6 Preparing a process architecture

INTRODUCTION

A constant theme of the *Riva* approach is the central place of concurrency in organisational activity: the fact that when we walk into the organisation's building we are surrounded by many instances of many processes all operating at the same time, some activating others, some interacting, and so on. At any one moment there is a network of interacting process instances at work. If we want to get our arms round all of that activity, to 'chunk' that activity into the constituent processes, how should we go about it? This chapter describes how to prepare the *process architecture* of the organisation, the picture that says what process types there are in the organisation and what their dynamic relationships are. Remember that I am not interested in static hierarchical decomposition – the world is not a neat hierarchy fixed in time – it is a dynamic network of interacting instances.

Getting this process architecture right is vital.

A poor division of organisational activity into processes can easily lead to unnecessarily complex designs or models. There is a lesson to be learnt here from the world of software design: when we invent software modules we look for what are called high 'cohesion' and low 'coupling'. The idea is that each module should contain closely related activity: it is coherent in its content. The result of doing this is that the bandwidth of the relationships between it and other units is reduced: the system is loosely coupled. This in turn means that we keep the interconnections between modules to a minimum, thereby reducing complexity, and hence increasing maintainability and reliability. Software of course is synthetic: how it looks and how it is structured is entirely up to the software designer. A business process is typically not synthetic: it has developed organically and is the result of some design and a lot of chance. We cannot expect it to be tidy. But when we chunk the organisational activity we can expect to look for 'natural' or latent modularity: chunking that is there because it has to be, because the organisation is in this particular business.

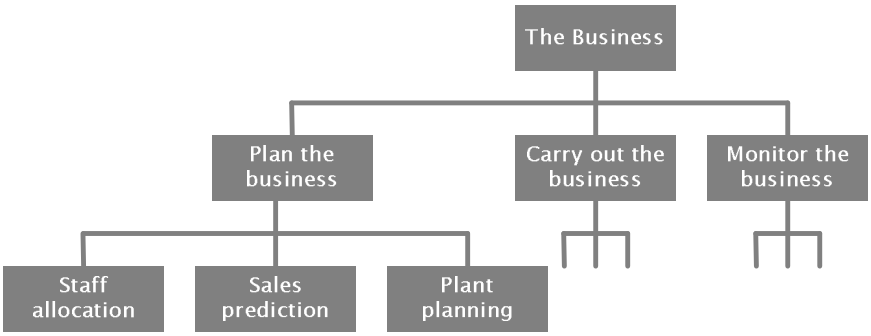
It is as if we are looking for natural fault lines in a rock when carving, or splitting wood with the grain rather than across it. When we model the real world, if we break the activity 'with the grain', we shall get fewer broken connections. We want to exploit any cohesion present in the real world. Our aim must be to find that natural grain.

If we are considering re-engineering, a bad initial division can at best obscure the possibility of a radical change to a process and at worst lead to the local optimisation of processes at the expense of the organisation. We also want to be able to distinguish between processes that are there because we are in a particular business and processes that are there because we have chosen to work in a particular way. The first group arise from what we shall call

'essential units of work', and the latter from what we shall call 'designed units of work' – the latter are of course candidates for re-engineering.

People (especially those of us with software engineering backgrounds) have a natural desire to decompose a process into successively smaller sub-processes, in other words to draw up a hierarchical structure. But the world is rarely constructed in such a neat way and the processes in an organisation are invariably connected in a network, rather than being contained in one another. The danger of thinking hierarchically cannot be over-emphasised. To draw a diagram such as the one in Figure 6-1 is not to prepare a process architecture. What does it mean to say, as the diagram implies, that one process is the 'sum' of three others? Where in the figure are the *dynamic* relationships between processes represented? It is, after all, the dynamic relationships that are crucial to an understanding of how an organisation *works*. It is network dynamics that matter, not some arbitrary hierarchical composition.

Figure 6-1 – Not a process architecture – more a random hacking



It is equally all too easy to draw up a list of 'major' processes in an organisation by looking for functional units in the organisation and imagining that they in some sense represent 'top level processes', e.g. the 'Finance process', the 'Analytical Chemistry process', or the 'HR process'. These are all meaningless and probably misleading titles that must be resisted at all costs. If we change the organisation's structure, its list of processes would change too, which would be absurd if it was still in the same business.

Figure 6-2 is an equally tempting 'chunking' of activity. At least in this picture there is a sense of dynamism, of the order of things. But it is hopelessly serial, and all too easily becomes just a list of silos.

Figure 6-2 – Not a process architecture – more a list of silos

Surely, what we are looking for is *a process architecture that is derived solely from an understanding of what business the organisation is in*. We would like to say 'If the organisation is in this business then it must have these processes.' 'This is the grain of the organisation.' If the organisation changes the business it is in, then we would expect the list of its processes to change appropriately. But if the business reorganises itself, perhaps from a hierarchical to a matrix organisation, why should the list of processes change? Clearly, how some of those processes are *done* might change, but not the list itself. If the business changes its culture from being production-focused to customer-focused, why would the list of processes change? If two departments are merged, why would the list of processes change?

We shall see that a *Riva* process architecture is an *invariant* for an organisation that stays in the same business. This makes it a particularly secure place to start any process design, or improvement activity, or computer system design. Even if I am asked to look at what is perceived to be a single process, I insist on preparing a process architecture in order to ensure that we have the right chunking and that what appears to be one process is one process rather than parts of several related processes arbitrarily lumped together. Never forget the axe.

Before we move on, let's remind ourselves of how we shall be using the word 'organisation': an organisation might be Jane and John, a project team, the HR Department, the sales force together with the production planning group, the Bristol branch, the four branches in Somerset, the entire company, or our market place and all its players. It's whatever we want to look at.

WHAT BUSINESS IS THIS ORGANISATION IN?

Essential business entities

Our first step in constructing the process architecture is to *characterise the business that the organisation is in*. We shall do this in terms of its *essential business entities* (EBEs).

In any business there are things, *entities*, that one cannot get away from. They are there simply because of the business the organisation is in.

For instance:

- In a pharmaceutical R&D company we would recognise *Drug compound*, *Clinical trial*, *Assay*, and *Batch of raw compound*. As long as the pharmaceutical company is in the business of developing compounds that are subject to regulatory control, these things will exist: they are the *essence* of the business.
- If we are in the business of administering a modular programme in a university faculty, candidates for EBEs would be things like *Module*, *Award*, *Student assessment*, *External examiner*, *Curriculum*, and *Appeal*. Take any one of these away and the business changes – or the faculty cannot do its allotted business.
- If we are a water utility, we shall think of things like *Customer*, *Supplied property*, *Meter*, *Asset*, *Job*, and *Customer contact*.
- A firm of consulting engineers will have things like *Customer*, *Project*, *Market sector*, and *Expertise area* on its list.
- A car repair shop will have *Customer*, *Appointment*, and *Item of equipment*.

An EBE can be something physical and concrete such as a batch of drug compound (you can stick your finger in it in the barrel once it exists), or something rather abstract such as a clinical trial (you could see a clinical trial going on but you couldn't touch it), or something entirely abstract such as a request to change a clinical trial (e.g. 'I have decided to double the strength of the tablets'). The entirely abstract EBEs often prove to be very important, as we shall see.

(The older among us will hear echoes of Michael Jackson's JSD (*System Development*, M. Jackson, Prentice-Hall, 1983). A JSD *real-world entity* must 'perform or suffer actions, in a significant time-ordering, exist in the real world … and be capable of being regarded as an individual.' Much of what follows has strong parallels with JSD's concepts.)

Sometimes, something that at first glance feels like an EBE is there only because of the way the organisation has decided to do its business. How should we react if we find *Invoice* on the list of candidate EBEs? Someone in the room will argue that we can't survive unless we send out invoices, we shall run out of money, and go bankrupt: 'They're essential – *Invoice* must be an EBE.' For a car manufacturer, *Invoice* is not an essential entity; it is a *designed* business entity. The car manufacturer is not in the business of invoices. On the other hand, *Payment* is essential and not designed. Invoices might be the way the manufacturer has decided to obtain payment: we must get paid, but how is a matter of choice. However, for the Invoice Handling Department – which *is* in the business of handling invoices – *Invoice* is certainly an EBE.

An EBE is called 'essential' because it is part of the essence of the business. Unless we are in the Invoicing Department, invoices will not be what we are about, we are not in business to handle invoices, invoices are not our subject matter, they are not what we are in business for. They are not 'essential' in that meaning of the word. And, of course, we can also argue that they are not essential in the other meaning, that 'we have to have them'. They are, after all, only a way of requesting payment, and we can think of many other ways of doing that. The fact is that we have *chosen* invoices as the way we will ask for payment. Invoices are certainly business entities for us, but they are designed business entities, and not essential business entities – they are there because of the way we choose to do our business, rather than because they fundamentally characterise our business.

This leaves us with a modelling decision. How should we handle designed business entities? The answer is to consider them, whilst noting that they are designed entities, and to replace them – where possible – by the essential business entities they stand for or implement.

Think of the list of EBEs as a searchlight that illuminates the part of the world we are interested in. We can focus in on the HR Department, or expand out to the entire company. We can illuminate both the pharmaceutical drug company and its partner companies in clinical research, or we can restrict our focus to the drug company alone. We can concentrate on some part of our 'internal' world, or we can extend our view to cover what our customers are concerned with. Indeed, whilst we might start by thinking about a particular organisation and then prepare a list of its EBEs, we shall invariably find ourselves revisiting exactly what 'organisation' we want to cover and then redrawing the boundaries differently in order to answer the questions we want to answer.

EBEs are a powerful way of focusing our thoughts on the things that matter, and avoiding analysis paralysis.

KEY POINTS

The process architecture must capture the network of concurrent activity in the organisation.

It should be derived solely from an understanding of what business the organisation is in.

We start the analysis and characterisation of that business by examining its essential business entities.

An essential business entity is an entity that is the essence of the organisation's business.

An essential business entity is part of the subject-matter of the organisation.

Finding the EBEs

Since the EBEs are the subject matter of the organisation, anyone who knows what business the organisation is in could – in principle – quickly give you a list of EBEs. So let's get half a dozen of those people in the room to brainstorm the list.

In true brainstorming fashion, the list will quickly build to thirty, forty, fifty, a hundred candidate EBEs. We can prompt suggestions with questions such as:

- What do we make?

  Cars, packs of biscuits, radios, furniture, bottled drinks, …

- What do we sell?

  Cars, palettes of packs of biscuits, water, electricity, insurance policies, items of furniture, packs of tablets …

- What product lines do we have?

  These models of cars, these ranges of biscuits, these designs of furniture, …

- What services do we offer?

  Giving road-side assistance for a vehicle breakdown, responding to an emergency call, answering a customer complaint, …

- What service lines do we have?

  These types of insurance policy, these levels of maintenance service, these levels of call-out service, these types of financial portfolio management, …

- What things can we simply not get away from?

  We are developing pharmaceutical drugs, so we cannot get away from the regulatory authorities.

  We build aero engines, so we cannot get away from the safety regulator.

  We are a quoted company, so we have shareholders.

  We have staff members.

  Company policy requires us to follow certain quality standards.

- Who are our external customers?

  Car buyers, car dealers, car wholesalers, fleet car buyers, …

- Who are our internal customers?

  Researchers, project managers, staff members, the Board, …

- Are there things that our customers have, or want, or do that are EBEs for us?

  Complaints, purchases, overdrafts, accounts, loans, loyalty cards, …

- What things do we think differentiate our organisation from others in the same business?

  Our quality focus. Our culture. Our expertise. Our prices. Our brands. Our customer focus.

- What sorts of things do we deal with day in, day out?

  Car engines, flour suppliers, drilling machines, power stations, customer complaints, machine failures, quality standards.

- What events in the 'outside world', the world outside our organisation, do we need to respond to?

  Power failures, drain collapses, customer complaints, significant share price changes, new financial years …

- What business entities are listed in our corporate data model?
- What things do our information systems keep information on?

Once the flow has dried up, we revisit each item on the list and apply a number of filters that test whether it is truly *an entity that is the essence of the business*. I like to keep the entire list of candidate EBEs as we move forward, whilst bracketing out – rather than deleting – those that don't get through the filters, and noting the reason why each has been bracketed. This is going to be an iterative process and we are certain to revisit this list later, perhaps reinstating a candidate, or adding new candidates. So it is a good idea to keep all that work and all those decisions as the work proceeds. Here are the filters:

- Since these are all supposed to be *entities*, test each by putting the word 'a' or 'the' in front of each suggestion. If it doesn't make sense, bracket it and think again: are there any other entities that are suggested and that do pass the test?

  This can cause consternation in some situations. If we are a water supply company, then 'water' surely must appear on the EBE list, yet you can't talk about 'a water'. Or, indeed, 'an electricity' if we were an electricity supply company. Have faith – bracket it and keep going.

- Bracket any designed entities.

  Invoices may be the meat and drink of the Invoicing Department and hence an EBE for them. But for the company as a whole, they are not: the organisation is not in business to issue invoices. Invoices, for the organisation, are just a way of obtaining payment.

- Bracket entities that are simply roles, and which are not 'of the essence' of the business.

The Accounts Department is an entity, and it's about the business. Someone (possibly from the Accounts Department) will say 'This company wouldn't function without the Accounts Department – they're essential.' Well yes, they are essential, but not 'of the essence': we are not in business to do accounts, we make cars. Accounts play an important role. But they are not an EBE, though they have a role.

WHICH EBES REPRESENT WORK FOR THE ORGANISATION?

Filtering off units of work

After the first filters have been applied to our brainstormed list of candidate EBEs, we shall have reduced the list to perhaps half. The items remaining are true EBEs. These are the things the business has as its subject matter. They define and characterise the organisation.

Our second step is to decide which of these are entities that have lifetimes during which we must look after them. We shall call these *units of work* (UOWs). Essential business entities can be *essential UOWs*, and designed business entities can be *designed UOWs*.

For instance, a clinical trial is a unit of work for an R&D pharmaceutical company: it starts, proceeds and stops, and we must look after that life. A drug compound is a unit of work: it is invented, tested and developed, taken to market, and finally withdrawn, and we must look after that.

We need a further set of filters to help us whittle the EBEs down to just those that are units of work:

- ☞ Bracket EBEs that are clearly not units of work for us.

  A *Purchase* of a theatre ticket would be an EBE for a Box Office and a UOW. A *Ticket* might be on our list of candidate EBEs, but we can bracket it because the ticket itself does not have a separate lifetime of its own of interest to us: we don't care how it is designed, printed and distributed. It's just a mechanism we use in the contract – it *stands for* a successful purchase and the right to occupy a seat at a performance. ('Ah, did we have "Performance" on the list? Is that a UOW for the Box Office?')

- ☞ Bracket EBEs that are not units of work for us, even if they are for someone else.

  A 'quality standard' clearly has a lifetime. If we are the central Quality Management Group, with responsibility for the quality standards that make up the Quality Management System, then quality standards are our meat and drink and that lifetime would certainly represent a unit of work for us: we shall decide on the need for a standard, draft it, have it reviewed, approve it, distribute it, make changes to it, and finally withdraw it. But for somebody who is only required to use quality standards, they may be an essential business entity but they are not a unit of work. Standards will only feature as controls in their processes, and will not be the subject matter of their processes.

- ☞ Bracket EBEs that are only roles that play a part in processes.

  The Safety Regulator is an essential business entity of, say, a company producing railway signalling systems. But there is no sense in which the *lifetime* of the Safety Regulator is something that the company concerns itself with: it does not have to look after the Safety Regulator in that sense. Clearly though, the Regulator will play a role in many of the company's processes, and we will expect to see it as a grey box on RADs.

It's quite typical for the list of candidate EBEs to contain all sorts of job titles and posts in the organisation: CEO, Project Manager, Salesperson, and so on. True, the business could perhaps not operate effectively without the CEO but the CEO does not have a lifetime that we need to handle. If we cannot imagine having a process to look after that role during its lifetime then we bracket it.

- ☞ Bracket any EBE that is only part of another EBE and does not have a separate lifetime of its own.

  If we manufacture DVDs, then *DVD* is probably an EBE and a UOW. But *Jewel case* is probably not a UOW for us. Yes, each compact disc we make goes into a jewel case, but we don't care about the lifetime of that case: we just buy them in. The disc itself does have lifetime however, from the moment it is a lump of molten plastic to the point where it is capable of being played in a DVD player.

Some EBEs might (or might not) turn out not to be UOWs themselves, but might point to other UOWs. This is commonly the case with collections of things that form another thing:

- ☞ As an electricity distribution company, we regard (the) *Transmission System* as an EBE – without it we are not in business – and there is a very real sense in which it has an (unending) lifetime that is of interest to us. But perhaps more interestingly we realise that the *Transmission System* is a collection of assets and *Asset* should be on the list too. So we add *Asset* to the list and show both as UOWs.
- ☞ *Our expertise* is in the list. Is it helpful to think of the lifetime of (the) expertise that we have? Probably not. However, *Our expertise* is made up of a number of individual expertises, and each of these will be chosen, developed, fostered, and perhaps finally dropped. So *Expertise* on the other hand does have a lifetime of interest to the organisation: that is a UOW.
- ☞ As a pharmaceutical development company, we are very concerned to have a good 'pipeline', i.e. a good stream of potential new drugs going through research and development. That *Pipeline* will appear on our EBE candidates list. But so will *Drug compound*. We can view both as EBEs and also as UOWs. Of course, we have only one pipeline, but that pipeline is composed of a number of compounds.

Finding 'unseen' UOWs

As in any requirements-gathering exercise the question arises: 'How do we know when we have finished? How do we know when we have everything we should have?' We have a number of ways of checking for units of work that we might have missed:

1. Examine the names of departments and groups.

   Individual departments often exist to deal with one sort of unit of work: for instance, we might observe that the Analytical Chemistry Department deals with assays. Is 'Assay' an EBE and a UOW perhaps for the organisation? The Emergency response Team deals with … emergencies. The Help Desk deals with fault reports. We need to be careful when finding UOWs this way: they might well be designed UOWs.

2. Put the words 'Change to' in front of each candidate UOW and see if it creates another UOW.

At one pharmaceutical company we found people who thought their unit of work was dealing with requests from clinicians for the patient packs (of drugs) for clinical trials. When we looked, we found that they did indeed do that and had a UOW called 'Request for supplies for a clinical trial'. But additionally – and very importantly – they spent a great deal of time dealing with changes of mind from clinicians, so they had a further unit of work which was the 'Change to request for supplies for a clinical trial': a change arrived, it had to be dealt with, and finally had to be incorporated into readjusted schedules. For each original request there might be several changes.

3 Put the words 'Collection of' in front of each candidate UOW and see if it creates another UOW.

Is there a sense in which the collection of things has its own existence? Like the portfolio of drug compounds in a pharmaceutical company, or the product range of a software application company, or the publisher's list of books in print. A publisher doesn't just deal with titles in isolation: it will build a list of a particular character and content. In other words, the list has its own existence and lifetime, apart from the titles that make it up.

KEY POINTS

A unit of work is an EBE that has a lifetime during which we must look after it. These are essential units of work.

Other units of work arise from designed business entities rather than from essential business entities. These are designed units of work.

Some units of work are collections of other things or changes to other things.

Tips for the process architecture brainstorming

- ☞ Work on flipcharts so that you have a full record of what happened. Whiteboards invite rubbing out and reduce the ability to backtrack earlier decisions. This also helps people to remember how they got to where they got and why.
- ☞ Treat the brainstorming of candidate EBEs as true brainstorming: write them down as they are shouted out, without discussion. The one exception I make to this rule is to apply the 'a/the' rule to check that something is an entity.
- ☞ When starting the filtering for real EBEs, start with easy ones to help people get their heads round the idea. If we make cars, then *Car* is a true EBE. If we design cars, then *Car design* is a true EBE. But *CEO* is clearly just a role.
- ☞ Do the same when filtering for EBEs that are also UOWs. If we make cars, then *Car* is a UOW. If we are in a regulated industry, the *Regulator* is a true EBE but not a UOW.
- ☞ Draft a sentence describing each UOW, in particular to capture its scope. This is an aide memoire for future work and future readers.

WHAT ARE THE DYNAMIC RELATIONSHIPS BETWEEN UOWS?

Our list has now been whittled down to those true EBEs which are also UOWs. The next step is to examine the relationships between those UOWs.

Now, we can think of all sorts of possible relationships between UOWs: the annual product portfolio review process (Carry out the annual review of the portfolio) perhaps needs information from the process for collating the annual accounts (Prepare the annual accounts). That's an information relationship; these two processes will therefore need to interact to exchange information. The question is 'which relationships are interesting?' Because we shall be interested in the *dynamic* relationships between the processes, we shall want to concentrate on the *dynamic* relationships between the UOWs.

When we were examining the different dynamic relationships that can exist between processes, we saw that relationships arise when some units of work 'need' other units of work. For instance, candidate drug compounds require clinical trials to be carried out. During the development of the compound many clinical trials will be required. There will be times when several clinical trials will be in progress for the same compound. Every clinical trial requires patients. They have to be recruited; they have to be given the drug, a placebo, or a comparative drug; and they have to be monitored and recorded. At any time during the trial many patients will be participating in it. In order for a patient to participate in a clinical trial 'patient packs' have to be prepared that contain the doses of whatever they are receiving for the trial. Since there are many patients in a trial – potentially thousands – there will be a need for many patient packs.

Compound, *Clinical trial*, *Patient*, and *Patient pack* are all units of work in the pharmaceutical R&D company. And there are important dynamic relationships between them. We summarise this sort of relationship with the neutral word 'generates':

A candidate drug compound generates clinical trials.

A clinical trial generates patients.

A patient generates patient packs.

A project generates work packages.

A customer generates customer orders.

A staff member generates expense claims.

An order generates batches.

The word 'generates' perhaps jars in some cases, but I use it as a catch-all word covering concepts such as 'need', 'require', 'call for', and 'activate'. And when I say that '*A* generates *B*' I mean 'during the lifetime of a case of UOW *A*, cases of UOW *B* are needed/called for/…' So, during the lifetime of a clinical trial, patients are needed. During the lifetime of a customer, orders are generated by that customer. During the lifetime of the portfolio, (new) products are considered. And so on.

In some cases, the lifetimes of the cases of UOW *B* are all contained within the lifetime of the case of UOW *A*: clinical trials must have been completed before development of the compound can be completed. Sometimes the generated cases of *B* live on, after the generating case

of *A* has finished: a project will generate invoices to the client, and their handling can linger on after the project itself has finished. We might allow the project to close even though the processing of its invoices continues after the project is declared to be finished.

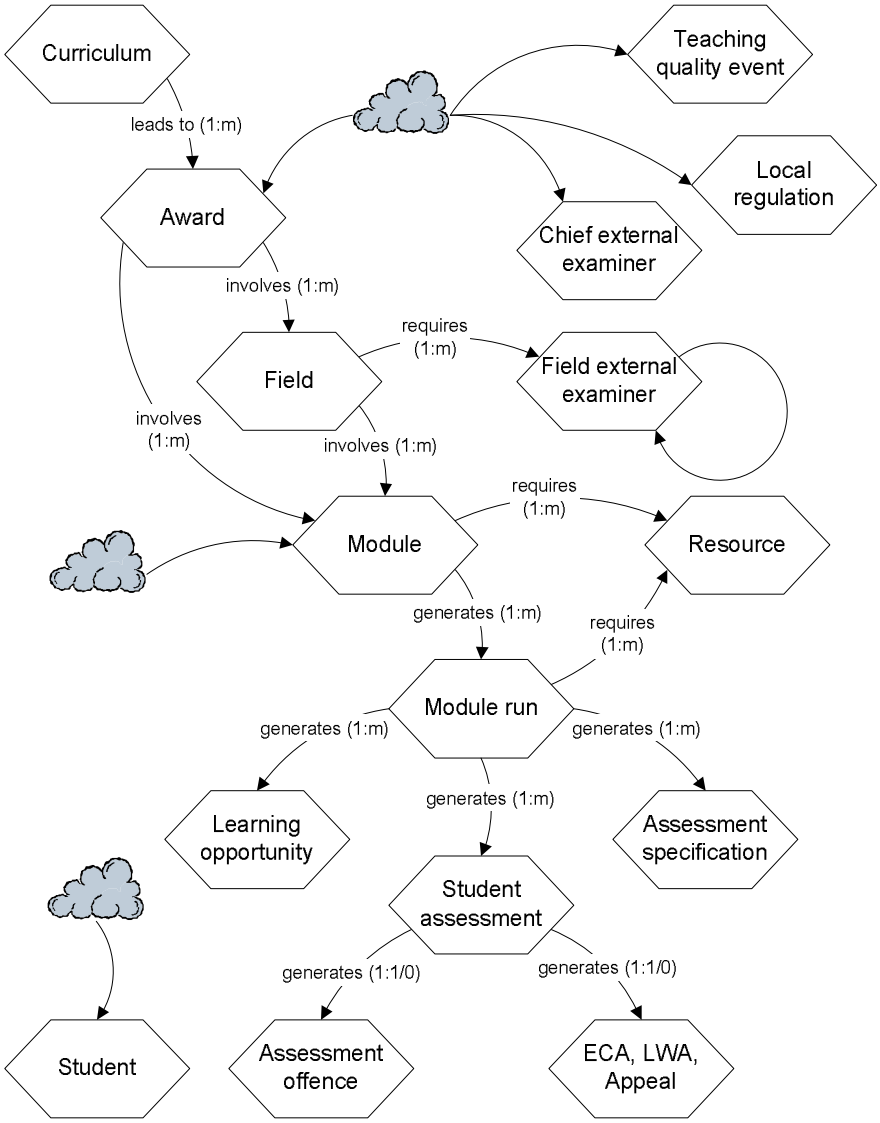
The 'cardinality' of the relationship between two UOWs can vary: each case of *A* generates exactly one case of *B*, or each case of *A* generates none, one or many cases of *B*. One clinical trial generates one clinical trial report. One clinical trial generates many patients.

Our third step in building the process architecture is to go through the filtered list of UOWs and draw out these dynamic relationships, the 'generates' relationships. We construct a *UOW diagram* in which we show the UOWs and the dynamic relationships between them; we name each relationship and identify its cardinality (one-one or one-many). For example, 'A Compound requires *many* Clinical trials'. We use the following conventions:

- ☞ Each UOW is shown as a hexagon containing the name of the UOW in the singular.
- ☞ Each 'generates' relationship is shown as an arrow from the generating UOW to the generated UOW, and appropriately labelled.
- ☞ Where a UOW is generated by an agent outside the organisation we are concerned with we show the arrow coming from a cloud suggesting 'The Outside World'.

Figure 6-3 gives an example of the sort of UOW diagram we might produce. It shows the units of work that characterise the administration of the teaching programme in a university faculty and their dynamic relationships.

Figure 6-3 – The UOW diagram for a university faculty administration



KEY POINTS

The UOW diagram shows only dynamic relationships between UOWs.

In particular, it shows only 'generates' relationships, where one type of UOW arises because of and during the lifetime of another.

No other relationships are drawn on the diagram.

Later, next to the water-cooler

Pupil: I'm worried that in your UOW diagram for the university faculty some UOWs are not connected to others. *Student*, for example, is just sitting on its own. And *Teaching quality event* and some others are generated 'out of the blue' but don't generate anything else.

Tutor: Do not be alarmed by this! There is absolutely no requirement for every UOW to be connected to some other UOW. Remember that we are only looking at this stage for those dynamic 'generates' relationships. Not everything generates something else or is generated by something else!

Pupil: Well, I'm tempted to suggest that there must be some relationship between the *Student* and a *Student assessment*.

Tutor: It's very important at this stage to draw only the dynamic relationships. Do not be tempted to think of other – possibly interesting – relationships. A Student assessment is generated from the lifetime of a run of a module. Yes, it concerns the student but it isn't the lifetime of the student that generates the assessment. This is a subtle but important point.

Pupil: How come students appear out of thin air?

Tutor: As far as this organisation – the Faculty Administration – is concerned, they do! We've turned the searchlight away from student recruitment towards the teaching side of the faculty.

PRODUCING THE FIRST-CUT PROCESS ARCHITECTURE

We started this chapter with the assertion that a true process architecture should be an invariant of the organisation, determined only by the business the organisation is in. So far, we have developed a characterisation of that business in terms of a set of essential units of work and their dynamic relationships, all drawn up on the UOW diagram. The next step – producing a 'first cut' of the process architecture – is entirely mechanical and uses the ideas we developed in the previous two chapters.

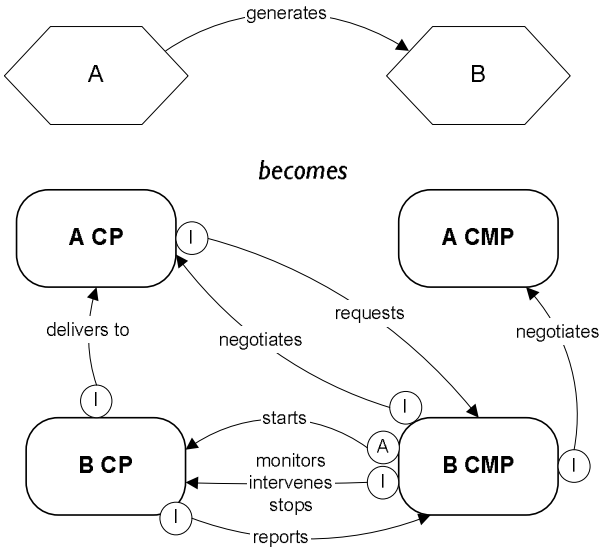
Firstly, we hypothesise that for each UOW on the UOW diagram there are three processes: its case process, its case management process, and its case strategy process. So, for the UOW *Customer call*, we know that Handle a customer call, Manage the flow of customer calls, and Maintain a strategic view of customer calls will all appear somewhere in our process architecture. By definition, if *Customer call* is a UOW, we have to look after it during its life-time: that is what the case process does – it looks after one instance. Since there will

potentially be many in progress at any one moment we need to handle that flow: that is what the case management process does. Since this thing is either of the essence of the business or important enough to be designed, somewhere we need to take a strategic view of it: that is what the case strategy process does.

Secondly, we hypothesise that each 'generates' relationship between two UOWs in the UOW diagram can be translated into relationships between the corresponding processes. Here we use the results of Chapter 5. We examine each 'generates' relationship and decide if it is a task force or a service function relationship.

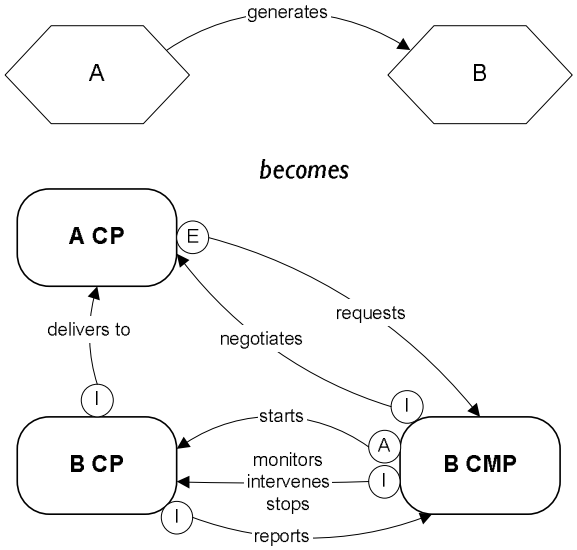
If *A* generates *B* and the relationship is a service relationship, then the translation rule in Figure 6-4 is used.

Figure 6-4 – Translating a service relationship between UOWs into processes



If the relationship between *A* and *B* is a task force relationship, then the translation rule in Figure 6-5 is used.

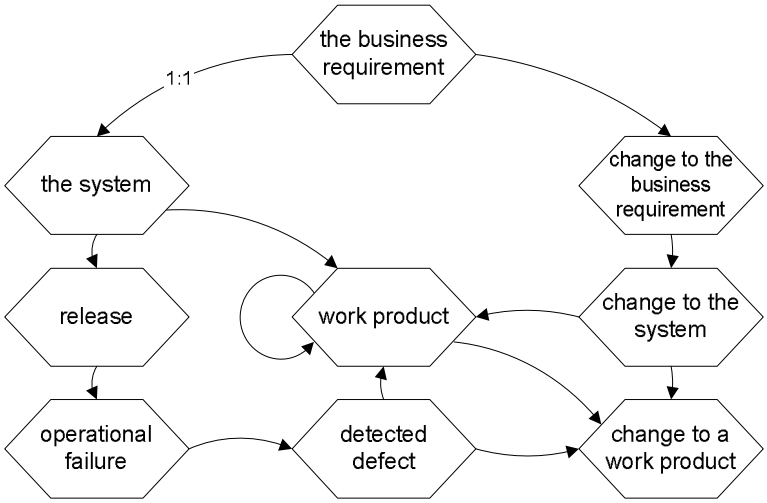
Figure 6-5 – Translating a task force relationship between UOWs into processes



Note that in both cases we have drawn the full set of relationships between case process and case management process: starting, monitoring, intervening, stopping, reporting, and negotiating.

Figure 6-6 shows the UOW diagram we might draw for the area in a company to do with looking after the system that supports the business requirement. There is one business requirement, which has a lifetime that is looked after and during which changes arise. Matching it is one system, which also has its lifetime. During the lifetime of the system, new releases are made of it. Looking after it also requires new work products to be generated. And so on. Spend a moment understanding the dynamics of this organisation.

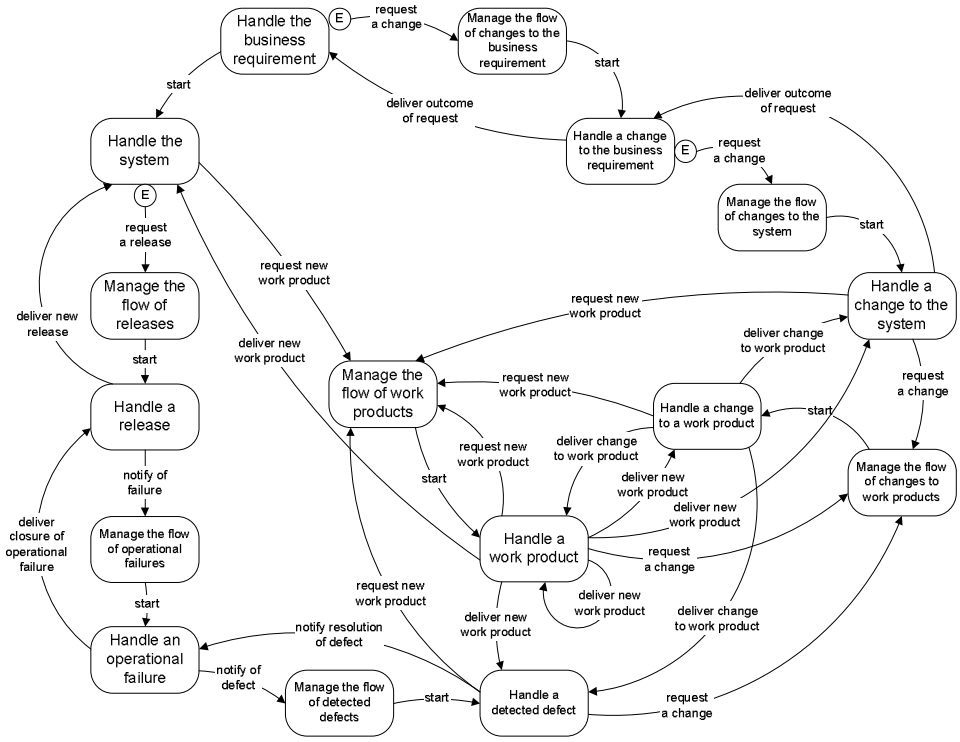
Figure 6-6 – A UOW diagram for the IS support to the business requirement



Knowing how to deal with the two relationships of service function and task force we can, quite mechanically, transform the UOW model of Figure 6-6 into the first-cut process architecture in Figure 6-7. We have only shown the 'encapsulates' relationships, as all the others should now be obvious from context.

(We shall generally omit case strategy processes from the process architecture unless they are of specific interest for our purpose.)

Figure 6-7 – The first-cut process architecture for the UOW model in Figure 6-6



Pupil: It's complicated – should I be worried?

Tutor: Ask anyone in a company looking after computer systems supporting business requirements and tell them it's simple! That said, this is the first-cut architecture. There are some reductions that we can make. But take careful note: they are reductions that mirror the real world. They are definitely *not* simplifications to make the diagram look simpler! That would do us no service at all.

Pupil: OK, I can accept that. But people will surely be put off by the fact that the picture is a mass of blobs and arrows.

Tutor: I'll put what you said another way: people will surely be put off by the fact that the business operates a complex network of interconnected processes. Whether or not they are put off, it is true! And we shall not get to answers if our pictures are lies or over-simplifications.

PRODUCING THE SECOND-CUT PROCESS ARCHITECTURE

That last step was purely mechanical. The resulting process architecture represents – in a sense – the most we can expect to find in the way of processes for the units of work we

identified. In practice, it often shows more than exist. Let's explore some of the ways that we can reduce the first-cut process architecture to produce the second-cut process architecture.

Folding a task force CMP into the requesting CP

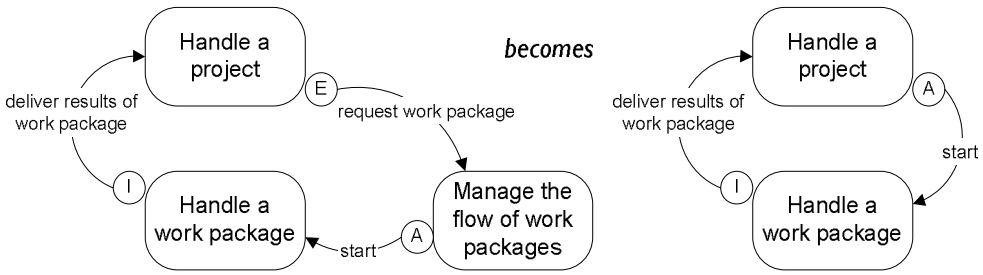
Where a task force relationship has been transformed, and the case management process that receives requests is shown as encapsulated in the requesting case process, we can decide to fold the CMP into the requesting CP, particularly if it is trivial or near trivial. Why does this make sense? We know that the things being managed by the CMP are only requested by that CP – otherwise it would be a service function. So, it's plain that we can consider that case management to be part of the requesting case process.

As an example, the fragment of first-cut process architecture on the left of Figure 6-8 is folded to give the fragment on the right. This arose from the UOW relationship

> A Project generates Work Packages

and a recognition that that relationship is a task force relationship: the project sets work packages going under its own initiative, rather than going to an 'outside' service that does work packages. Put another way, the **Handle a project** case process does its own case management for work packages.

Figure 6-8 – Folding a CMP into the requesting CP



It is vital to remember that when we fold the CMP into the requesting CP, we are *not* saying that the CMP doesn't exist, or that there is no case management to be done. We are saying that it sits within the requesting case process and is best modelled there. We are entirely at liberty to leave it as a separate process if we think that our purpose will be better served by doing that.

Dealing with 1:1 'generates' relationships

In some cases we will have on our UOW diagram one instance of an *A* generating precisely one instance of a *B*, what we shall a 1:1 relationship. For instance, we might have drafted the following

> A Customer Purchase generates one Invoice.
>
> A Clinical Trial generates one CRF Design.
>
> A Draft Document generates one Approved Document.
>
> A Job Applicant generates one Staff Member.

There are some subtleties here.

Suppose that in our organisation one *Customer purchase* generates precisely one *Invoice*. Assuming that invoicing is done, for all sorts of reasons, as a service within the organisation, we shall want to keep the processes arising from *Customer purchase* and *Invoice* separate. We shall need the separate Manage the flow of invoices to sort out priorities between invoices from the different sources, i.e. from case processes other than Handle a Customer Purchase. So the handling of invoices is a service function and we must leave the full set of case and case management processes on the second-cut architecture.

Suppose we are the Clinical Trials Division of a pharmaceutical drug development company. The 'Case Report Form' is a large document that records the life-history of a single patient taking part in a clinical trial. For each clinical trial, a CRF is specially designed to take into account the particular requirements of that trial. In *Riva* terms, a *Clinical trial* generates just one *CRF design*. Do we still need both UOWs? Yes, we probably do. The CRF design has its own life that runs alongside the trial's: it is drafted, reviewed, approved, copied, amended, and so on. However, since there is only one CRF design for one clinical trial, and its lifetime is intimately related to the trial and not passed off to someone else to look after, will we need case management for CRF designs? Yes, we probably do: there might only be one CRF design for each clinical trial, but there are many clinical trials and hence many CRF designs, and if we provide the development and maintenance of CRF designs as a service then we shall need both Manage the flow of CRF designs and Handle a CRF design in the process architecture.

A *Draft document* is a document in a particular state: draft. After it has been through certain processing it might change its state and become an *Approved document*. What happened there? Perhaps it would make more sense to say that a *Document* changes state, and hence to replace the two UOWs by one. The first half of Handle a Document would then be about the lifetime of a document in the draft state and the second half would be about a document in the approved state. We would not distinguish between case management of draft documents and case management of approved documents, contenting ourselves with Manage the flow of Documents.

Suppose we have a UOW called *Job applicant* and another called *Staff member*. In some cases, job applicants become staff members. In some cases they don't. Job applicants sometimes 'turn into' staff members. Strictly speaking, *Job applicant* is in a '1: 0 or 1' relationship. with *Staff member*. Following the *Document* analogy, we would try to find a single UOW of which *Job applicant* and *Staff member* were just different states. Hard. Moreover the lifetime of a job applicant is quite different from the lifetime of a staff member – quite different things happen to you – and it feels better to keep them separate and hence with their own case processes. Also the case management of job applicants is a wholly different thing from the case management of staff members. So we shall find the full set of processes in the process architecture – Handle a Job applicant, Manage the flow of Job applicants, Handle a Staff member, and Manage the flow of Staff members – and at some point Handle a Job will request a new staff member from Manage the flow of Staff Members. That is the moment of recruitment.

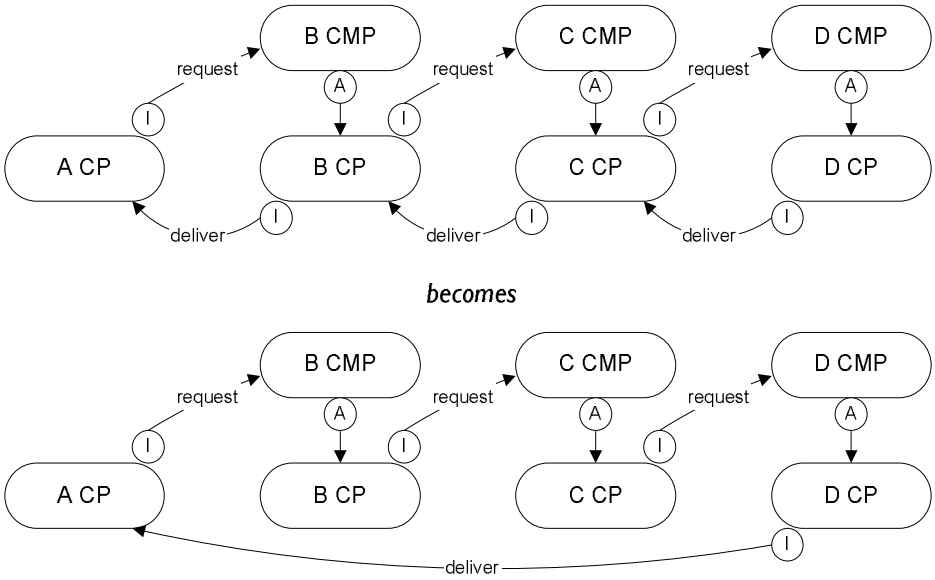
Dealing with delivery interactions and delivery chains

The standard transformation always produces a 'delivers' interaction from the requested case to the requesting case. When a project generates a work package, the work package is

assumed to deliver something back to the project. And in this situation that is probably what happens in reality. But in other situations the 'generates' relationship is more 'fire and forget'. So once we have the first-cut architecture in front of us, another validation we can carry out is to examine each 'delivers' interaction and ask 'Does this happen in reality? Does anything really get delivered?' If the answer is 'No', then we can delete that interaction.

Let's look at another situation. Suppose *A* generates *B* generates *C* generates *D* on the UOW diagram. This will yield the first-cut process architecture shown in the top half of Figure 6-9. Note how the chain of delivery is from *D CP* to *C CP* to *B CP* to *A CP*. It is always worth thinking this through and comparing it with reality. We often find that in practice this delivery chain is short-circuited and that the actual relationships are as shown in the bottom half of the figure.

Figure 6-9 – Some delivery chains can be short-circuited



We shall see an example of this in a case study later.

Dealing with collections

We saw earlier how our UOW analysis might turn up a UOW that is actually a collection of another UOW. For instance, a *Programme* is a collection of *Projects*; a *Project* is a collection of *Work Packages*; a *Transmission System* is a collection of *Assets*; a *Product Portfolio* is a collection of *Products*; and so on. In other words, during the analysis we have decided that not only does an individual project have a lifetime that needs handling, but a collection of projects – a programme – also has its own lifetime that needs handling. We will probably also have identified that

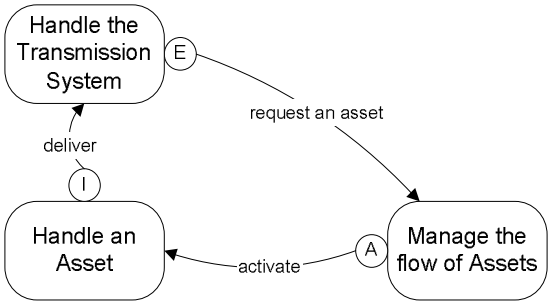
A Transmission System generates Assets.

A Project generates Work Packages.

A Product Portfolio generates Products.

with our special use of the word 'generates'. Let's take the first of these. It will produce the snippet of process architecture shown in Figure 6-10. Let's think what the process Manage the flow of Assets is likely to be about. The more we think about it the more we suspect that managing the flow of assets is actually part of what handling a transmission system is all about. So the case management process for the atomic object is the same as – or at least is contained within – the case process for the collection, and once again we are likely to fold the case management process into the collection's case process.

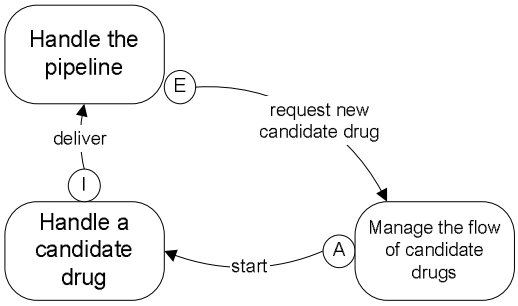
Figure 6-10 – UOWs, collections, and their processes



As another example, let's look again at the way in which a pharmaceutical development company runs the business of getting new chemical compounds into the drug market. We can identify an obvious unit of work: the *Candidate drug*. Each candidate drug will have its own life history: it starts as one of thousands of compounds that have been successfully screened, and it proceeds through a variety of ever larger trials; at the same time, there is a parallel development of the chemical process by which it will finally be manufactured in bulk. If the candidate drug proves safe and efficacious it will reach the market. This 'Molecule to market' process will involve many scientific groups that deal with the candidate drug: pharmaceutical sciences, clinical trials, manufacturing process scale-up, analytical chemistry, health and safety, quality assurance, and the regulatory group.

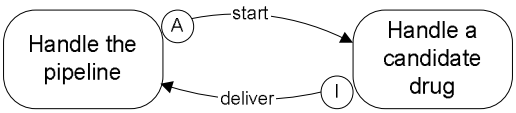
At any one moment the company will have many drug candidates at different stages. Such an organisation is in fact a 'case pipeline', designed to get the successful candidates through to the market place in the shortest possible time and to weed out the unsuccessful candidates as soon as possible. (So few candidates typically make it to market that perhaps the process should be called 'Molecule to reject bin'.) The pipeline can be considered a unit of work in its own right: it has a (never-ending) lifetime and must be looked after. Clearly the pipeline is the set of candidate drugs. At any one moment there will be one pipeline containing many candidate drugs. Indeed, the pipeline generates candidate drugs, so we shall expect to find the three processes shown in Figure 6-11.

Figure 6-11 – The first-cut processes for a pipeline and its candidate drugs



Now, requests for candidate drugs do not come from anywhere else other than the pipeline, and the pipeline *is* the set of candidate drugs. So a task force relationship is appropriate between Handle the pipeline and Manage the flow of candidate drugs (notice the E-relationship). This suggests that we can fold Manage the flow of candidate drugs into Handle the pipeline. This will make immediate sense: if we watch Handle the pipeline at work we will see that it is amongst other things about managing the flow of candidate drugs. The result will be Figure 6-12, but note that we are never obliged to fold the case management process away like this: it is a modelling decision to be made for each situation.

Figure 6-12 – The reduced processes for a pipeline and its candidate drugs



Dealing with empty case management processes

It is not uncommon for a case management process to be … empty: some things simply don't need or don't get case management. As a trivial example, we know in Figure 6-10 that there is only one Transmission System – there will be no process called Manage the flow of Transmission Systems and there will only be one instance of Handle the Transmission System.

Perhaps Jill deals with invoices. If you want an invoice, you walk up to her desk and leave an invoice request on it. She picks up invoice requests from her desk at random and works on them, sometimes working on several at once. The process Manage the flow of invoice requests simply does not exist. For our second-cut process architecture we might therefore remove it: any requesting case processes will simply activate Handle an invoice request itself, rather than asking a Manage the flow of invoice requests to do it. Dropping your request on her desk is all that needs to be done.

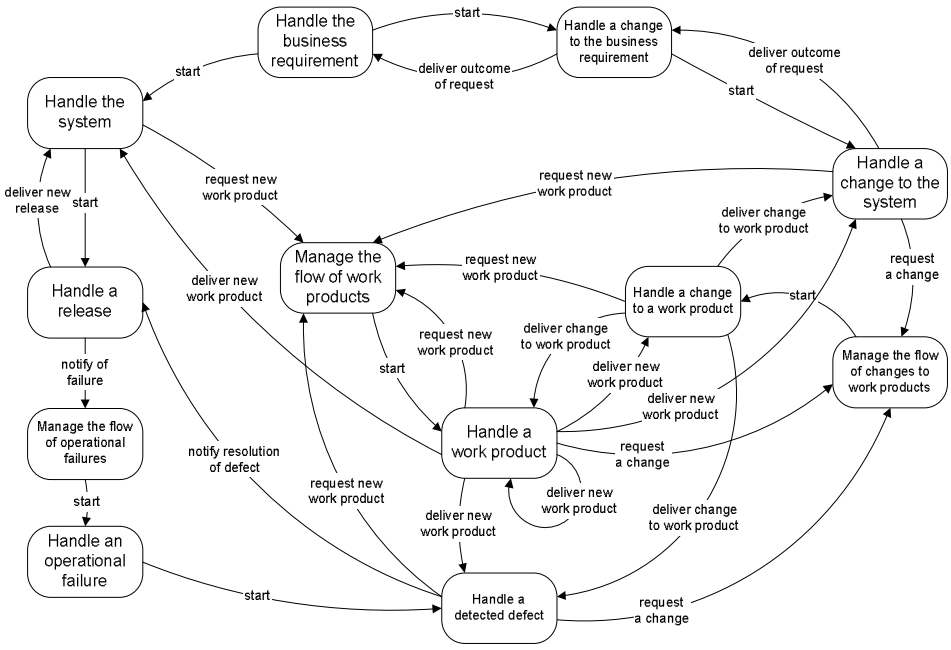
But then Jill decides to organise herself better: she now has an in-tray where you put your invoice request. It's annoying but she simply takes the top request off the pile, deals with it, takes the next one off the top, and so on. Last on, first off. Now there is a Manage the flow of

invoice requests process, and it has a serious effect on the variability of the time it takes for an invoice request to be dealt with.

From first-cut to second-cut architecture

We are now in a position to reduce the first-cut architecture in Figure 6-7 to the more realistic second-cut architecture in Figure 6-13. You can see that much of the reduction (a word I prefer to 'simplification') has occurred around Handle the system – not surprisingly as there is only one and hence it does a great deal of case management itself. Such reductions can't be made for the 'smaller' UOWs – e.g. *Work product* and *Change to work product* – where requests for them come from all sides and the flow is managed as a service. We have also short-circuited a couple of delivery chains to reflect reality.

Figure 6-13 – Second-cut architecture from the first-cut architecture in Figure 6-7



Designed business entities and UOWs

When we analysed our candidate EBEs looking for UOWs, we bracketed out designed entities on our list of candidate EBEs and so they were not carried forward onto the UOW diagram. Our reason was that, if we want a process architecture that is solely based on the business the organisation is in and is independent of how it chooses to do its business, then we must drop them from the list. The result is that our process architecture really does concentrate solely on those processes that must exist because we are in the business we are in. There are no processes that are there simply because of some decision about *how* we should do our business.

This gives our process architecture a 'purity' that can be very useful in many situations. If we want to re-engineer then it is important not to be blinded by current mechanisms.

On the other hand, if we are building an 'as-is' view of the organisation, then we shall want to see processes in the architecture that *are* there because that is the way we choose to do our business. So we can take the brackets off the designed UOWs concerned and let them generate processes in the architecture.

KEY POINTS

To build a process architecture:

1. Identify the essential business entities (EBEs).
2. Use the filters to extract the units of work (UOWs).
3. Map the UOW relationships and add their type and cardinality.
4. Transform the UOW diagram into the first-cut process architecture using the two standard transformations.
5. Consider folding some processes, especially where task force relationships are involved.
6. Put the resulting architecture against the world to validate the relationships.
7. Restore designed units of work, if appropriate.
8. Make any further reductions and finalise the second-cut process architecture.

Other process interactions

When constructing the process architecture we deduced the principal interactions between processes: the request from a case process to a case management process, the delivery from a case process to a requesting process, etc. There will of course be others: when we get into detail about how processes operate – how we have chosen to do our business – we shall see that there are other interactions between processes. But since they are about how we do our business, we would rather not have them on our 'pure' process architecture, an architecture that we want to be re-engineering-proof. They will arise naturally as we design/model the individual processes.

THE PROCESS ARCHITECTURE AS SEARCHLIGHT

Pupil: You were very rude earlier about so-called process architectures that were random chunkings of organisational activity, either strung together in some sort of sequence like a kebab, or else hanging Christmas tree fashion. How is a *Riva* process architecture different?

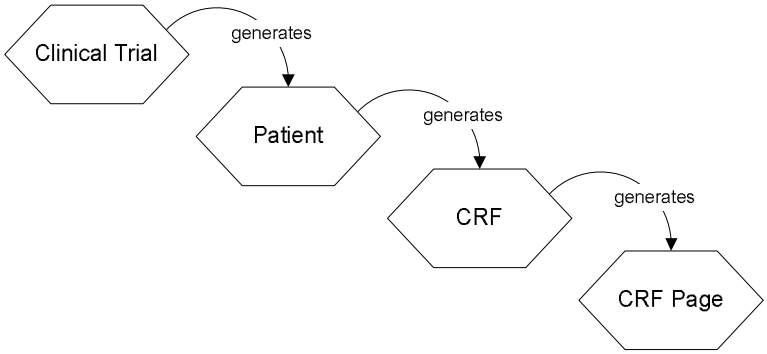
Tutor: Well, we've seen how units of work come in a variety of sizes: a Compound represents a much larger unit of work than an Assay. We can also see a partial ordering ('generates') amongst units of work. It is likely, though not certain, that as we follow the 'generates' relationships we'll find our way to

'smaller' and 'smaller' units of work. Look at the (real-life) example I've drawn in Figure 6-14 and tell me what you see.

Pupil: Well I see it shows a sequence of UOWs that get smaller in 'size': I know a clinical trial is a massive thing, a CRF page is … just a page in a document called a CRF.

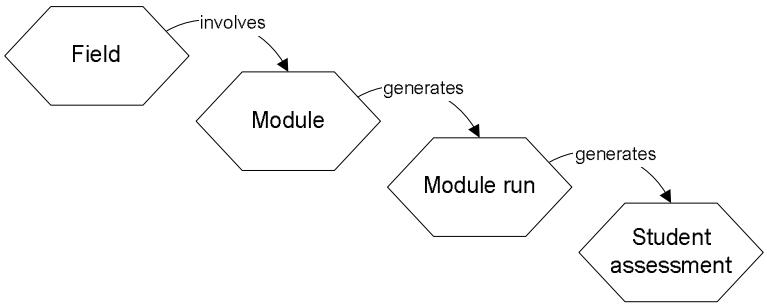
Tutor: Right, and all of these entities have a similarly long duration: a patient is a patient as long as the trial persists, as do the records about them. Indeed we might reckon that the records last longer than the patient in the trial in that even though the patient has left the trial and ceased to be a patient, the records about them are kept and looked after a great deal longer.

Figure 6-14 – Big fleas have smaller fleas …



I've drawn another sequence in Figure 6-15 to do with the administration of a university faculty. This time the UOWs are getting smaller in duration.

Figure 6-15 – Big fleas have shorter fleas …



Can you see how this says that, to add more 'detail' to our process architecture, we must add more UOWs to our UOW diagram, with the new UOWs being generated by (probably) 'larger' UOWs already on the diagram. Note

that we don't add more detail by decomposing into lower levels of a hierarchy. We add more detail by *adding more nodes to our network*. And indeed this reflects the real world. The question is only whether we choose to include them in the model or exclude them from it. By choosing to include them we are turning our searchlight on that part of the organisation's activity.

Pupil: So, it's not that these 'smaller' units of work are 'inside' the larger ones – more that we don't see them if we choose not to?

Tutor: Exactly. Here's another way of looking at it. When we stare up at the night sky, we can only see the stars we can see: the ones that are bright enough. If we go somewhere where there is less light pollution, we see more of the less bright stars. It's not that we are seeing *inside* the bright stars and seeing stars there. Finally, we need a telescope to see the very faint stars between all the others – note how I said 'between'. That's how it is with units of work and the *Riva* process architecture.

Turning my night-sky metaphor on its head, I can also think of the process architecture as a searchlight which I train on the organisation. I can widen or narrow its spread and look at some parts and not others. Equally I can increase or lower its intensity, allowing me to see more or fewer of the smaller (fainter) units of work.

Pupil: So when I choose the units of work I choose the ones I want to choose for my purpose.

Tutor: Right. But remember that the stars are all there – you don't invent them. You look at this part of the sky or that part. And you simply see more or fewer of them, as you choose. We're modelling reality. We're not imposing nice structures on reality.

Pupil: You made a big deal earlier about why a sound process architecture was essential. How have we benefited from doing it the *Riva* way?

Tutor: I'll give you three ways. Firstly, by deriving the architecture from the entities that are at the heart of the organisation's existence we've ensured that the architecture is deduced solely from *the business the organisation is in*. We haven't considered at all how the organisation chooses to do business: its organisational structure, its geography, its culture, whether it is a command-and-control or empowered or consensus organisation, whether it operates strict hierarchical procedures or allows cross-functional communications, whether information is treated as shared or on a need-to-know basis, and so on.

Interestingly, we haven't been at all concerned with the *goals* of the organisation either. Goals are achieved by the way that processes are done. The *Riva* process architecture tells us what processes we must have to do our business, and we use the goals when we design those processes to ensure that they do what we want.

Pupil: In other words, the process architecture says 'If you are in this business you will have these processes in these dynamic relationships,' and it's only when we start to look inside the individual processes and at how their relationships are implemented that we introduce culture and organisational style.

Tutor: Yes. We do that by our choice of roles and the interactions we require between them, in particular in the use of approval, delegation, reporting, agreement, authorisation, negotiation, questioning and informing, and the mechanisms for these interactions.

Secondly, by mapping the relationships between those entities into relationships between their respective case and case management processes we ensure that the dynamic content of the architecture matches what is happening in the real world. Our architecture doesn't impose structure on our processes in the form of some arbitrary decomposition with no analogue in the real world.

Pupil: And the third benefit?

Tutor: Well, by recognising those two well-defined ways in which processes can be related – activation and interaction – we again capture reality, and moreover we do it in a way that can be directly modelled when we look at the detail within processes in a Role Activity Diagram.

Pupil: The message I'm getting is that a *Riva* process architecture would 'survive' re-engineering, in other words it's in some sense an *invariant* of the organisation; we've divided the organisational activity along the natural cleavage lines and as a result our processes will have greater cohesion and less coupling; and we've an approach that allows us to turn the spotlight wherever is appropriate to our concern.

Tutor: Precisely. I can put it more mundanely: we can really rely on a *Riva* process architecture. As long as the organisation stays in the same business, it will have those processes in those relationships. And if it changes its business, we can very easily determine the changes in the architecture: 'changing your business' means adding or subtracting essential business entities, which – we know – will tell us immediately which new processes appear and which existing processes disappear.

CASE STUDY

Let's start with a relatively straightforward example of a process architecture. Suppose we are looking at the job management part of a water utility, the part that undertakes various works on its assets: drains, sewers, water supplies, reservoirs etc.

A workshop brainstormed the following as potential EBEs, and decided that the ones in brackets were either not units of work or were not units of work that came within the scope of the business process management project:

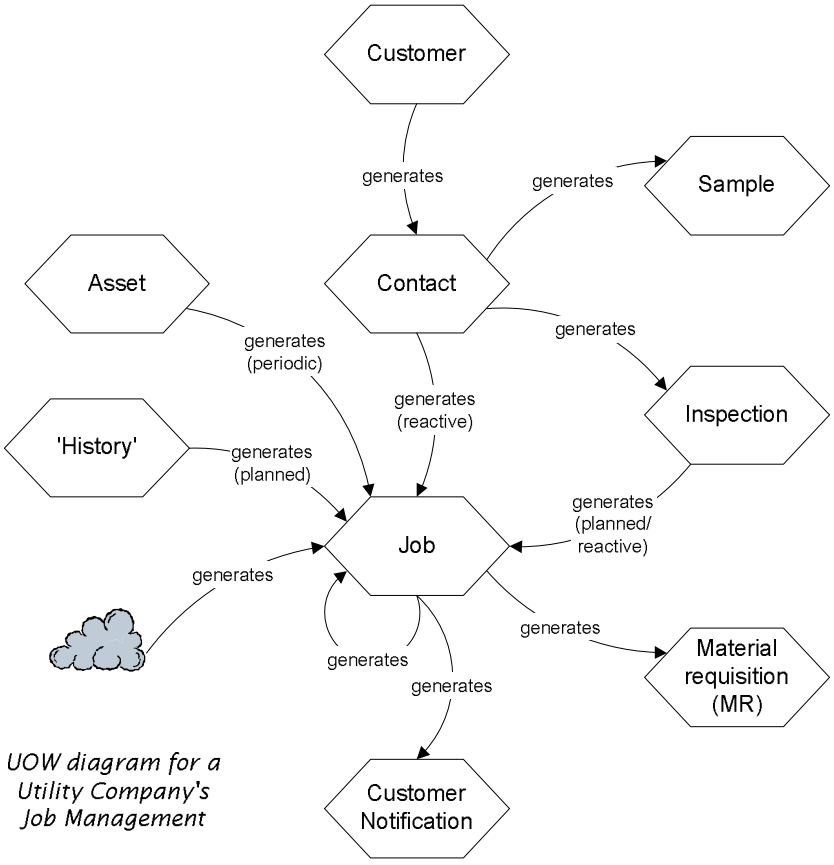
Customer

Contact (by a customer)

Job
Sample (of water)
Inspection
Customer notification
Material requisition (MR)
Asset
History of an asset.
(Statutory notification)
(Event/Incident)
(Appointment)
(Meter reading).

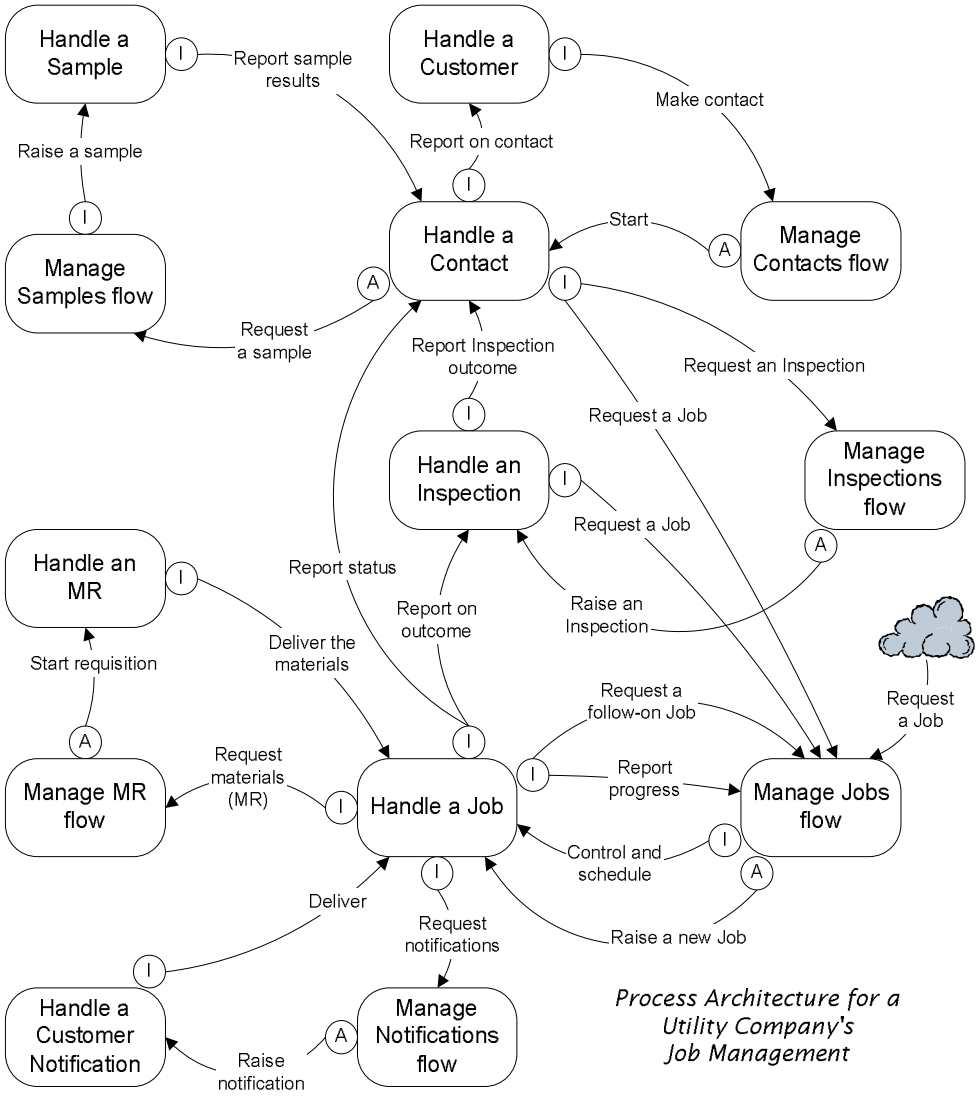
This led to the UOW model in Figure 6-16. Not surprisingly, given the scope of the study, the searchlight is on *Job* and its UOW 'neighbours'.

Figure 6-16 – UOW model for a utility company's job management



This in turn led to the process architecture in Figure 6-17.

Figure 6-17 – First-cut process architecture for a utility company's job management



Discussions at the workshop led to the following decisions:

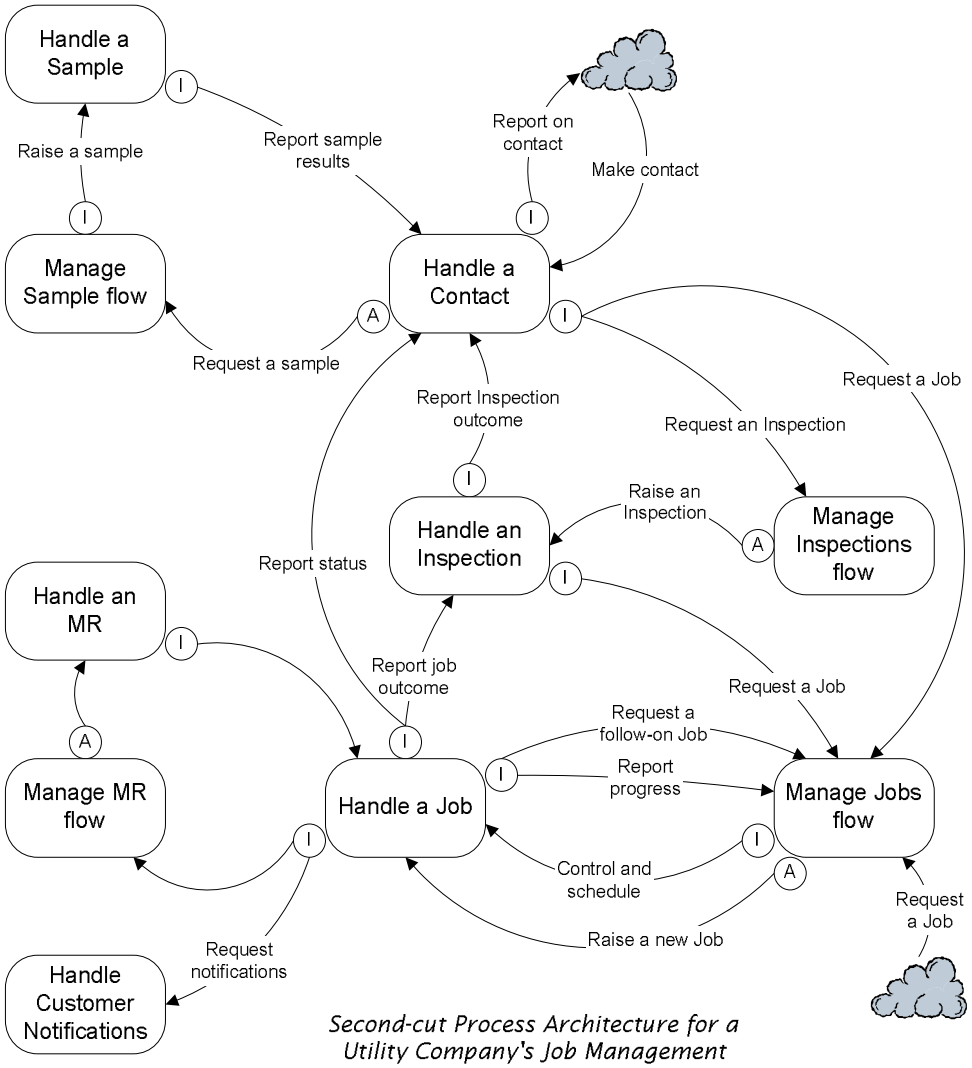
- The UOW *Asset* was dropped because the processes concerned with assets were considered outside the scope of the model: they were the business of the engineers who specified, procured and maintained them.
- Various sources of requests for *Jobs* (including the periodic maintenance of assets and planned requests arising from *History*, e.g. a trend of leaks in a particular area)

were bundled and shown as a cloud to represent an external source of no interest other than as a source of requests.

- ☞ All of the UOWs were supported by service functions so case management processes appeared for each.
- ☞ The relationship between Handle a Job and Manage Customer notifications is a little different from the usual. Here, in one request, Handle a Job requests Manage Customer notifications to organise the notifications for all customers. Manage Customer notifications is given a single geographical location, determines all the customer to whom notifications should be sent, and then sends a notification to each. It is probably not worth separating out Manage Customer notifications and Handle a Customer notification, the latter being a very 'small' process, so they have been replaced by a single process Handle Customer notifications which takes a geographical location and sends out all notifications for that location. This is shown in the second-cut process architecture below.
- ☞ Note also how Handle a Job is not interested in knowing the outcome of the notification of customers, so there is no closure of that request.
- ☞ The lifetime of customers was deemed out of scope so Manage Customers flow and Handle a Customer were dropped.
- ☞ Some reduction is possible, knowing more about the real situation. In particular, all contacts are made by telephone and it is the telephone system itself that effectively does such contact flow management as is necessary. For this reason the Manage Contacts Flow (case management) process can be dropped.

The result is shown in the second-cut process architecture in Figure 6-18.

Figure 6-18 – Second-cut process architecture for the utility company's job management



KEY POINTS

The process architecture is a searchlight, focussing our attention on the area of the organisational activity that we are interested in.

By including 'fainter' units of work (typically designed) we increase the intensity of the searchlight.

There is no sense in which we are successively decomposing processes.

We are adding more nodes to the network of processes.

Later, next to the water-cooler

Pupil: I know that people get very worried about analysis paralysis. Isn't there a danger that people will perceive all this process architecture stuff as simply getting in the way of getting on and looking at the processes themselves?

Tutor: Yes, there is. People can be very keen to get on with what they perceive to be 'real work'. I hope that I've made the case for preparing a process architecture whatever situation we are in. What I now have to do is satisfy you – and those folks with busy lives and hard targets – that preparing a process architecture can be a very quick activity.

Remember firstly that we do it principally to make sure that we get the right chunking of all the organisational activity, and that we identify at least the dynamic relationships between the chunks – the processes.

Pupil: OK.

Tutor: Then remember that the work centred around brainstorming the EBEs, filtering them into UOWs , and finally doing any appropriate reduction of the first-cut architecture to the second-cut architecture. To give you a flavour of how quickly that can be done, let me give you an example.

I was at a sales meeting with a client, trying to sell them *Riva*-based consultancy. One of the client team had said she only had a short amount of time as she had a video-conference with colleagues across the Atlantic at which they would discuss a new framework for their Quality Management System. She needed to get away from the meeting early in order to draft a framework for discussion. 'How long have you got before your video-conference?' I asked. 'Two hours,' came the reply. 'That's plenty for what you want: let's spend the first hour brainstorming EBEs and deriving UOWs,' I said, 'and the second hour deducing the process architecture, and reducing it as much as we can in the time.' My challenge of free consultancy was accepted, and we did it. She went off with a second-cut process architecture to get her discussions started. I don't want to suggest that two hours is all you'll ever need, but we are talking about days and not months.

CASE STUDY

Finally, let's examine a simple process architecture and look in particular at how the dynamic relationships would appear in RADs for them.

Suppose a software product company has a range of products. During the lifetime of a product, changes to it are proposed. Occasionally the outstanding changes proposed for a product are reviewed; some are deleted and some are incorporated in a new release of the product. A brainstorming of essential business entities might come up with the following:

Product

Change proposal

Release

Sale

Customer.

Which of these are units of work? In other words, which of them have a lifetime which must be serviced by our 'organisation'? For this exercise, we might decide that we are not concerned with selling and marketing, only the generation of product for sale. Our list of UOWs is therefore

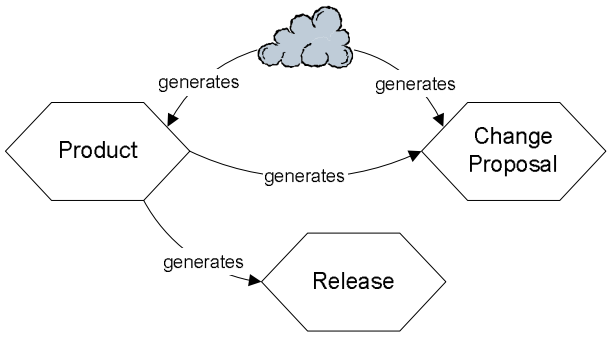
Product

Change proposal

Release.

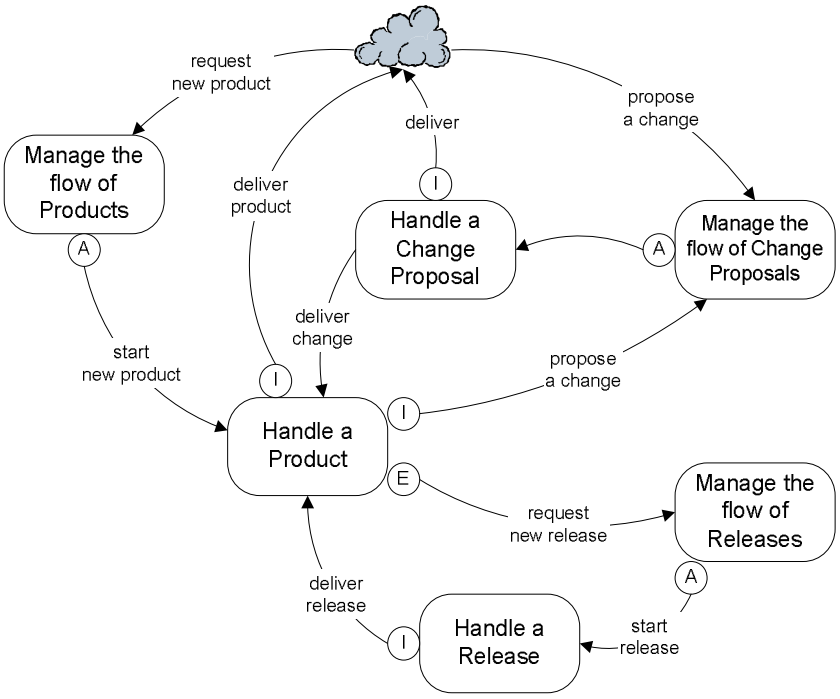
As far as the development group is concerned, ideas for new products come from outside, perhaps a marketing function within the company. During the lifetime of a product, releases are produced. Change proposals relating to products come from outside the group (customers in particular, we might suspect) and are raised during the management of the product itself. Our unit of work diagram would look like Figure 6-19.

Figure 6-19 – UOW Diagram for a development group in a software product company



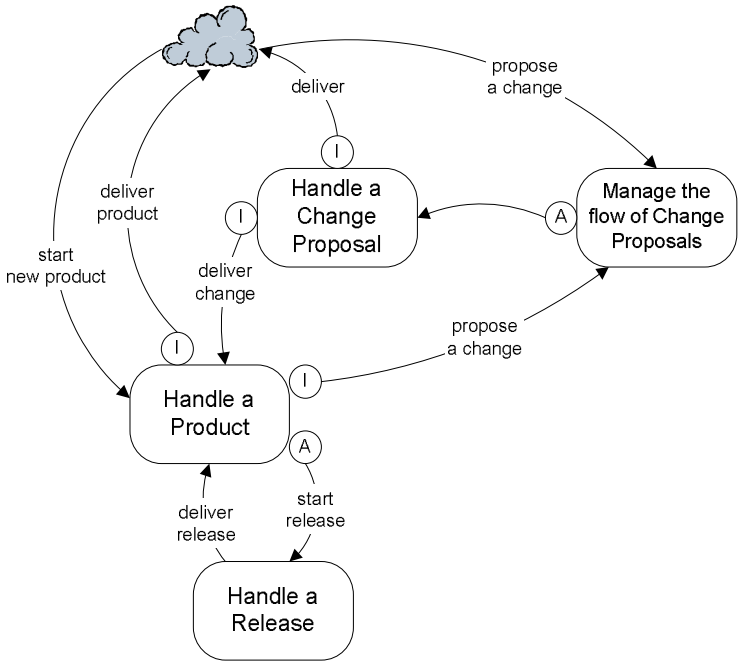
From this we can deduce our first-cut process architecture, as in Figure 6-20.

Figure 6-20 – First-cut process architecture from Figure 6-19



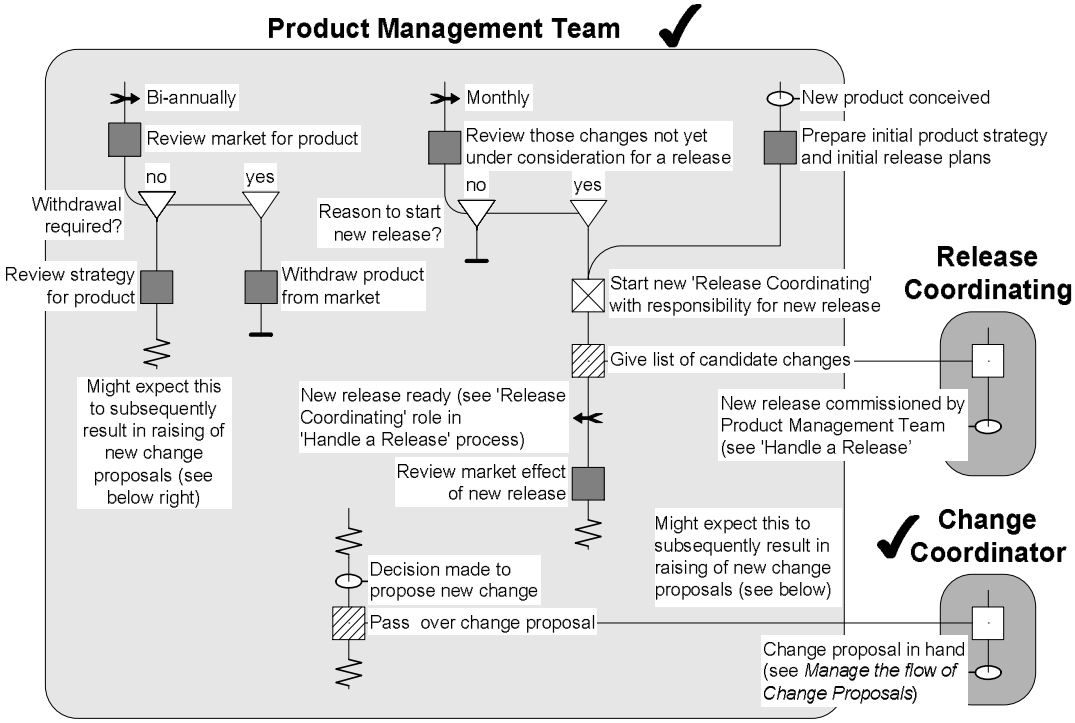
Each product manages its own releases and since releases are done one at a time for a given product, the case management process **Manage the flow of releases** doesn't exist so we can drop it. We can also assume that the business of deciding on new products – **Manage the flow of Products** – is outside the area lit by our searchlight. So we get the second-cut process architecture in Figure 6-21.

Figure 6-21 – Second-cut process architecture from Figure 6-20



Now we can start to get inside the individual processes. Figures 6-22 through 6-25 are incomplete RADs for the four processes: we have concentrated only on the activity around the relationships between the processes – we aren't interested here in the minutiae of change management. Let's listen in on our Tutor and Pupil discussing the models.

Figure 6-22 – Part of *Handle a Product*



Tutor: Why don't you walk us through the case study.

Pupil: OK. Looking at **Handle a Product** (Figure 6-22), first of all I would guess that somewhere else there's a **Manage the flow of Products** case management process that will create the instance of Product Management Team that the RAD shows.

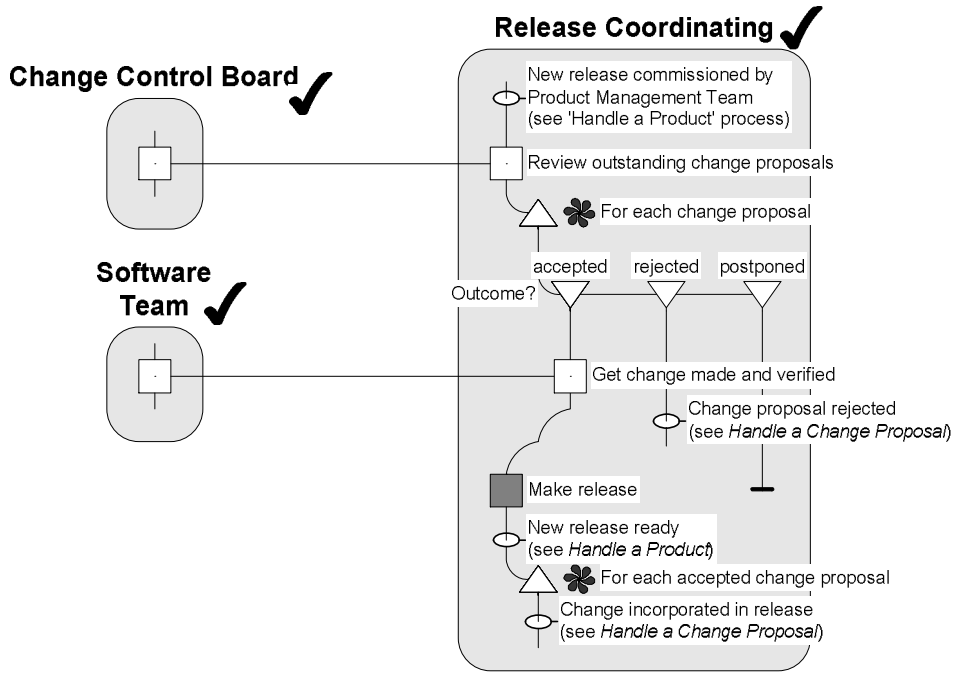
I can see that the Product Management Team has three main threads of activity. One (top right) is the main thread that gets under way immediately: the Team does some planning and then arranges for the first release; monthly it reviews things and decides whether a new release is called for, given the changes waiting their turn; and bi-annually it decides whether to withdraw the product from the market or perhaps make some changes – probably quite radical ones, if any, I would guess.

If the Team decides … we don't have any detail here … that it wants to propose a change to the product, it uses the appropriate route like everyone else and interacts with the *Change Coordinator* … that's the 'propose a change' interaction with the case management process *Manage the flow of Change Proposals* of course.

Going back to the point where the Team generates a new release … we know that case management for releases is null, so the Product Management Team simply activates the case process itself by instantiating its lead role – Release Coordinating – it creates the responsibility for the release … and lets it get on with it, waiting until the release has been done before checking what effect it has on the market. Interestingly, that thread – waiting to see the effect on the market – could still be in progress when the next monthly review is started for the next release?

Tutor: *True. Exactly the sort of concurrency we find: things rarely happen one at a time. It's entirely possible for several releases to be in hand at one time. Let's move on to the case process for Release.*

Figure 6-23 – Part of *Handle a Release*



Pupil: *In* **Handle a Release** *(Figure 6-23), I can see the instantiated Release Coordinating role preparing the release by filtering the outstanding change proposals, getting the appropriate software changes made for all of the change proposals that are accepted for the release, postponing some proposals for a future release presumably, and rejecting others there and then. Presumably that rejection is picked up in* **Handle a Change Proposal** *… I can check that process interaction in a minute. Once the release has been made there's an interesting state – New release ready – which of course the Product Management Team*

are waiting to hear about back in **Handle a Product** in Figure 6-22. And there's an interesting little … trick? We need to signal that each individual change that made it into the release has indeed made it, presumably so that the Change Coordinator can take the appropriate action over in the appropriate instance of the case process **Handle a Change**.

Tutor: Well, you might call that a trick but it captures reality!

Pupil: I noticed that the process architecture shows the new release being 'delivered' to the product via an interaction between the **Handle a Release** and **Handle a Product**. We've actually modelled this in your minimalist way as a state–trigger pair: the state *New release ready* in **Handle a Release** is spotted as an external event in **Handle a Product**. I guess we could have chosen to be more explicit and modelled the interaction that – say – takes place between the Release Coordinator and the Product Management Team to notify delivery?

Tutor: Yes, we could – that's a modelling decision of course.

Pupil: Another thing I noticed was that the responsibility for a new release has been represented as an abstract role – *Release coordinating* – and that the Product Management Team instantiates it for each release. But the same isn't true for a change proposal: we're not instantiating a role – a responsibility – just for that change proposal. Instead, we're passing it to a fixed post, the Change Coordinator … which I feel makes sense, but I'm not convinced … wait … most of the handling of a change proposal actually goes on in the handling of a release. The case management of change proposals is little more than a matter of checking that the request is a valid one and then sticking it on the pile of change proposals waiting for a release … oh, and notifying people of the outcome. The Change Coordinator does the case management and deals with individual cases – which is, I guess, why it makes sense not to have an abstract role taking responsibility for an individual change proposal.

Tutor: Yes, that's fair – in designing these processes we've chosen to map the responsibility for all change proposals and for their case management onto a single post in the organisation. In a different situation we might have modelled it differently. With the model as it stands we haven't yet said how the responsibility for a release is going to be allocated. That's a design decision we still have to make.

Let's move on to the case process and case management process for Change Proposals.

Figure 6-24 – Part of *Manage the flow of Change Proposals*

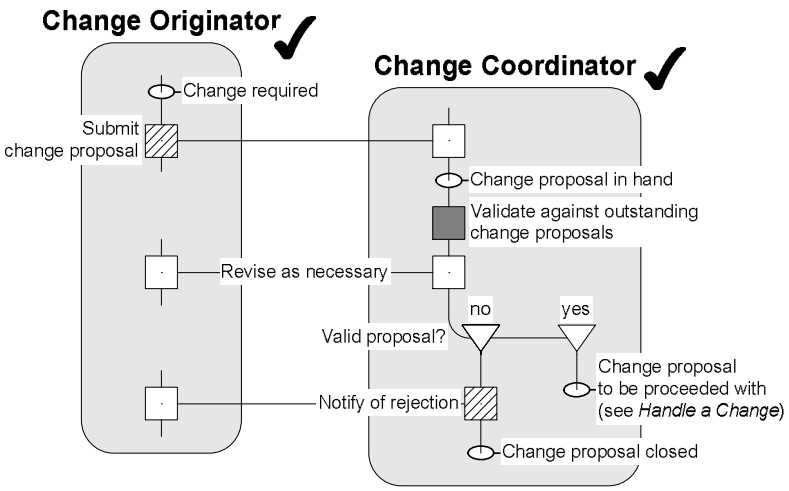
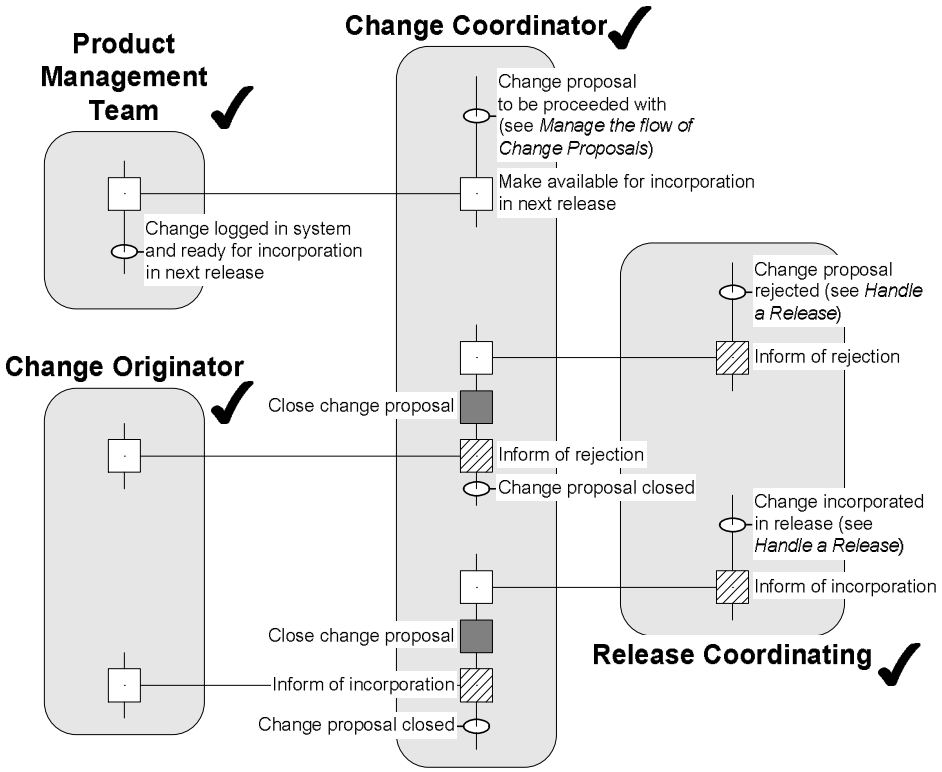


Figure 6-25 – Part of *Handle a Change Proposal*



Pupil: In both **Manage the flow of Change Proposals** (Figure 6-24) and **Handle a Change Proposal** (Figure 6-25), we've got a role called *Change Originator*. I guess this is a sort of anonymous role, in that a change can be originated by a Product Management Team, for instance, or by any agent in The Outside World.

Tutor: That's right – remember the cloud in the second-cut process architecture: proposals for changes can come from there. So, let's start with the case process **Handle a Change Proposal** (Figure 6-25).

Pupil: The top thread is where the case management process has started a new case .. and all the Change Coordinator does is pass the proposal to the Product Management Team, who put it on the pile ready for consideration at the next monthly review. That's an information-passing interaction between **Handle a Change Proposal** and **Handle a Product.**

Then there are two process interactions with **Handle a Release** where we hear about the rejection (*Change proposal rejected*) or incorporation (*Change incorporated in release*) of a Change Proposal. In both situations, the case ... the Change Proposal ... is closed and the outcome is notified to the original Change Requester. I suppose both of these process interactions are the result of design decisions about how we want these processes to operate, so they weren't on the process architecture?

Tutor: That's right. We must always remember that the purpose of the process architecture is to give us our initial chunking of the organisational activity into processes and also to tell us the dynamic relationships that must be there. When we decide how we want the processes to operate we might well generate the need for additional process interactions, very often for information-passing.

Pupil: You've been careful to separate the case process and case management process for *Change Proposal* but since, as you observed, the former does run straight through into the latter is there really any sense in keeping the RADs separate?

Tutor: We could indeed combine those RADs, as a modelling convenience. I've separated them, as you say, just to emphasise that there is case management going on here, and it might result in a change proposal being rejected before it is put into the pot, so to speak. In a realistic situation we could imagine far more complex decision-making being required at this stage, before a change proposal ever finds its way into the list. But certainly we could combine processes onto one RAD if it helps understanding, without losing precision. You might like to try the exercise of doing precisely that and putting all four of the processes we have just looked at onto one RAD. Be careful with pre-existing instances!