase') the contents of which themselves conform to rules about format, atomicity etc. But this assumes we know what our process (our 'script') already is, and our intention is to make it as efficient and as flexible as possible by providing optimised support and guidance from an optimised store of rules. This is fine as long as the script itself is optimal. If it is not how do we know it is not? Because it is more expensive or slower than our competitors'? Because its error rate is higher than our competitors'? Because it is therefore not 'best practice'? But this may only focus on how the script is followed, not on the script itself. Where does the script come from? As soon as we ask that question we open ourselves to possible ways of analysing the process itself; and I suggest that a helpful way to do this is in terms of

¹⁹ Ronald G. Ross, Principles of the Business Rule Approach, Addison Wesley Professional, Boston, MA, 2003.

rules, without presupposing any criteria as to what a 'rule' is, other than its being used as a rule.

There are many reasons a statement (typically in imperative or indicative mood) may be used as a rule. It could be a fact about the universe outside the organisation or an internal fact about the organisation. It could be something true by definition or something the organisation or some other body wants to hold true by definition. It could be a control the organisation wants in place either temporarily or permanently to mitigate risk, to ensure survival or compliance, to monitor or maintain or improve its competitive position or its quality or its profitability. And so on.

Many of the rules will be reasons why current implemented processes are the way they are. Some of the rules may be better served (now or in the future) if the processes were different. Some of the processes may be the way they are because the rules are believed to be what they are. Some explicit or implicit rules may no longer make sense or may never have made sense.

Clearly the intention should be – at least ideally, and assuming the opportunity exists – to establish what the rules are and what they should be, regardless of what is actually taking place in the organisation, including everything that is or should be used as a rule, and excluding nothing on grounds of content or format. Then the processes can be logically defined.

If formulations like 'first, take the order, then check the order, then check credit rating...' can qualify as rules, then there can be rules within rules. There can also be rules within rules within rules – and so on. But this does not mean the logical process flow itself needs to allow nesting or indefinite hierarchies. The request-process-outcome model provides a compelling argument for recognising exactly three levels.

Process, subprocess and task

The top level, **process**, is that of the request-process-outcome model itself. A request of a particular type (eg purchase order) initiates an instance of a particular type of process (eg order process). The model allows us to define a process unambiguously. It is not an arbitrary concatenation of 'process fragments', nor an arbitrary stretch of an 'end-to-end process delivery chain'. The process starts with the request and ends with the well-formed outcome.

The second level, **subprocess**, is that of the steps already identified: 'Take order', 'Check order' etc. 'Subprocess' here means one of these steps. High-level rules like 'First, take the order; then check the order; then check the customer's credit rating' etc indicate status changes which are important to the business. These business status changes apply to all orders, irrespective of their attributes. It is that generality which defines the subprocess level.

This is important. The request must qualify as a request of a particular type. Because it is a request of that type, a particular structured set of rules applies to it. That structured set of rules is the process. Because the rules are sequenced (according to sequencing rules!), the request will undergo changes in business status, reflecting what rules have already been applied, and what rules are yet to be applied. The subprocess level is the level applying to all requests of the particular type, regardless of any characteristics or contingencies of the individual request itself.

The third level, **task**, reflects those individual characteristics and contingencies. Different request datasets will need to follow different nested sets of rules and different routing because of their different attribute values. Every order will make the subprocess-level transition from status 'awaiting Check credit rating' to status 'awaiting Match against stock' (or, in *Figure 2*, status 'awaiting Authorise order'). But different orders may make that transition in different ways.

Assume for example that these are the business rules for the 'Check credit rating' subprocess:

- 1 An order may pass Check credit rating if accompanied by a cash payment greater than or equal to the total order value.
- 2 An order may pass Check credit rating if the customer is not in arrears and the total order value is less than or equal to the customer's total current unused credit.
- 3 Any credit increase where the total order value is greater than the customer's total current unused credit and the customer has not sent cash with the order (but the customer is not in arrears and the order is accompanied by a bank reference justifying the credit increase) must be manually approved.
- 4 A customer who is in arrears and has sent in an order unaccompanied by cash must be written to requesting payment before the order can be accepted.
- 5 A customer who is not in arrears, but has sent in an order greater than the total current unused credit and unaccompanied by sufficient cash must provide a bank reference justifying the credit increase before the order can be accepted.

Some of these rules allow the subprocess to be passed automatically – for example if the order meets the criteria of either rule 1 or rule 2.

Because of rule 3, the subprocess will need to accommodate manual approval before an order meeting that set of conditions can pass to the next subprocess.

Rules 4 and 5 require the subprocess to include writing to the customer for additional documentation or action. Because of this the subprocess also needs to allow for both recording and acting on the customer's reply or, if the customer does not reply, some sort of follow-up.

The subprocess could therefore have a task structure as in *Figure 3* below.

'Automatic credit check', 'Manual credit check', 'Manual record documents' and 'Automatic follow up' are all tasks. As the names suggest, 'Automatic credit check' and 'Automatic follow up' are automatic tasks, requiring the application of business rules but not human intervention. 'Manual credit check' and 'Manual record documents' on the other hand are tasks requiring human intervention.

We said above that the task level is there because the request datasets going through the process (and therefore the subprocess) may be different. Because they are different, different things may need to happen to them before the objective of the subprocess is achieved.

There is no reason why a subprocess like 'Check credit rating' cannot start

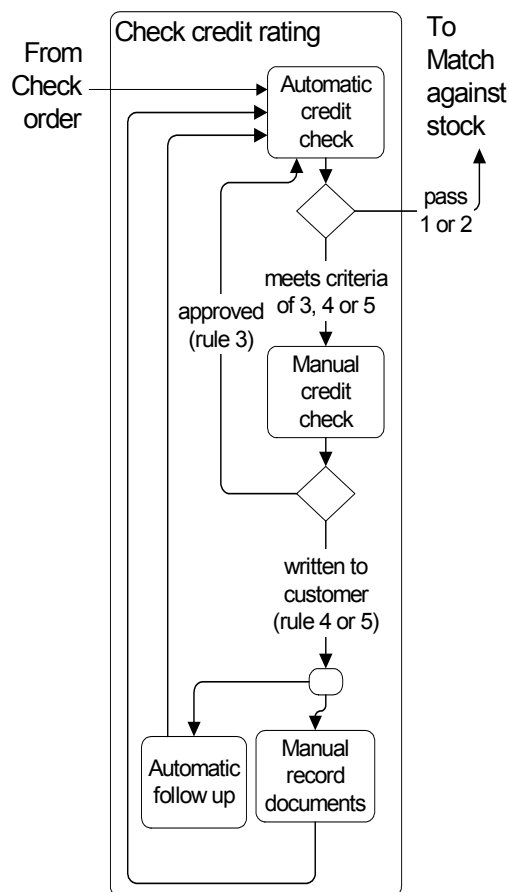


Figure 3

with an automatic task which processes the requests it can process, and re-routes the ones it cannot.

So the 'Automatic credit check' task applies all five rules. If the order passes either 1 or 2, it can go on to the next subprocess, 'Match against stock'.

If the order meets the criteria described in rule 3, then the order would need to route to a manual task to allow the increased credit to be approved.

To keep it simple, it will be assumed that the same manual task could allow for both approval (rule 3), and also writing to the customer (rules 4 and 5).

So if the order met the criteria for either rule 4 or rule 5, the

automatic task would also route it to the same manual task.

In the case of just approval (rule 3), then the order would route back to the automatic task, which would then pass the order to the next subprocess. (In theory this action could be taken in the manual task, which would be an alternative routing.)

If the customer needs to be written to (rules 4 and 5), then we need two additional tasks: a manual task for recording and acting on the customer's reply; and an automatic follow-up task to send the order back round the loop again if the customer does not reply after n days.

The details of the tasks need not concern us. The number, nature and configuration of tasks and the routing between them will be determined by the business rules governing the process itself, and applicable to the particular subprocess.

With this meta-model we do not start with what people do and what order they do them in. We do not start with actors or their roles, however 'abstract'. This is where we may end up. But we start with rules.

This example also shows that although the subprocess level is important for analysis and design, as far as implementation is concerned (other than for management information and control purposes) it is technically redundant. All the work of the subprocess is contained in its tasks, and the routing between them. A process is a set of tasks strung together with routing pathways.

But can the task break down further? Yes and no. If we consider its functionality then clearly the task could break into components and components of components. But from a pure process and control perspective a task is a unit. A task cannot 'partially complete', since the request (in this case the order) must exit in a well-formed way so as to route to the next task correctly. This is because everything that happens in a task can happen together: in the case of an automatic task it is a set of rules which can be applied in one go (in a defined sequence if necessary); in the case of a manual task it is a set of things it makes logical sense for one person at a particular authority level to do in the same session.

In the case of a manual task, it is obviously possible for a user to stop halfway – perhaps capture only part of the data the manual task is there to capture. But for this partially updated state to be possible it must be possible to exit the task at that point, and either lose everything that has been done so far or leave things in a partially-updated state. The task itself has completed in a well-formed way, and subsequent routing will take account of the abandoned or partial update – most likely by looping back to the same task to redo or complete the update.

We have now introduced an important feature of the meta-model: the distinction between the process component (the task) and any functionality component(s) needed to implement the task. From a process and control perspective the task, like the subprocess, is a transition in the business status of the request. The task is a transition from one status to one of perhaps a number of possible statuses representing possible subsequent routings. In order to achieve those status transitions functionality will be needed – and there could be many different ways of implementing the task.

The task as process component is a unit and cannot logically be broken down further. As a process component it has attributes like:

- Task name
- Input status and possible output statuses
- Whether automatic or manual
- If manual:
 - What roles and/or skill levels required
 - ...and so on.

An automatic task and a manual task will be different in that they will call for different types of functionality. A manual task may need to display and accept data via a user interface, while both classes of task will run rules and update data. But as nodes in a process flow they are equivalent, in the sense of taking the request from one input status to one of a number of possible output statuses. In theory at least it is possible to replace a manual task with an automatic task which has the same outcome, or vice versa. (In practice incremental automation is more likely to involve increasing the proportion of requests which can be handled by automatic rather than manual tasks, rather than replacing manual tasks one-for-one by automatic tasks. Removing a manual task will more likely involve rationalising routing within the whole subprocess rather than just replacing the task with a single equivalent automatic task.)

We should therefore think of the task-as-process-component as something separate from but loosely coupled with the functionality component(s) which implement it. The relationship could be many-to-many: one task could be implemented by a set of functionality components (possibly linked together hierarchically) and/or the same functionality component could implement more than one task. (This by the way is a reason for choosing ‘task’ rather than the more familiar WfMC term ‘activity’. ‘Task’ almost corresponds to ‘activity’, but the ‘task’ the meta-model needs is unambiguously a process component rather than something incorporating both process and functionality – or something which is primarily functionality.)

It could be objected that by introducing the task level the model stops being purely logical (*what*) and becomes physical (*how*). Perhaps the process could be implemented using different tasks; or in a different way entirely, not needing ‘tasks’? Perhaps. But backtracking through the articulation of

the model we will recall that the task arose by considering holistically the impact of the subprocess rules on the entire range of possible attribute values of the request dataset passing through the subprocess (including relevant external contingencies). An additional assumption is that the task – like the process and the subprocess – is a control unit at the appropriate level. The process is the control unit responsible for taking the request through to the well-formed outcome. The subprocess is the control unit responsible for taking the request from one business status to the next, defined by business rules applicable to all requests of the initiating type, regardless of individual case characteristics. The tasks are the minimum control units required so that any possible request can make the status transition defined at subprocess level.

‘Control unit’ should be construed at a management/process information/state description level. Since these are *business* processes there will be different stakeholders interested in knowing for example where each request is, or perhaps just knowing this information exists.

If the task level appears more ‘solution’ level and less purely ‘logical’ this could be because of the explicitly ‘real world’ nature of the last two axioms – about including all possible cases (and therefore by definition all ‘the more complex aspects such as handling exceptions and various error conditions’²⁰); and about the nodes being unambiguously defined control units.

But it is really a question of what the meta-model is, and so what is inside it and what is outside it. The meta-model is a coherent set of inter-related concepts which can be used to design process models at logical level, models which can then be implemented in solution architecture.

We have already alluded to the two analogies of the relational data meta-model and the double-entry paradigm in accounting.

The relational meta-model provides a coherent set of inter-related concepts (entity, relationship, attribute, primary key, foreign key, cardinality, ordinality etc) which can be used to design data models at logical level, models which can then be implemented in physical databases. It guides design choices, but does not prescribe them. It does not tell you what entities to recognise. A real-world data domain could be represented in two or more equally valid logical data models, equally aligned with relational principles. The different data models could reflect different business and strategic priorities, and different assumptions about future flexibility needs. But there would be a vast and indefinite number of ‘invalid’ data designs, ones ignoring or flouting relational principles. The strength of the meta-model is that it provides vocabulary and criteria for identifying the relatively small number of good data models and choosing between them.

²⁰ David Hollingsworth (2004), *op. cit.*

Like the relational model (and also like the double-entry paradigm, which does not impose a particular chart of accounts or ledger structure, but guides accounting design) the process meta-model provides building blocks from which a logical process design can be created. It does not say what an organisation's rules should be; or what requests an organisation should honour and so what processes it should support. It does not say what controls and break points are important and what status transitions to recognise. It does not prescribe allowable ranges of values for requests or their extended data sets. But it provides a coherently related set of concepts which, given an organisation's business context and objectives, allow that organisation to make rational choices as to what its end-to-end processes should be, how those processes should interact, what scope of intervention would repay technology investment, and how processes should be designed and implemented to maximise return on that investment.

It does this not by viewing technology implementations in increasingly generic and abstract terms, but by providing tools for analysts and architects to identify and then represent the fundamental logic of those processes in terms of their rules. The eventual breakdown and configuration of tasks will depend on the scope of the particular process initiative and the strategic objectives and priorities of the organisation. Just as there may be more than one possible logical data model for a particular context there could be more than one possible process model. But none would need to assume any particular technology implementation.

IMPLICATIONS OF THE META-MODEL

The meta-model implies and facilitates a logical process architecture. More simply: it *is* a logical process architecture.

To illustrate this I shall indicate how the model might inform some of the process issues raised in Hollingsworth's paper.

Process fragment

First the 'process fragment'. The process itself is seen as

a combination of process "fragments" which can be recombined in various ways to deliver new or modified business capability. This has become important to support business agility and follows from the increasingly integrated business relationships between trading organizations.

...The internal view defines the actual or intended internal behavior of the process fragment—it includes not just the activities and transitions between them but, also [significantly] the internal resources required to support enactment of the process. It will also identify the boundaries of the fragment in terms of interactions with other process fragments or outside objects.

The external view defines the behaviour [sic] of the fragment as a “black box”, seen from the outside and addressed through its interfaces. This view sees the process fragment very much as a source and sink of messages or events of different types.

Other than the need for ‘agility’ because of ‘increasingly integrated business relationships’, there is little discussion of the implied business context. What about ownership, governance, accountability? We need to think through what ‘integrated business relationships’ might mean in the real world.

One possible way of integrating two or more businesses might be where a process owned and defined by one business requires participation by another business. Perhaps a life assurance company A decides to outsource its new business underwriting to company B. This could translate into the meta-model by having a new business process owned, defined and implemented by A, but with manual underwriting tasks available to users in company B. Company A may then pursue greater ‘agility’ by outsourcing the underwriting of different products to different underwriting companies. For example manual underwriting task instances for products 1, 3 and 5 might be available to users in company B and equivalent tasks for products 2 and 4 might be available to users in company C. The meta-model would support this – which is not to say it would impose a particular solution.

In this scenario (where, say, users from companies B and C accessed task instances over the web) the functionality used by companies B and C might not qualify as ‘process fragments’, as the whole process functionality may be supplied and owned by company A. But the interaction could be different. Company A could send details of new business applications to companies B and C by post, email or other transmission medium; and receive back underwriting decisions – again by an agreed transmission medium. This would be similar to a process in company A generating correspondence to a customer or agent and then processing the response. It would be more like where ‘process fragments’ take place at company B and company C.

If the ‘details of the new business application’ were in a file or packet of data which was input into (say) company B’s ‘underwriting system’, then this may be even closer to a ‘process fragment’ – as company B’s system functionality could be seen as performing part of company A’s process. Perhaps company B also handles claims underwriting for company A, and interacts in the same way. Then the packets of data from company A might need to contain ‘process type’ indicators so that company B can identify them as new business or claims. Company B would read part of the data as a ‘process type’ indicator and the rest as details of the underwriting work to be done. If company B did underwriting for other life assurance companies then the data packet might need a ‘client’ attribute indicating which life assurance company it was from. This is approaching a world where

standards are needed – if multiple clients like company A and multiple suppliers like B and C intend to interact in the same way. We may need a ‘choreography’

to identify the valid sequences of messages and responses across and between the participating process fragments. The choreography requires each process fragment to exhibit a set of prescribed external behaviours in response to such message sequences.

But company A may want a more controlled, more structured, more arm’s-length, relationship with B and C, more like the customer-supplier relationship it in fact is. One of the disadvantages of the familiar ‘B2B’ acronym is that it can obscure business-architectural features by focusing too immediately and exclusively on technology implementation and the drive to ‘agility’. There is a risk of overlooking the fundamentals of the ‘B2B’ relationship itself and leaving issues of ownership, accountability and governance unclarified.

Figures 4 and 5 below show alternative ways in which companies B and C might interact with company A in respect of new business underwriting. Both diagrams use BPMN symbols, and both are aligned with the proposed meta-model.

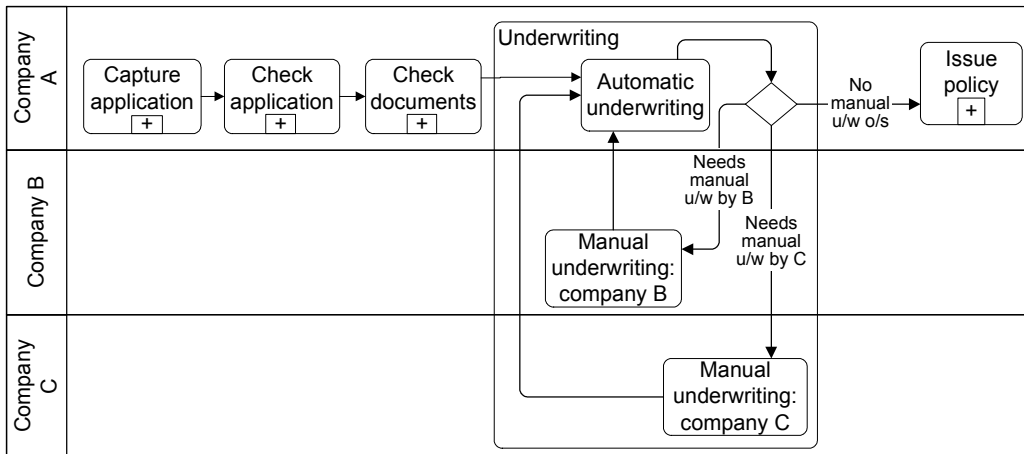


Figure 4

Figure 4 would fit a scenario where the entire new business process functionality is supplied and owned by company A. An automatic underwriting task identifies cases needing no manual underwriting (or which have now been underwritten) and passes them through to the ‘Issue policy’ subprocess. It splits the others between companies B and C according to rules, perhaps based on product type. The two manual underwriting tasks are made available to companies B and C respectively.

Figure 5 below shows a different relationship. Here the new business process owned by company A excludes manual underwriting. The automatic underwriting task again splits the cases into ones which need no manual underwriting (or which have now been underwritten); ones which need to go to company B; and ones which need to go to company C. But instead of routing to manual underwriting tasks in the same process, 'underwriting request' messages travel to company B or company C as appropriate. These initiate instances of underwriting processes in B or C respectively, and generate 'outcome' messages which travel back to company A where the 'Auto receive response' task processes them.

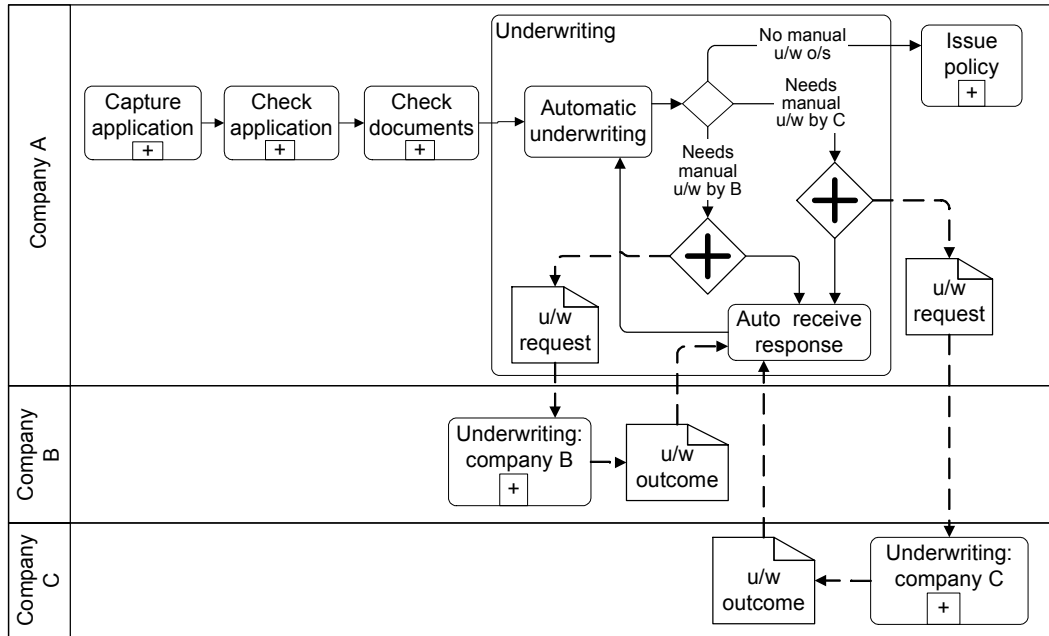


Figure 5

The two processes 'Underwriting: company B' and 'Underwriting: company C' can be completely 'black box' as far as company A is concerned, as long as they can read the right request messages and generate the right outcome messages. 'Underwriting: company B' and 'Underwriting: company C' can also be completely different from each other internally. (The boxes do not have to be black. They could have windows and/or doors, depending on the relationships companies A, B and C want to have with each other.)

Examples like these show an important feature of the meta-model in 'B2B' and 'B2C' contexts. It is possible to depict process relationships at a logical, business-architectural, level without (at one extreme) showing the process in such an abstract way that it and all its components float in space, not supplied, owned or governed by any responsible entity; or (at the other extreme) showing ownership etc only in terms of physically implemented processes. This is because the request is from a customer, and the outcome

is for a customer: the process itself is fundamentally from the supplier perspective. But 'customer' and 'supplier' are construed broadly enough to cover any 'B2B' or 'B2C' relationship.

So the meta-model allows us to simplify and sharpen our language. In *Figure 5* company A's new business process (from subprocess 'Capture application' to subprocess 'Issue policy') is a process. A process has an internal logical structure (subprocess, then task). A process can interact with other processes (initiate; suspend; release; terminate; etc) and can also interact with itself (suspend itself; terminate itself; etc). Company A's process initiates process 'Underwriting: company B' and process 'Underwriting: company C' by generating the appropriate requests. If it has issued a request message for company B it will suspend itself until it receives an appropriate outcome message from company B, which allows it to carry on to completion. We do not need to call company A's process an 'end-to-end process delivery chain'; or see 'Underwriting: company B' and 'Underwriting: company C' as 'process fragments'. The meta-model defines them all as (business) processes.

Nesting and re-use

Some models allow nested or hierarchical relationships at subprocess or task ('activity') level. For example the WfRM 'nested subprocess' construct

*allows a process executed in a particular workflow domain to be completely encapsulated as a single task within a (superior) process executed in a different workflow domain. A hierarchic relationship exists between the superior process and the encapsulated process, which in effect forms a sub-process of the superior.*²¹

The meta-model proposed here may at first sight appear more restrictive. It acknowledges nesting between *functionality* components, and re-use of *functionality* components between tasks. But the only nesting it allows at *process* component level is between processes themselves, as in the process interactions mentioned above. The reason is purely logical. The process is everything from initiating request to outcome; the subprocess is a transition in business status of the request across all possible attribute values of the request; the task structure within the subprocess allows for transitions in business status as required by the possible attribute values of the extended dataset identified by the request. A task is therefore a status transition of a specific request type, at a particular point in a particular process. It cannot at the same time be another transition of another request type, or of the same request type at a different point in the process. Nor would anything be gained by 'allowing' this kind of 're-use'. The actual functionality (by definition) is contained in the functionality component(s) implementing the task, and functionality components can be re-used and nested with complete freedom.

²¹ David Hollingsworth (1995), *op. cit.*

The only other permissible nesting is between task and process – in effect at process level. The model defines ‘process’ as everything from initiating request to outcome, so a process needs a request entity to start it. It is perfectly in order for a task P1Tn in process P1 (initiated by request R1) to create *another* request R2 initiating process P2. (See *Figure 7* under **Process interactions and agility** below.) But P2 cannot be initiated by the same request R1 otherwise R1’s outcome (‘O1’) could be generated by P2, and by definition it must be generated by P1. This is true purely logically, and is no more a limitation than that the number 2 is limited by not being 3. It is actually a strength of the meta-model that it insists on definition of request entities. For practical purposes it achieves the same result as the WFRM’s ‘nested subprocess’ construct, but in the context of formal rigour. (Process interactions will be discussed further under **Process interactions and agility** below.)

Work management

A perhaps more clearly practical strength of the model is that it can incorporate internal work management into the same control structure as external interactions with customers, agents, suppliers and other partners. In *Figures 4* and *5* above, subprocesses ‘Capture application’ and ‘Check application’ represent data entry and validation work internal to company A, and subprocess ‘Check documents’ represents that part of the process (again internal to company A) where applications are checked to ensure no supporting documents are outstanding. A mature process control architecture would integrate these into the same work management/work delivery functionality. With web deployment there is no reason why (in *Figure 4*) this should not also extend to tasks ‘Manual underwriting: company B’ and ‘Manual underwriting: company C’. *Figure 5* assumes different business interaction and ownership choices, but the interactions themselves (generation of requests and processing of outcomes) can be incorporated into the same (‘internal’) work management/work delivery functionality as the rest of company A’s process. There is no logical or architectural reason why ‘B2B’ and ‘B2C’ interactions should be in a different control domain to ‘internal’ work management, and every reason why they should not. We do not have to risk

ignoring the organizational and human aspects of the business process, in favour of a resource model wholly based on web services.

An effective process model can cover every part of every business process – including both (i) fully internal and (ii) web-deployed interactive and indeterminately resourced processes and parts of processes.

Insofar as the meta-model implies (*is*) a logical process architecture, it can also inform and guide a physical solution architecture capable of integrating process control, work management and application system functionality in the most immediate way – such that the business process

design and the application system design are one and the same.²² There only need to be 'two domains' when implementation technology determines (and therefore constrains) business process. When the horse is before the cart, and business process determines implementation technology, there is only one domain – of process implemented in technology – calling for one holistic mindset, one holistic skill set, one holistic methodology.

Late binding

Figure 5 also indicates where the model can handle 'late binding of resource to task' and/or 'dynamic discovery [of services] at fragment execution time'. Trawling cyberspace for available underwriting services may be far-fetched, but if it did make business sense then the only change would be to make the 'u/w request' message unspecific to either company B or company C, and instead conform to a standard format and be transmitted to an appropriate broking service ('resource directory') which would link it up with an underwriting service provider capable of handling the request and generating an 'u/w outcome' message in an equally standard format.

For a less far-fetched example, subprocess 'Match against stock' in the earlier order process (*Figure 1* or *Figure 2*) could be not the purely internal subprocess it is perhaps assumed to be but a set of tasks responsible for issuing standardised request messages destined for appropriate wholesale suppliers (again via an appropriate broking service/resource directory) and deciding (on eg price and lead-time criteria) which 'outcome' messages to confirm and therefore which (perhaps previously unknown) suppliers to contract with.

Process and data

Under the heading **Information and its relationship to process and organization** Hollingsworth sees a distinction between 'process-based' and 'information based' architectures:

*Process-based architectures tend to emphasise process as the dominant dimension; processes consume, generate or transform information, behaving in accordance with a set of corporate governance rules. By contrast, information based architectures emphasise the information dimension, viewing processes as operations that are triggered as a result of information change.*²³

Does this have to be true? Must we choose between process-based and information-based architectures? Remembering the distinction between task (as process component) and the functionality component(s) implementing the task, then what sort of thing is a task?

²² Chris Lawrence, *Make work make sense*, Future Managers, Cape Town, 2005.

²³ David Hollingsworth (2004), *op. cit.*

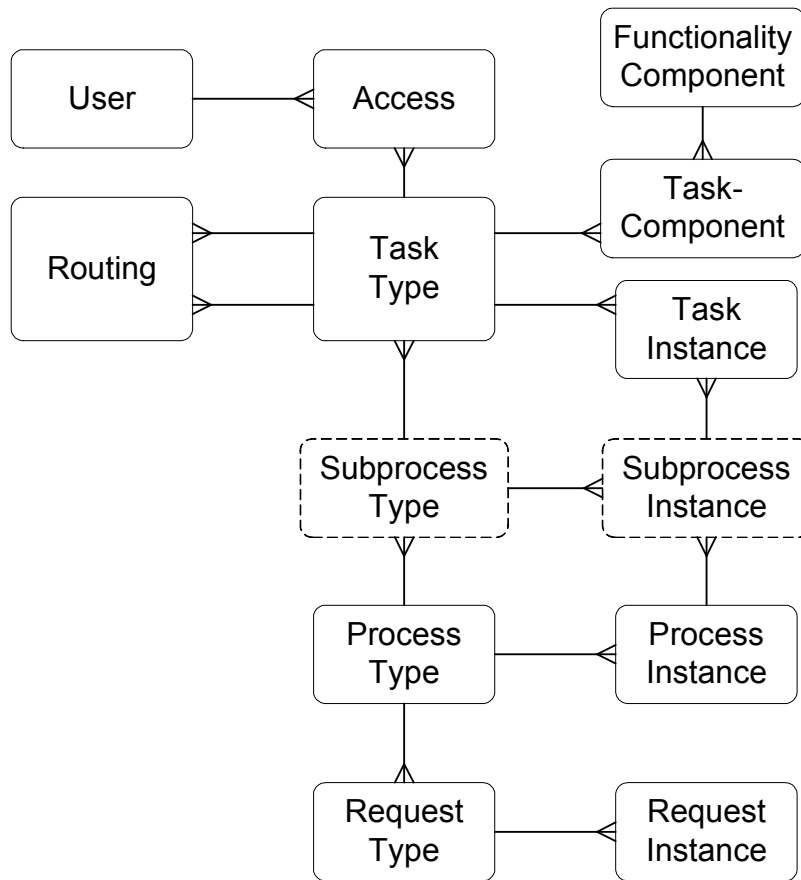


Figure 6

In an implemented process-architected solution the relevant entities might be related according to a logical data model something like *Figure 6* above.

Note the repeating type–instance pattern. A ‘process model’ (of a business or business area) is at type level, where the rules are. Each instance is an entity created as and when required for the individual case (RequestInstance) and conforming to the rules set by its type. For example an ‘automatic credit check’ TaskInstance (for eg order number 0001) will behave as determined by the ‘automatic credit check’ TaskType.

Stepping through this data model then, a request (RequestInstance) is recognised as being of a particular RequestType, which identifies the correct ProcessType. A ProcessInstance is created of that ProcessType. The ProcessType defines a set of SubprocessTypes which are the subprocesses for that process. (SubprocessType and SubprocessInstance are shown in broken lines because, as mentioned, while useful for purposes of analysis, design and management information they can be regarded as redundant for implementation.) SubprocessType and/or ProcessType identify the set of

TaskTypes for each relevant SubprocessType/ProcessType. TaskInstances are created as and when required, as not every ProcessInstance will need instances of every TaskType. Typically however a TaskInstance for the first TaskType will be needed to start the process.

The (first) TaskInstance will run by causing the relevant FunctionalityComponent(s) to run. The relationship between TaskType and FunctionalityComponent is many-to-many, but there would typically be one controlling FunctionalityComponent responsible for starting the task and calling other FunctionalityComponents as required. If the TaskType is a manual task then Access will define the User(s) able to perform the TaskInstance.

Routing is an intersection entity representing rules as to what 'next task(s)' are possible for each task. Attributes would be eg 'From TaskType' (ie this TaskType), 'To TaskType' (ie next TaskType) and 'Condition' (ie exit condition under which the 'From TaskType' would lead to the 'To TaskType').

In a physical implementation many of the actions and links described here would be undertaken by generic process functionality operating against process control type data and creating process control instance data as required. This generic process functionality, along with the type and instance process control data, is typically referred to as a 'process engine'.

This summary of a possible process-architected solution demonstrates that components like process, subprocess and task are best thought of as *data entities* which control, guide and implement the behaviour of a process-based system. A sophisticated 'process-based architecture' is therefore one which treats process as data at the logical level and implements process as data at the physical level. Conversely a sophisticated 'information-based architecture' is one which extends its reach to include process entities. If 'process' is a subset of 'information', why should we distinguish between process- and information-based architectures? An information-based architecture which viewed processes *only* as 'operations ... triggered as a result of information change' would be better described as an *incomplete* information-based architecture. Conversely a process-based architecture which saw processes *only* as things which

consume, generate or transform information, behaving in accordance with a set of corporate governance rules

– thereby missing the point that *business* processes themselves can and should be modelled and implemented as data – would be better described as an *incomplete* process-based architecture.

We do not have to settle for incomplete architecture. We can take business process seriously, which includes taking seriously its representation in data. Hollingsworth admits that the WfRM

does embrace all three dimensions [process, information and organization] but takes a relatively simplistic view of the information dimension.

This simplistic view could be one of the reasons it is often a challenge to extend proprietary workflow packages to provide true process control functionality – despite vendor claims. Process control is about rules, and rules are data.

Process interactions and agility

This is a convenient point to return to the discussion of process interactions – important because of implications for ‘process agility’.

The model lets us distinguish two different kinds of interaction. One is the type mentioned under **Nesting and re-use** above, where eg a task P1Tn in process P1 creates a request R2 to initiate process P2:

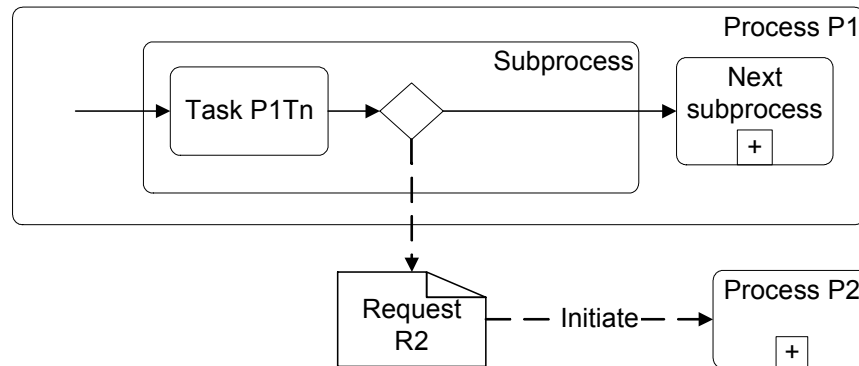


Figure 7

Another is where task P3Tn in process P3 merely creates or changes some business data so that when a rule in task P4Tn in process P4 runs it has a different result than would have occurred before the update:

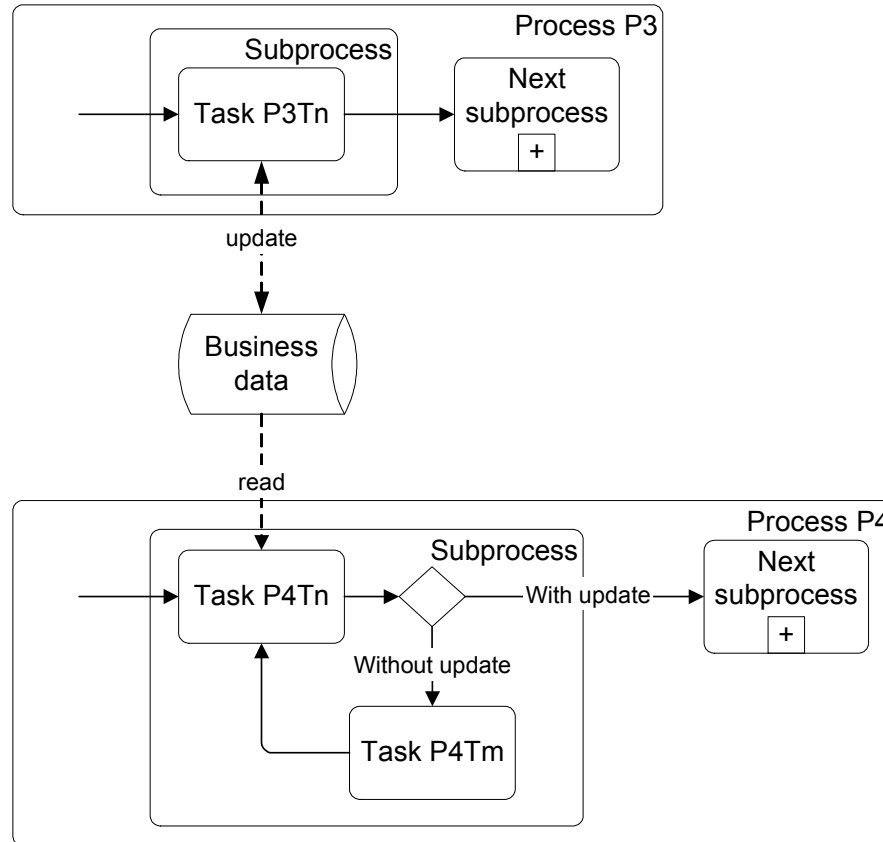


Figure 8

On one level (that of the functionality components) there is no distinction, as 'P1Tn in process P1 creating request R2' could be described as 'creating or changing business data'. The difference is of course that the 'business data' in *Figure 7* is a request, and 'request' is a structural component of the process meta-model.

Consider now a familiar version of *Figure 8* where the business data task P3Tn updates is control data implementing a business rule. An example might be where process P4 is an order process, the affected subprocess is 'Authorise order', task P4Tn is 'Automatic authorisation' and task P4Tm is 'Manual authorisation'.

In *Figure 9* below the automatic authorisation task passes orders where the values are within limits stored on the 'Authorisation limits' file. 'Authorisation limits' could hold maximum values for, say, different product types and/or customer categories. Any order within these limits passes to the next subprocess. Any order exceeding one or more limits routes to a manual authorisation task.

The authorisation limits themselves are updated by a separate Parameter update process. Imagine that for a particular product/customer category combination the Parameter update process increases the limit from £100 to £120. Before the update an order for £110 for that same product/customer category combination would route to manual authorisation. After the update an identical order would pass automatic authorisation.

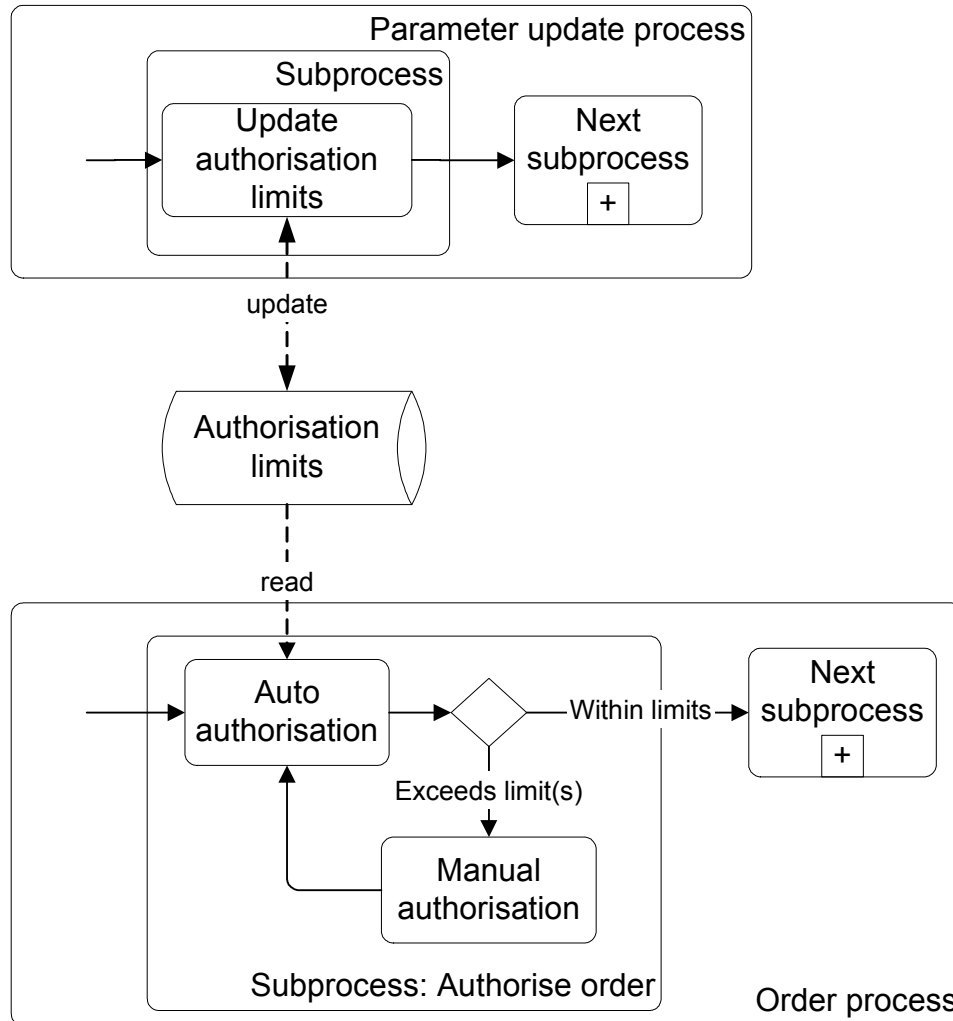


Figure 9

The meta-model supports this kind of ‘agility’. The order process itself has not changed, but its manifest behaviour has because a rule implemented as control data has been changed dynamically.

It is perhaps a relatively ‘tame’ kind of agility. But even so its dynamic nature has implications which need to be acknowledged. Should we worry about equitable treatment? Consider two orders from two customers in the

same category arriving on the same day for the same amount of the same product. One is delayed by an earlier subprocess so that by the time it gets to subprocess: Authorise order it is automatically approved, because of the update. The other order is not delayed, and therefore has to be manually approved because it gets in before the update. Does this matter? Or should the effective date of the parameter update be compared with the date the order is received, rather than the parameter update taking effect at run time? If so the old authorisation limits would need to persist for a while alongside the new.

The impact of rule and control data changes will depend on business context. Questions like these are commonplace in administration, and are unproblematic when human beings provide the integration between 'work management' and 'administration' systems. The challenge comes with increasing automation – an architecture integrating business process with business functionality.

Consider for example the impact of an 'agile' change to the behaviour of the automatic underwriting task in *Figure 5* above. Perhaps we want to add another company (D) to share some of the underwriting on product 4, and also move the whole of product 1 from company B to company D.

In *Figure 10* below the 'U/W company parameters' file holds details about what company can underwrite what product, and rules about how to split underwriting on a product between more than one underwriting company. The 'Automatic underwriting' task will need to read this file to know which company to generate the underwriting request for. The 'Auto receive response' task may also need to read the file to validate an underwriting outcome.

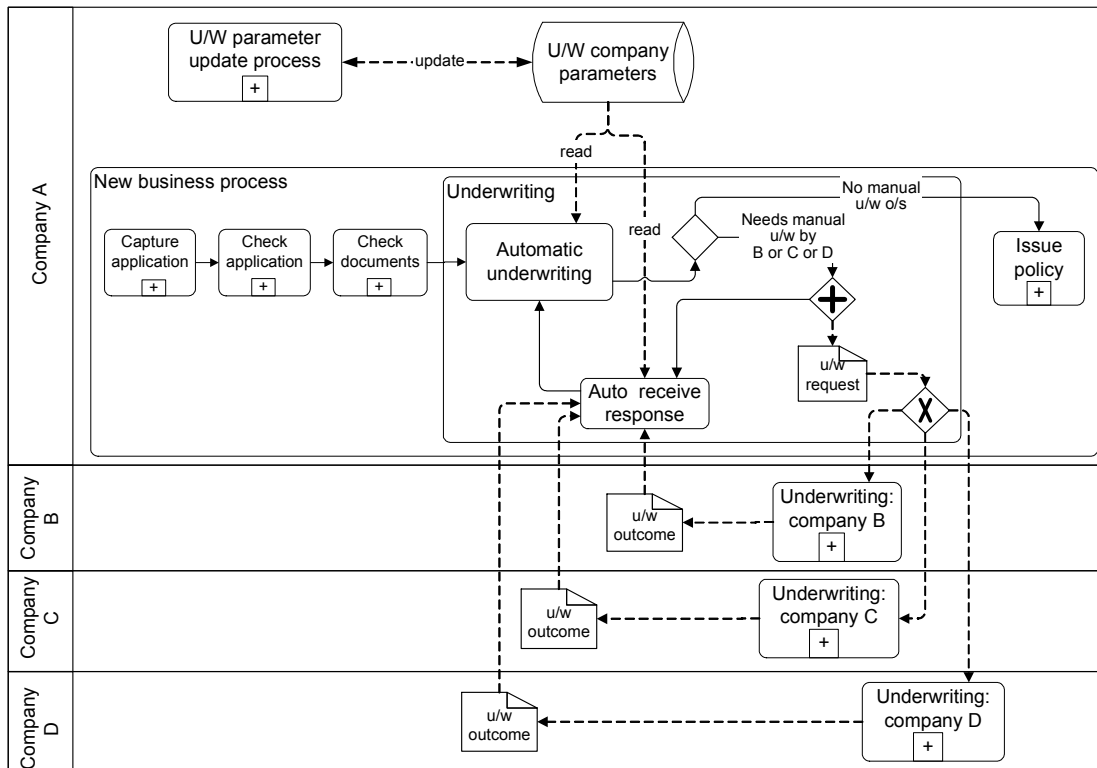


Figure 10

Company B may have received underwriting requests for product 1 but not yet completed them. So although, following the update, company B is no longer ‘allowed’ to underwrite product 1, an outcome from company B for a product 1 request issued before the update should be treated as valid.

But there could be more to it than this in a real business context. If company A changes its rules so that underwriting work for products 1 and 4 is now outsourced to company D, company D must expect to receive these requests, and companies B and C must also expect changes in their incoming requests. Do we assume the ‘U/W parameter update process’ handles some or all of this correspondence, or is it just an internal data update process? What does ‘just an internal data update process’ mean? What if the strategic intent is to implement as many business processes as possible seamlessly, transparently and explicitly in application technology, as this was seen to maximise effectiveness and efficiency?

The overall business process could be like this:

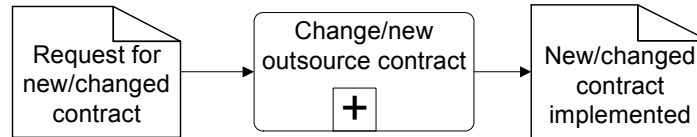


Figure 11

The ‘customer’ in this case is actually the supplier or potential supplier. Or perhaps the customer is internal to company A – a department responsible for contracting with underwriting providers. The point is that the ‘Underwriting parameter update process’ of *Figure 10* could be seen more holistically as part of a *business process* to set up a new outsource contract and/or change an existing one. (The ‘**big** difference’ between *system process* and *business process* is not so much an ontological distinction between surrogate and what the surrogate represents.²⁴ It is more a question of how holistically we choose to see the process.)

In our example there may need to be three instances of the process: one to remove product 1 from company B’s contract; one to set up a new contract for company D to be sole underwriter for product 1 and pooled underwriter for product 4; and one to change company C from sole underwriter to pooled underwriter for product 4. Other than the parameter update itself, the process could be responsible for generating and receiving correspondence: eg new/amended contracts for signature and return.

The amount of automation (and system support generally) appropriate for a business process like this has more to do with its economics than with its logic or purpose. Company A may have functionality to update underwriting company parameters in one of its IT systems, but no functionality other than that to support changes in outsource contracting. But its business processes should not be seen in terms of the systems it happens to have – unless of course they were successfully and completely process-architected, in which case the processes and their system implementation would be one and the same. I accept the idea that a data entity is a ‘surrogate’ for the real-world entity it represents, but it does not follow that

*any model of the business (including its processes)... [has to] ...differ from any model for the design of an information/knowledge system (including its processes).*²⁵

The reason for discussing scenarios like these is to stress that it is often not so much what technology do we need to maximise agility, but how sophisticated and exhaustive do we want (or can we afford) our impact analysis to be. Automation at any level, in any context, ushers in a new

²⁴ Ronald G. Ross, "How Rules and Processes Relate ~ Part 4. Business Processes vs. System Processes," *Business Rules Journal*, Vol. 7, No. 1 (Jan. 2006), URL: <http://www.BRCommunity.com/a2006/b265.html>

²⁵ Ronald G. Ross, *ibid.*

world. Run-time opportunities for human intervention and ingenuity are reduced by design, so the intervention and ingenuity must be supplied at design time.

People often expect two things from a process engine or a BPMS. One is 'process flexibility', in the sense of being able to change a process at the click of a button. The other is 'rule-based routing' – being able to store and change 'business rules' controlling how work travels through a business. Both are possible, but they come at a price.

We have spoken about the run-time impact of rule and control data changes, and said that process, subprocess and task themselves are best considered as data. With 'process' being treated as type and instance data entities manipulated by generic functionality, a useful approach to agility (and to constraints against agility) is by way of data-model crows' feet:

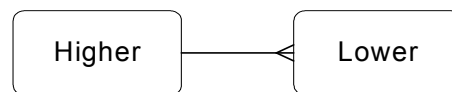


Figure 12

In general the 'higher' an entity is the more impact there is in changing an attribute value, as it may affect all 'lower' entities. In a one-to-many relationship the 'one' affects the 'many', but the 'many' tends not to affect the 'one'.

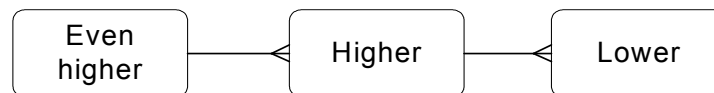


Figure 13

In *Figure 13* 'Even higher' will impact 'Higher', which in turn will impact 'Lower'. So 'Even higher' will generally indirectly impact 'Lower' as well.

A classic example of an 'Even higher' entity is Frequency, taking values like monthly, annual, half-yearly etc. Frequency is important in many business contexts but it is not a 'process' entity like 'ProcessType', 'TaskInstance' etc. Consider the impact of adding a new frequency, eg 'quarterly'. In many financial services and general commercial contexts this would not be possible without adding or changing functionality – eg for calculating charges and/or interest. Another complication is external impact: there may be an implicit assumption that an internal Frequency value is also allowable externally, eg for bank debit orders or standing orders. A new internally allowable Frequency value may not be allowable in all external contexts. Functionality changes may therefore need to be made to prevent 'unforeseen' consequences. In view of the position of Frequency in the data model the changes could be extensive.

The impact of removing a Frequency value could be even greater. If we suddenly make 'half-yearly' invalid, what happens to transactions that have

already happened at half-yearly frequency? We cannot pretend they never happened. There will also be customers who have been used to having (say) half-yearly statements and must now be told they cannot any longer.

The impact of changing processes is similar to that of changing (other) control data. A 'process change' can mean different things depending on what actual data is changing.

Often all that is intended is the sort of parameter change already discussed where an authorisation limit is increased. A change like this has 'process' implications, but they are relatively untraumatic.

Another fairly innocuous type of 'process' change would be a change to the Access entity in *Figure 6* above – for example removing access for a particular TaskType from one set of users and granting it to another set of users. Such a change could have big consequences: for example re-routing categories of work from one department to another. So it would have to be made in a coherent and controlled way, with full regard to training, management accountability etc. But it would be unproblematic from an architectural perspective.

What happens though if we want to change the process structure itself, for example the sequence of subprocesses in the order process in *Figure 1*? Some changes would make little sense, eg putting Check order before Take order – unless of course the change had a special meaning:

Check order = manual check that the order was correct and complete;

Take order = capturing the order in a system, which can only be done (or which we only want to do) with orders which have been manually checked to ensure they are correct and complete.

What if Despatch order came before Authorise order? This would mean implicitly authorising every order, and would make the final Authorise order redundant. Unless of course the supplier had a special relationship with its customers, who would be happy to return any order not authorised after despatch.

A more plausible example might be if Match against stock and Authorise order were to change places, as this might represent an equally valid (if more cautious) way of doing business. There are a number of areas of possible impact.

What about previous process instances – before the change? Are our systems, databases and people able to handle two types of history records: 'old' ones conflicting with the new process and 'new' ones matching it? Is there anything about any aspect of the business infrastructure which forces 'old' cases to be interpreted as 'new' ones? Consider a query from a customer who did not receive exactly what he ordered, and whether a

supplier employee might try to resolve the query differently if it was an 'old' case or a 'new' case. What if it was hard to tell 'old' from 'new'?

Another issue is process instances already started but not finished at the time of the process change. A sequence could be:

- 1 Order matched against stock.
- 2 Order process changed.
- 3 Order authorised.
- 4 Order matched against stock (again).

Easily resolved (in concept at least) by having the process change only apply to new orders. But this means either having two process models in existence for a time, or having to 'empty the system' of all 'old' orders first before making the change. This could delay 'new' orders while 'old' ones are completing.

Quite apart from timing issues there is the business impact of the change itself. In the 'old' process someone responsible for manual authorisation would know the order had already been matched against stock. We shall assume this means both making sure there was stock to fulfil the order and allocating particular stock to that order so that a subsequent order for the same material cannot be matched against the exact same stock. In the old process therefore an authorised order could be immediately despatched. If a customer queried where his order was a supplier employee could see the order was authorised (either automatically or manually) and confirm it was on its way. An order which was not authorised however would need to have its stock allocation de-allocated, to make it available for subsequent orders.

The new process has different logistics. The supplier employee answering the customer query cannot confirm the order is on its way just because it has been authorised, as the stock may not yet be available. An order which is not authorised however will not need stock de-allocated, as none would yet have been allocated.

Scenarios like these are fairly mundane but they are what process and process changes are about. The more automated and integrated process control is the more 'agility' is a matter of thinking through the logistics of what the intended process change actually means in practice. Not so much how agile can we be, and what technology solution will make us more so, but what do we need in place and what thinking and designing and testing must we do so that agility (like changing a few data items on the Routing entity in *Figure 6*) does not result in chaos.

USING THE META-MODEL

This section will indicate a few specific and practical ways in which the meta-model can be used to guide both process design and translation between different process representations and implementations.

Guiding process design

Much of the material in previous sections has been about this. To summarise: the model provides concepts and criteria for identifying processes, and for knowing where they start and stop and how they interact. The key is to define the request entity for each process. This determines the level the process is at (account, customer, claim etc); what constitutes completion (outcome); and what sequence and structure of rules are needed on the way to completion. A process may trigger other processes, and the triggering process may or may not be dependent on their completion.

The request entity will undergo status transitions related to the sequential and structured application of the rules. The meta-model guides the identification of these status transitions first at subprocess then at task level. It is therefore a coherent methodology for designing task structure. Once the tasks have been designed at a process logic level, the functionality they require can be designed and built (and/or found and adapted from existing applications). The details will vary depending on what solution architecture is assumed.

The request entity identifies an extended data set. The implications of the interactions between that data set and the rules of the process are what determine the process design. The request entity is therefore a crucial process design component. Treatment of process is methodologically aligned and intertwined with treatment of data.

Because a subset of tasks may be manual, the meta-model draws 'work management' into the same holistic logical architecture. (There are further implications for strategic change, benefits realisation etc which are explored elsewhere.²⁶)

Process translation

Examples of this are

to enable the integration of different process design products with different execution products or to allow simple migration of existing process definitions to a different design/execution product.²⁷

Because the meta-model derives from an analysis of what it is to be a business process, rather than what it is to be a workflow system or a

²⁶ Chris Lawrence, Make work make sense, Future Managers, Cape Town, 2005.

²⁷ David Hollingsworth (2004), *op cit*.

BPMS, it provides grammar and semantics to represent the essential structure of a business process at logical level. Because a process implementation is a *how* to the *what* of the logical representation, the grammar and semantics of the meta-model can also be used to represent the implemented process. How this might be done would depend on the implemented process and the intention behind the representation.

Translation involves mapping. The process meta-model will be a successful aid to translation if different process designs (including both conceptual and executable models) can map to it. The intention could be (i) to store the essential features of an implemented process, however successful or unsuccessful, in an independent language or format so as to support the implementation of that same process into different technology and/or architecture. A translation like this would need to go via a conceptual model. Or, more in line with Hollingsworth's diagram of a BPM component model²⁸, the intention could be (ii) to have a common conceptual model level capable of translation into different executable models appropriate to different implementation architectures. And in a 'B2B' context (iii) two or more organisations interacting at the process level might want to share an understanding of their interactions.

A first observation is that scenarios like these will rarely feature a single 'process' – with 'process' defined as the meta-model defines it. In all three scenarios the 'unit' will typically be of a set of linked processes, as this is more likely to be the scope of a practical business intervention than a single process.

In scenario (i) the essential features of an implemented process set need to be stored in an independent format so as to support a different implementation of that same process set. The implemented process set is effectively to be reverse-engineered into a conceptual model. Regardless of what the implementation was like (whether for example a single, explicit, generic, process engine controlled the process set in its entirety or, say, choreographed messaging was used) I would want to understand what the process set meant in business terms. This would involve identifying the request entities, and therefore the outcomes, and therefore where each process begins and ends. In the case of a messaging solution, it is possible that some of the messages and responses may be requests and outcomes, but perhaps not all. Also not every request and outcome may be explicit in the implementation.

However just because that was what I would want to do does not mean it has to be done. It depends on the objective behind the translation. A description of an implemented process set could, in certain circumstances, be translated completely mechanically into the terms of the meta-model. Anything identified as a process would be translated as a process, and anything which starts a process would be translated as a request. The

²⁸ David Hollingsworth (2004), *op cit*.

result may not be a very good conceptual model, nor would it in itself represent an advance on the previous description. But it would now be in a conceptual format where clear and coherent methodological principles could be applied to improving and optimising it.

Having identified the request entities and start and end points of the processes in the process set (either analytically or mechanically) I would then proceed as in the articulation of the meta-model, by identifying subprocess-level status transitions and then the task structure.

But again how analytical this step would be would depend on the translation objectives, and how much of a conceptual model or specification already existed. If there was no intention to critique or rationalise or improve the process set, then the as-is components and routing could be transferred one-to-one into manual and automatic tasks, perhaps with no attempt at logical analysis via subprocesses. Rules would not be exposed, they would just be assumed to be correct and correctly belong where they are in the implemented components: see *Figure 14* below.

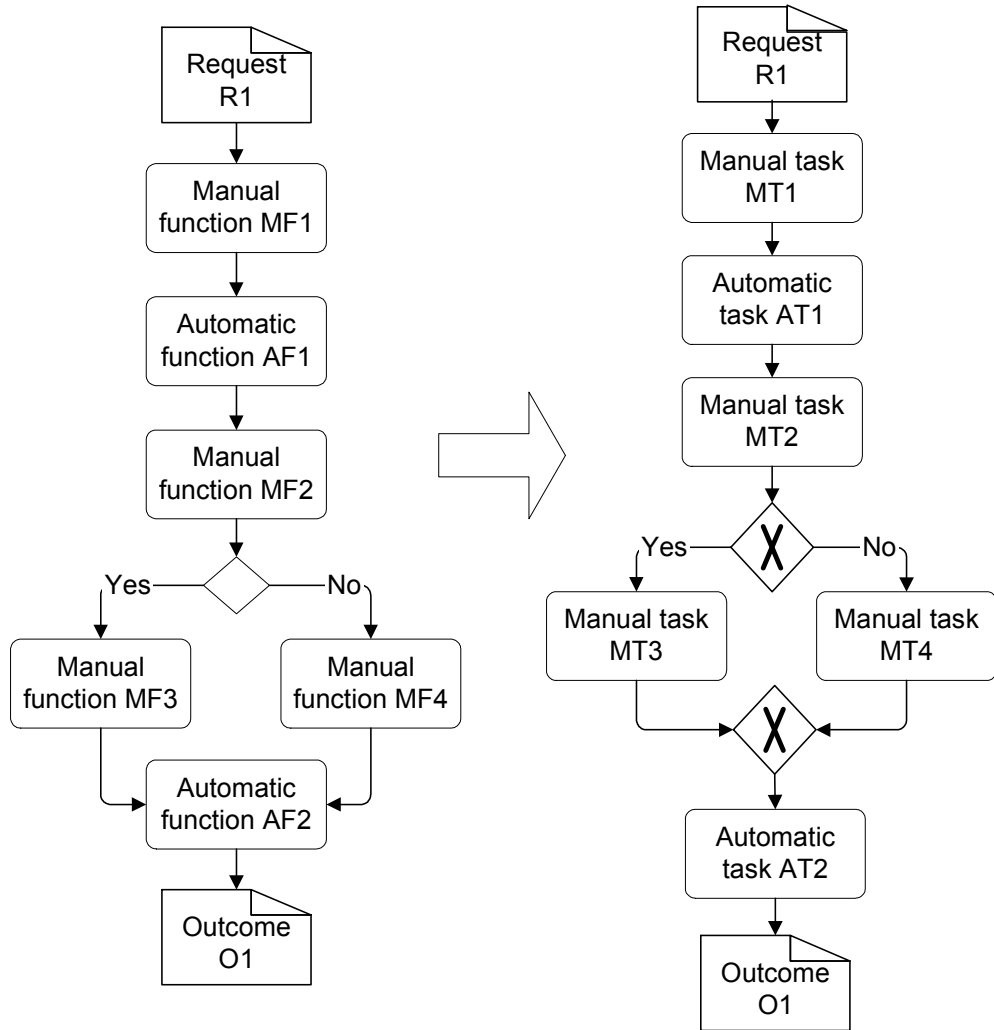


Figure 14

Task-Component	
Task	Component
MT1	MF1
AT1	AF1
MT2	MF2
MT3	MF3
MT4	MF4
AT2	AF2

Figure 15

Task-Component	
Task	Component
MT1	MF1
AT1	AF1
MT2	MF2
MT3	MF1
MT4	MF4
AT2	AF2

Figure 16

In accordance with the meta-model however the as-is components would not transfer across as tasks themselves: an automatic or manual task would correspond to each component, but with a 'Task-Component' (or equivalent) record linking it to the appropriate Task record (see *Figure 6* above and *Figure 15* above).

It may however be that the actual components for (say) tasks MT1 and MT3 are or could be identical. In that case the correspondence could be rationalised as in *Figure 16* above.

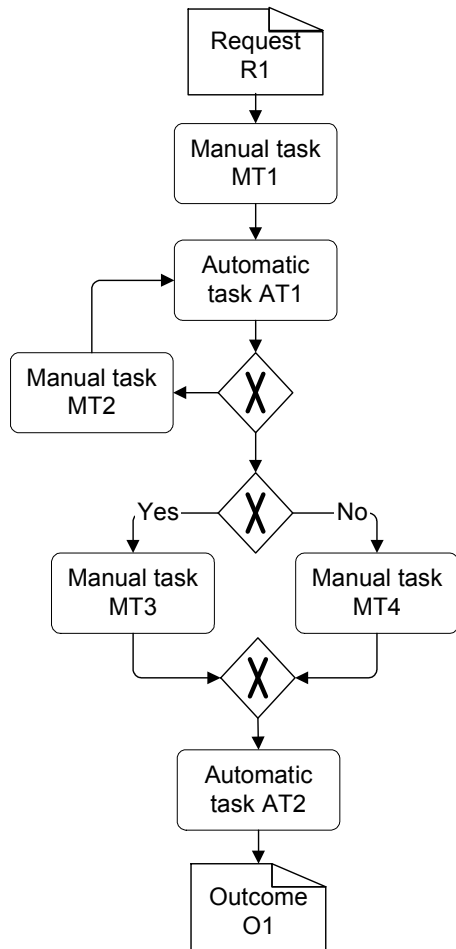


Figure 17

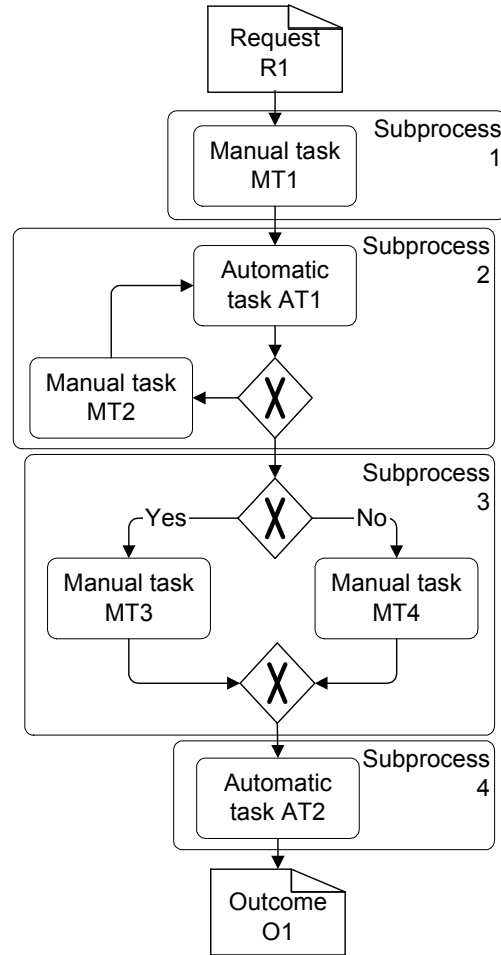


Figure 18

Further changes could be made, depending on whether there was any need or intention to improve or optimise the model. For example it may be realised that the interaction between implemented rules and process flow might be improved by altering the configuration of AT1 and MT2 as in *Figure 17* above.

It might then be decided that further improvement might be served by recognising a subprocess level, as in *Figure 18* above. But none of these changes are mandatory, and they can be made incrementally. The extent of the rationalisation would depend on the scope and objectives of the exercise itself.

Reverse-engineering a logical data model from a physical file structure is often an occasion for critique – eg discovering that a repeating group inappropriately limits entity instances, or that denormalisation compromises data integrity. In the same way deriving a rigorous conceptual

model from an as-is implementation will often expose anomalies, incompleteness and faults. For example a common omission in process environments based on proprietary workflow implementations is that automatic tasks are under-represented, awkwardly or inexplicitly engineered, or just plain absent. This can happen when 'process' is seen in terms of sequences of manual activities rather than sequences of rules.

Obviously, the more the reverse-engineering is mechanised the less opportunity there would be for this.

Access	
Task	User
MT1	Clerk1
MT2	Supervisor
MT3	Clerk2
MT4	Clerk2

I would express this conceptual model in BPMN – assuming BPMN continues to emerge as the standard – but in the knowledge that the same information could be expressed in a different format, eg in a populated database (see *Figure 6* above). One implication of using BPMN for the meta-model is that process, subprocess and task all use the same symbol. Therefore each box should be carefully named to indicate whether it was a process, subprocess or a task.

Figure 19

Again depending on the objectives of the exercise there are possible extensions. One would be to document human actors and/or roles involved in the process – simply by populating entities equivalent to 'Access' and 'User' in *Figure 6* above. *Figure 19* above shows a possible population of 'Access'.

Taken together these two extensions would be

*specifying the work resources associated with the process work items, or activities.*²⁹

Much of what has been said is also relevant to scenario (ii), except that reverse-engineering does not apply. For that reason it should be safe to assume an intention to get as 'pure' a conceptual model as possible. It should therefore not be too simplistic to see translation into different executable models as *in theory* mostly a matter of different population of entities like 'Access', 'User', 'FunctionalityComponent' and 'Task-Component' (*Figures 6, 15, 16 and 19*).

Adopting a meta-model is ultimately a choice to see a particular domain in a particular way. So in the B2B scenario (iii) it is for the participants to decide how they want to see their process interactions.

²⁹ David Hollingsworth, *ibid*.

Although the shared scope will also typically be a set of linked processes, we need not and cannot assume there will be one conceptual model which both represents that set of processes and is shared by the participating organisations. It is more important that where organisations interact, they understand those interactions in the same way, and that the interactions make sense to each participant in terms of its own individual process model.

This is more likely if the participants share the same meta-model, either explicitly or implicitly (by subscribing to the same agreed standards).

Figure 4 and *Figure 5* above show two different ways in which the participants might interact. It is worth analysing these diagrams to draw out the implications for 'meta-model sharing'.

In *Figure 4* the process in a sense 'belongs' to company A. A familiar implementation option would be where the new business process from Capture application to Issue policy is defined in a workflow or process management application belonging to company A. The two tasks 'Manual underwriting: company B' and 'Manual underwriting: company C' would be part of that same application belonging to company A. It is just the users with access to the two tasks that belong to company B and company C respectively. Even if the linking between the tasks making up the new business process was based on (say) messaging, then *Figure 4* could still apply: the diagram does not assume any particular technology.

It does however make an ownership assumption. Even if the process control technology used messaging, *Figure 4* does assume company A owns (supplies, defines, supports etc) the two manual underwriting tasks. Companies B and C are suppliers to company A but (in this context) they do not supply by way of a business process. They supply resources. The mechanism by which this was set up was no doubt a business process (eg contracting), but in *Figure 4* companies B and C are not receiving a request from company A which they are meeting by starting a process. Instead they are participating in company A's process, as that is the agreement.

Figure 5 could also be physically implemented using messaging technology, but the relationships between the interacting companies are different.

As far as 'meta-model sharing' is concerned, in *Figure 4* companies B and C may or may not share, understand, or subscribe to the process meta-model company A employs in its business architecture. As long as B and C supply underwriters and fulfil their service agreement, the process will work. The scenario in *Figure 5* however would be helped if companies A, B and C shared the same process meta-model at least implicitly, by conforming to the same standards. It would be even better if the participants explicitly shared, and knew they were explicitly sharing, the meta-model which underpinned those standards and provided their rationale.

Two organisations interacting commercially generally share an understanding of the double-entry paradigm. Communication would be more difficult if not. On a slightly less universal level, communication between two organisations transferring data between each other is greatly eased if they can each assume they both understand the principles of the relational model in the same way.

Two organisations interacting with each other's processes are more likely to find those interactions go smoothly if, implicitly or explicitly, they subscribe to the same process meta-model. At the very least, seeing interactions in term of requests and outcomes means identifying 'customer' and 'supplier' roles and, by extension, ownership and accountability. In a business process outsource (BPO) context the sharing of a meta-model is almost mandatory, not only between the BPO provider (who provides the white-label process shell) and its clients; but also between the separate clients who all make use of the same process shell as if it was their own.

SUMMARY

It is possible to derive an effective logical and generic meta-model for **business processes** from the concept of a **request**.

The request is implicit or explicit, and from an internal or external **customer**. Because it is for a specific **outcome** and of a specific type it unambiguously determines the end-to-end process. The process starts with the request and ends with either the outcome the request was for or some alternative outcome according to the **rules** of the process.

By applying process rules to the request **dataset** regardless of the range of possible attribute values the process is analysed to **subprocess** level. Each subprocess is a **business status transition** of the request. So is each **task**. The task structure is defined for each subprocess from the minimum set of status transitions required for each possible extended request dataset to achieve the status transition of the subprocess. (The subprocess is important for analysis and design, but once the process has been designed to task level, the subprocess level is technically superfluous – although valuable for management information, control and reporting.) As status transitions process, subprocess and task are primarily **control units**. Conceptually at least the task would be **loosely coupled** to any **functionality component(s)** which implement it.

Process components (request, process, subprocess, task) are considered as **data** – both **type** entities (RequestType, ProcessType etc) and **instance** entities (RequestInstance, ProcessInstance etc). The **process model** of an organisation is defined at type level (ProcessType, SubprocessType, TaskType, plus all **routing** and interactions), and is itself ultimately a structure of rules.

Processes can **interact** with each other (and with themselves) in a number of ways. A general and obvious type of interaction happens when functionality in one process (instance) updates data which then influences what happens in another process (instance). But also a task in one process can generate a request which initiates another process – either a new instance of the same process type or an instance of another process type.

The meta-model is independent of any implementation technology. However since it is a logical **business process architecture**, it can guide the successful design and implementation of process-architected business solutions.³⁰ Different implemented processes and process-modelling formats can also be **mapped** to it, because it represents the underlying *what* of the business process.

³⁰ Chris Lawrence, Make work make sense, Future Managers, Cape Town, 2005.