4 Process relationships

WHAT HAPPENS IN AN ORGANISATION?

So far, we have seen how a *Riva* RAD can be used to capture the concurrent and collaborative activity that makes up a single business process in terms of roles and their actions and interactions. But if we walk into a company's building, we might guess that going on around us are many processes dealing with the many different aspects of the organisation's life. As we look around, we sense a number of things:

- Some of them are 'big' processes – the development of a new pharmaceutical drug involving thousands of people across the world – and some are 'small' – someone claiming their expenses.
- Some are long-running, and some are over quickly – think again of the two processes in the above example: ten years against ten hours, say.
- Some processes support others: during the process of running a student examination, the process of marking an individual script is carried out many times; during the development of a pharmaceutical drug, many clinical trials are carried out; during a clinical trial, many medical tests are carried out.
- Some processes interact with others: the process for designing a new product interacts with the process for making a test batch; the process for preparing the annual budget interacts with the process for planning marketing campaigns.

We sense that all the activity going on around us in the organisation is a *network* of interacting processes, a network that is changing by the moment. Some processes are occurring many times at the same instant: 124 different clinical trials are 'in progress'; 2,489 expense claims are 'going through the system'; 167 papers are 'in the process of' being marked and moderated.

Let's think about that last paragraph more closely. We have one process for running a clinical trial, but there are 124 *instances* of that process in action at this moment. This sounds familiar. Just as it was possible for an action or a role *type* to have many instances at a given moment, a *process type* can have many instances in progress at any moment. And those instances come and go in the same way: when the 33 Physics students have sat the examination paper, 33 new instances of the process for marking an exam paper must be created. When the result for an exam paper is finalised, so the corresponding instance disappears, its job done.

(Note that we must be careful now to say whether we are talking about a process type or a process instance. I shall distinguish between them if it is not obvious which is being referred to. So, remember that the word 'process' on its own really needs to be read as 'process type' or 'process instance' as the situation demands.)

It's hard to over-emphasise the importance of these points when we look at how we will think about and represent processes in some sort of model, whether we are defining, or analysing, or designing, or improving processes. Let's see why.

If we fail to note that there are many process instances running at one moment, we will ignore the question of what makes them start and when and how. We will ignore the problems that arise from managing all that concurrent activity. We will ignore the effect of all that concurrency on productivity, on resources, on scheduling, and more.

When we describe a process in a RAD, we draw a static structure of role types. But we saw how, when that RAD 'runs', instances of roles can be created dynamically, and, within the role instances, instances of activities and interactions are created dynamically. The RAD captures the dynamics of the process, and all its potential concurrency, by describing the types and how they get instantiated. We need a similar approach to describing an organisation in terms of its processes. We can draw a static network of process types. But when that network 'runs' there will be a dynamic network of interacting instances. In Chapter 6 we shall look at how we can determine that 'process architecture diagram' – or PAD – that captures the dynamics of the organisation, and all its potential concurrency.

The important conclusion is that if we want to model activity at the organisational level then we shall need an approach that captures the dynamic relationships, and that captures the network and the way that it operates. We now have some important questions to answer:

- How do we decide what process types an organisation has? Put crudely: how do we chunk all that organisational activity? Putting aside which processes are started when and how they interact, what is the list of processes used by the organisation?
- Given that we have a network of interacting processes, what sorts of interaction can processes have? What sorts of relationship can exist between two processes? And remember that we are interested in dynamic relationships, not static relationships.
- Knowing what processes the organisation has, and the sorts of relationship that can exist between processes in a network, how do we decide precisely what dynamic relationships this organisation has between its processes?

If we can answer these questions, we should be able to walk into the building and, after some analysis, draw a picture of the network of process types that the organisation must have: its *process architecture*.

We shall answer these questions in a different order, and start by looking at the two main types of relationship that can exist between two processes, and how those relationships get modelled in a RAD. They are:

- *interaction*: where the two processes operate independently but interact at various points;
- *activation*: where one process starts another which then operates independently.

I must make an important point here. In each case, what we want to do is recognise that we will be drawing a RAD for each process in the relationship. We want each of those RADs to be free-standing and 'readable' on its own. But processes cannot be cleaved apart so cleanly; if we cut an arm off a living body we chop through nerves and blood vessels, and to get a true

picture we need to show where they came from or were heading. So we want a way of producing free-standing RADs whilst still showing where they are related or connected.

Finally, before moving on, I want to stress again how nervous I am about hierarchies and decomposition when we are thinking about processes. In *Riva* you will never see a hierarchy, you will only see networks.

KEY POINTS

One process instance can create an instance of another by *activation*.

Two process instances can collaborate via *interaction*.

From moment to moment, there is a flux of interacting process instances at work in the organisation.

Organisational activity is the operation of an evolving network of concurrent, interacting process instances.

INTERACTION OF PROCESSES

When we look at an organisation we know we will see many processes operating – strictly, many process instances. We also know that these processes do not operate entirely separately from one another, in particular two processes will occasionally interact in some way. For instance, a company might have an annual budget-setting process in which it reviews the portfolio of projects and decides on the budget for each in the coming year. At the same time, each project will follow a project life-time process that might take several years (i.e. several rounds of budget setting). At points during its lifetime, the project (process) will clearly need to interact with the budget-setting process to find out what its budget is to be. We will probably model the two processes on separate RADs. But we know that the processes interact at various points, so we will want to show those interactions on their respective RADs whilst allowing each RAD to be free-standing.

What precisely do we mean by 'process *A* interacts with process *B*'? In *Riva* we know that everything happens within roles, so a process interaction will clearly show itself as a role interaction. Firstly, this tells us that there is at least one role that the two processes have in common, in other words at least one role has a responsibility in each process. Secondly, we can expect to find a state of that role which appears in both processes; such a state represents some sort of synchronisation between the two processes. It is as if the person carrying out the common role can say 'When I'm here in this process, I'm there in that process' – the common role has a common state too.

The modelling of process interaction

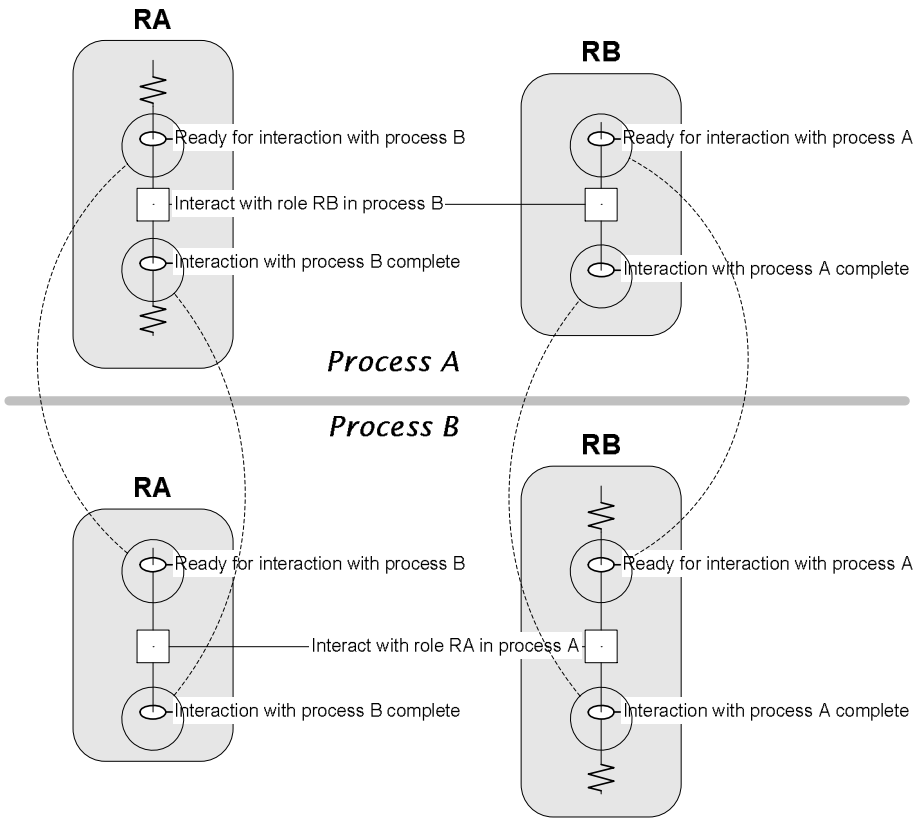
Let's look at the 'rule' for modelling a process interaction and follow it up with some examples.

When we model an interaction between two processes, *A* and *B*, it is best to start by modelling it as an appropriate role interaction in each process/RAD, and then either to pare things down or to beef things up, as the situation demands.

- ☞ Find the point of interaction in terms of the two roles in *A* and *B* that interact. Let's call them *RA* and *RB*.
- ☞ Draw that interaction in the RAD for *A* at the appropriate point, and in the RAD for *B* at the appropriate point. We can represent the interaction differently in the two models if that is appropriate. Note that *RA* and *RB* now both appear in both RADs.
- ☞ On the RAD for *A*, name the pre- and/or post-states of the part-interactions for *RA* and *RB*. Give the same names to the corresponding pre- and/or post-states of the part-interactions on the RAD for *B*.

Figure 4-1 shows the final situation in this general case. In both RADs we have shown the role interaction that constitutes the interaction between the processes. We have then labelled the pre- and post-states of each part-interaction, and used the same state labels in both RADs. You can think of the shared states as solder points that make electrical contact across – create the same potential in – the two RADs. In the RAD for process *A* we show a minimal amount of role *RB*. In the RAD for process *B* we show a minimal amount for role *RA*.

Figure 4-1 – A general model of process interaction



A sample process interaction

Consider the simple example in Figure 4-2. In the Carry out Annual Portfolio Review process, the *Product Divisional Manager* is told of a change in their budget as decided by the *Product Portfolio Management Committee*. As far as that process (model) is concerned we are not interested in how the *Product Divisional Manager* responds to that, simply that they are left in the state *Budget change received*: their response is the subject of the interacting process: Handle a Product. The *Product Divisional Manager* plays a part in that too, and it is in the model of that process that we map their response. Note how the two interacting roles appear in both models, as does their interaction.

The *Product Divisional Manager* and the *Product Portfolio Management Committee* are the two roles that are common to the two processes. Their respective post-states – *Budget change received* and *Budget change notified* – appear on both RADs.

Figure 4-2 – A simple process interaction

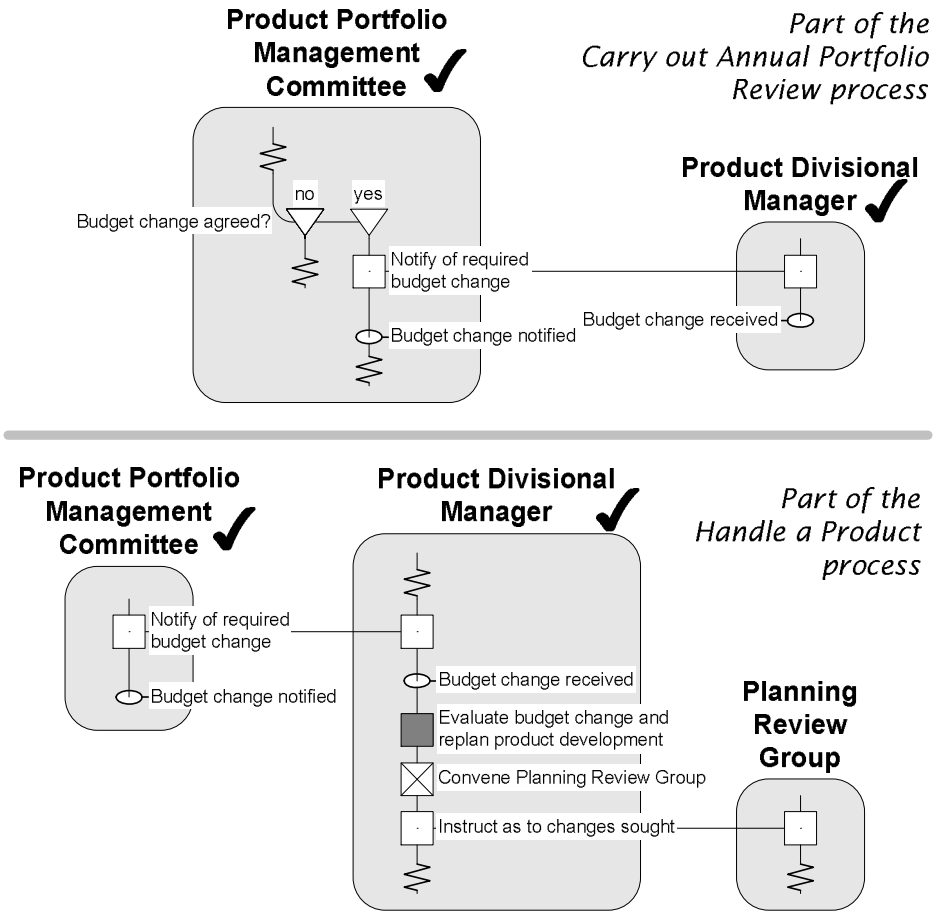
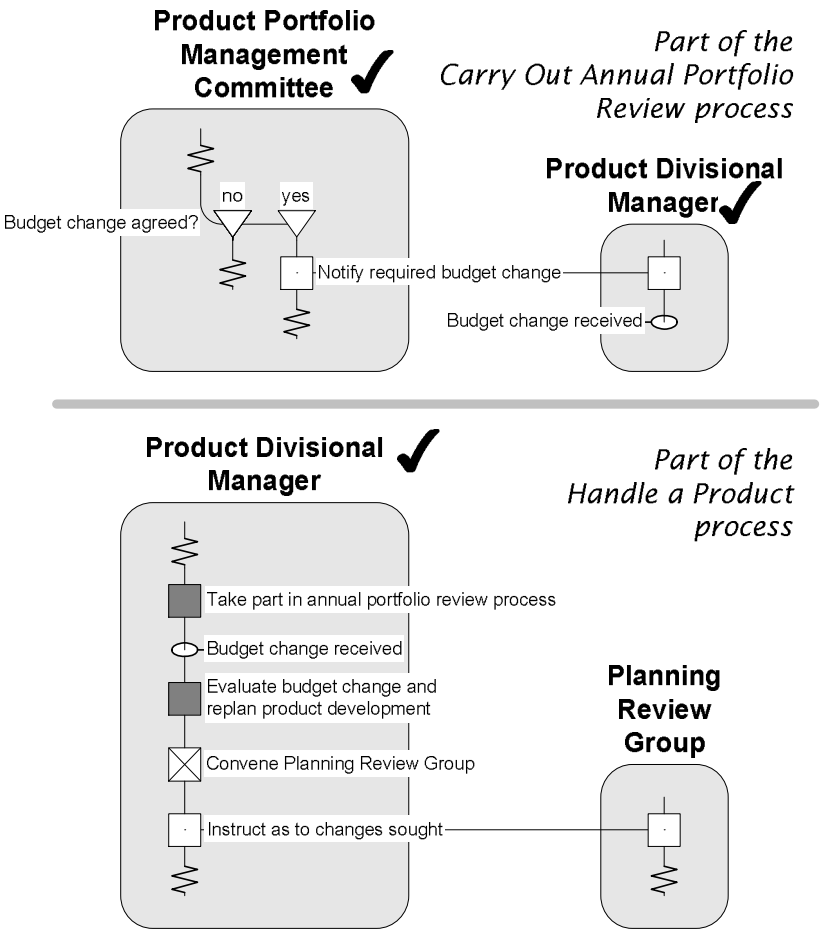


Figure 4-3 takes a slightly more minimal view of the process interaction, as a simple example of the modelling choices open to us. Other variations are possible of course.

In Carry out Annual Portfolio Review we have shown the role interaction between the *Product Portfolio Management Committee* and the *Product Divisional Manager* explicitly: *Notify required budget change*. The *Product Divisional Manager* is then in the state *Budget change received* after the interaction.

In Handle a Product we have used the action *Take part in annual portfolio review process* to stand for the interaction that the *Product Divisional Manager* has in Carry out Annual Portfolio Review. The *Product Divisional Manager* is then in the state *Budget change received* after that action.

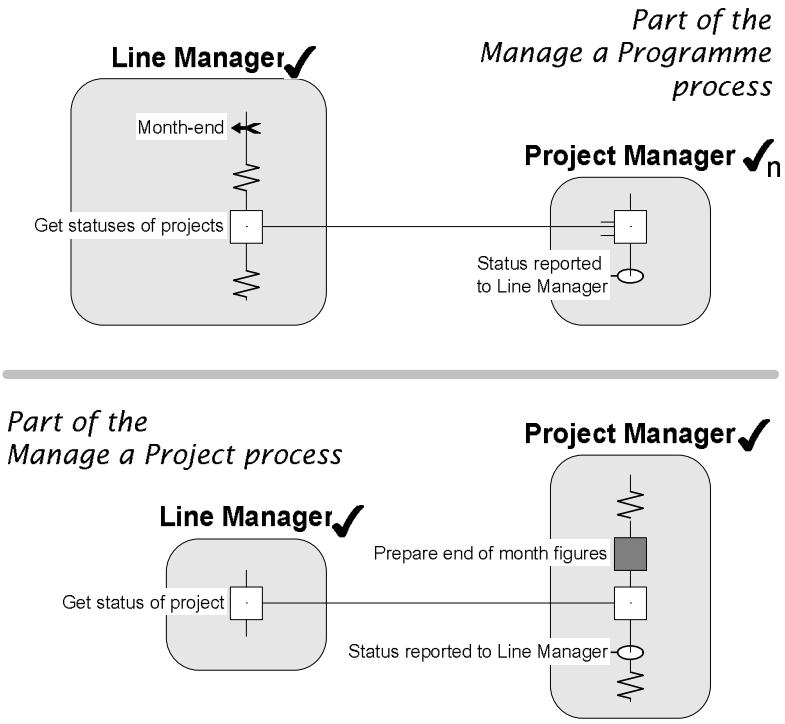
Figure 4-3 – Figure 4-2 slightly reduced



Clearly, in some instances the interaction could be much more complex with a number of state equivalences defined to tie them together. But note that we always do it by giving the same name to states in the *same* role in the processes that are interacting.

Figure 4-4 shows the same basic modelling scheme at work in the situation where a replicated interaction is involved. In the Manage a Programme process, the *Line Manager* goes separately to each *Project Manager* and gets the status of their project. How the *Project Manager* determines that status is the subject of another process altogether: Manage a Project. In the RAD for that process, we have chosen to show the same interaction but now at the appropriate point in the management of a project. To tie the two RADs together, we have labelled the shared state (*Status reported to Line Manager*) in a shared role (*Project Manager*). We could also have labelled the other three states on either side of the part-interactions in the two roles. That is a modelling decision.

Figure 4-4 – Process interaction modelled with two role interactions



In principle, we can of course cut the boundary between two interacting processes at slightly different places, and there are many ways of representing the boundary. Where we draw the boundary and what level of detail we show depends on what we find useful for our purpose. When we move into the next chapter and start to look at how to chunk organisational activity into processes, we shall find that in practice the boundary is generally clear and the representation straightforward.

KEY POINTS

When two processes interact, at least one role will appear in both their RADs.

The interaction is minimally a common state in the shared role. That shared state represents the synchronisation of the two processes.

Whether we show all the shared roles and/or all the shared states and/or all the interactions between the roles is a modelling decision.

ACTIVATION OF PROCESSES

The other important type of relationship between processes is the one where one process is able to 'activate' another, to 'set it going', to 'kick it off', or 'start it up'. The assumption is that if 'process *A* activates process *B*', then process *B* can then go its own way independently of process *A*. The two processes might subsequently want to interact in some way.

As an example, suppose we have a Carry out a Strategic Review process in a company. As the mission, critical success factors, and future strategic goals are examined during the review, new targets are set for different parts of the company, and those targets in turn lead to the need for a number of Tactical Reviews. We can think of the Carry out a Strategic Review process spinning off a number of instances of the Carry out a Tactical Review process. Each of those instances will do its work and feed back to (/interact with) the Carry out a Strategic Review process.

This sort of thing is exactly what goes on in organisations constantly. During the development of a pharmaceutical drug, clinical trials are 'set going' to examine aspects of the safety and efficacy of the proposed drug, and they report back to the 'main process' with the results as and when they complete. Many clinical trials are in progress at any one time. That's the sort of concurrency that runs right through any organisation. In *Riva* we have precisely the language we need to describe what happens in the real world. We will say that 'process *A* activates process *B*', or that that 'process *A* instantiates process *B*.' They have the same meaning: 'An instance of process type *A* instantiates process type *B*.'

We must now say precisely what we mean by saying that one process instantiates another. We shall start by showing how it can be represented in a very minimal way.

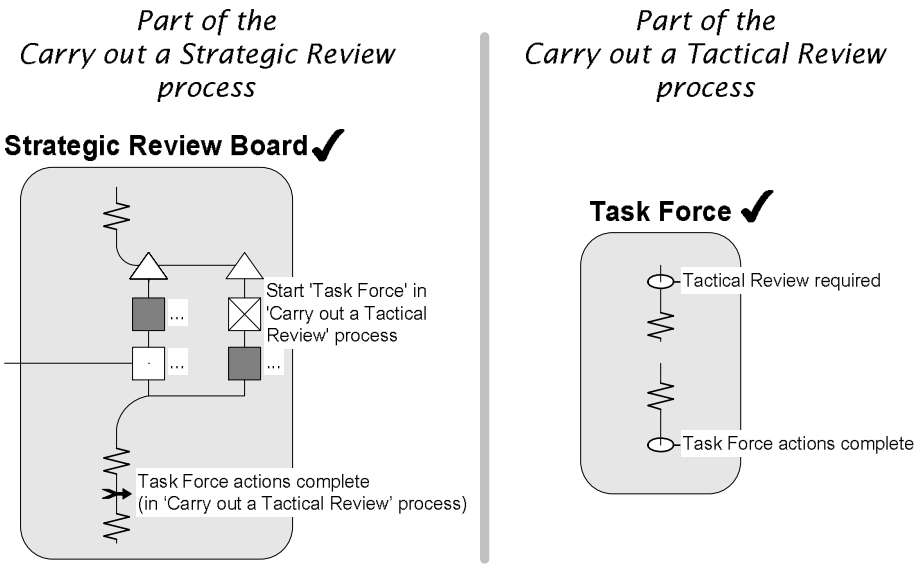
Suppose that, when we model Carry out a Strategic Review, we want to show Carry out a Tactical Review being *activated*. Figure 4-5 shows how we do this. In the activating process – Carry out a Strategic Review – we show an appropriate role (here *Strategic Review Board*) instantiating what we call the *lead role* of the activated process (here *Task Force*). The *Task Force* role is an 'abstract' role: it doesn't have pre-existing instances, in particular such things do not appear on the organisation chart; instances are only created as and when required, and once they have done their work they disappear. The lead role represents the responsibility for the process: hence the term 'lead role'. This reflects real life: when we decide we want to have a Task Force do something for us, we *create the responsibility* for that something: that responsibility is the lead role. Moreover, we can associate the instance of *Task Force* with the instance of the Carry out a Tactical Review process. So when we want to instantiate a process we simply instantiate its lead role – everything then follows.

When we then draw up the RAD for **Carry out a Tactical Review**, we naturally show the lead role, *Task Force*, and, to indicate that it has a pre-existing instance (generated in **Carry out a Strategic Review**) we tick it with a ✓ .

Note that we do not need to show what makes things start in the RAD for **Carry out a Tactical Review**: the ✓ is enough to indicate that the *Task Force* is already instantiated, so we can simply show what it does when it gets going. That said, we might choose to label the state at the start of the role with a suitable caption (e.g. *Tactical Review required*) simply to make this separate RAD that bit more free-standing, so that it can be read on its own.

The RAD for **Carry out a Tactical Review** shows that the role *Task Force* carries out some activity and then, at some point, is in the state *Task Force actions complete*. This state is probably important to the Strategic Review process; the *Strategic Review Board* is not interested in how it is reached, only when it is reached, since it needs to pick up and respond in some way to the results of the Task Force's work. Figure 4-5 shows how we can represent that in a very minimal fashion.

Figure 4-5 – Minimal process activation and interaction

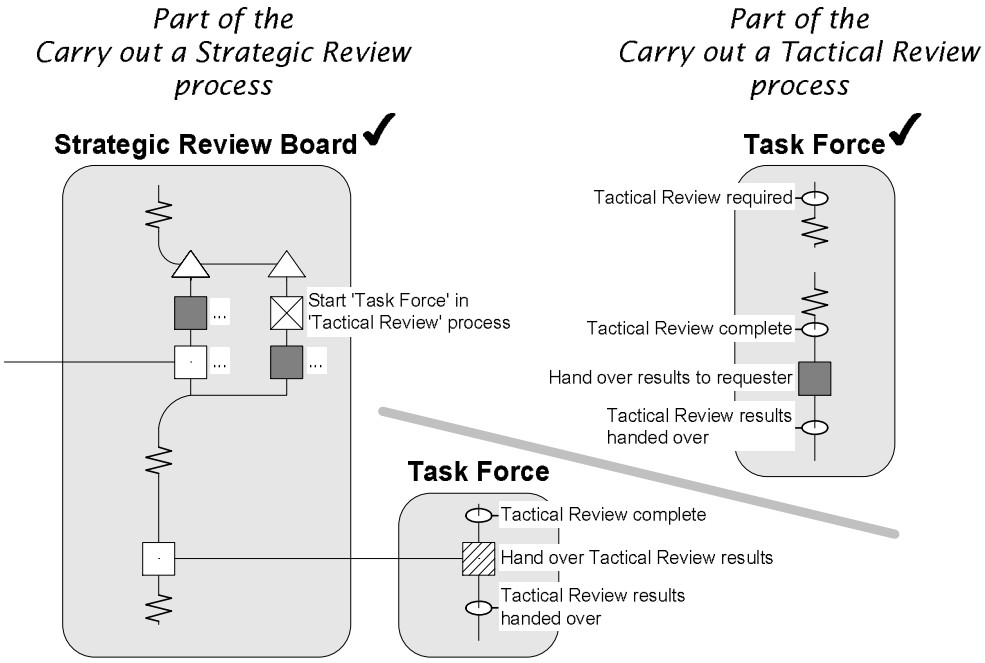


Note how we use the ➞ to spot the completion of the Task Force's work in the activated process, and how (in this model) we have used it to import the results of their work into the activating process. In other words, the calling process synchronises with that state by waiting on a corresponding event (*Task Force actions complete*). We can think of this as a very bare way of representing a process interaction.

Note how the two resulting RADs can either be read entirely separately or be seen as a coherent pair, using this minimal amount of modelling to capture the activation and the subsequent interaction.

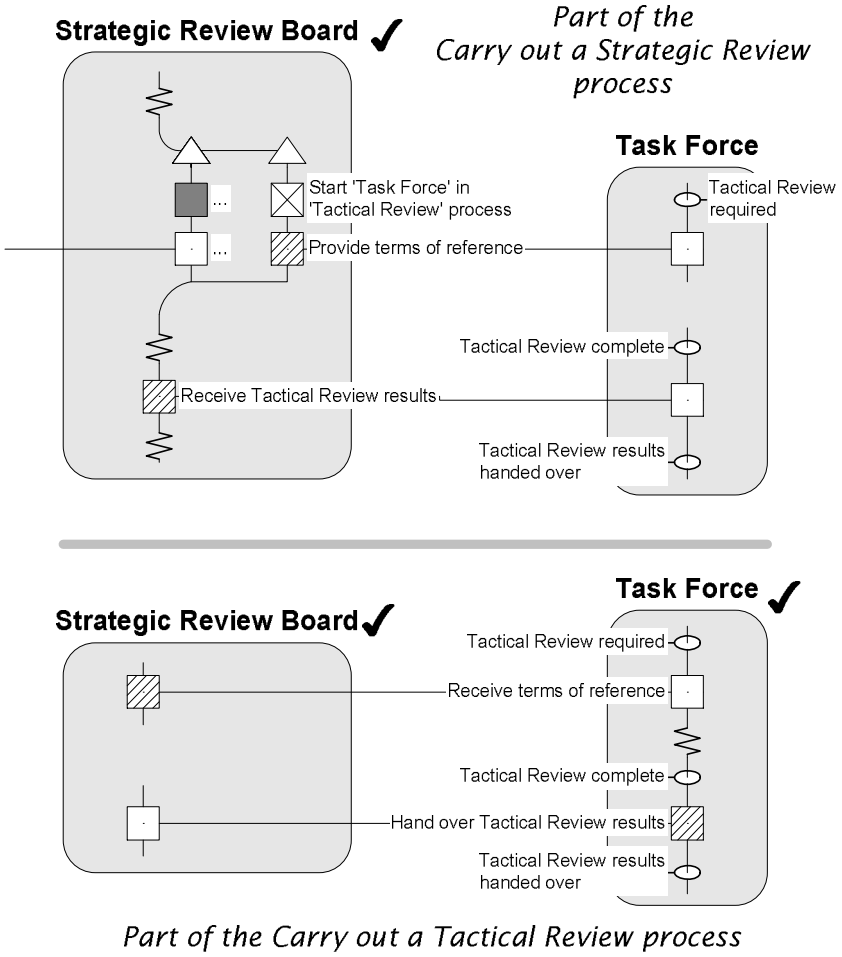
Suppose we didn't want to be quite so minimal in our model. For instance, on the model for Carry out a Strategic Review, we might want to explicitly show the final interaction with *Task Force* where results are returned to the *Strategic Review Board*. In particular, we would like to use the technique we developed above for showing the interaction between two processes, which is what this return of results is. The result would be something like Figure 4-6 in which the shared role *Task Force* appears on both RADs. In Figure 4-5 we used a ➞ to spot completion of the Task Force actions over in Carry out a Tactical Review and to import the results. In Figure 4-6 we use an explicit interaction to hand over the Tactical Review's results, and use our normal process interaction notation to tie this back to Carry out a Tactical Review.

Figure 4-6 – More-than-minimal process activation and subsequent interaction



Let's go one step further and suppose we are interested in the fact that the *Task Force* receives terms of reference when it starts. We might draw the RAD in Figure 4-7. Now we have still shown the role *Task Force* in the Carry out a Strategic Review RAD, but we have added the interaction to pass the terms of reference to the *Task Force*, and have also moved the start boundary of the Carry out a Tactical Review RAD by adding that initial interaction for receiving terms of reference. Take a moment to check how the events and states have been used to tie the two RADs together whilst keeping them free-standing, and to observe the complementary nature of the way the relationships between the two processes are represented in the two RADs. Note the symmetry of the two RADs in Figure 4-7, and compare it with the minimalist model in Figure 4-5.

Figure 4-7 – Full process activation and interaction



The minimal modelling of process activation

To model process activation in a minimal way, we proceed as follows:

- ☞ In the activating process *A*, instantiate the lead role in the activated process *B*.
- ☞ In the activated process *B*, identify (for information) the start state of the lead role, and show that role as having a pre-existing instance.

Later, next to the water-cooler

Pupil: I'm rather surprised by all this. What you're saying is that process A activates process B simply by instantiating B's lead role. That's all that is necessary for things to begin?

Tutor: Yes, it is. Like all things *Riva*, we try to keep things minimal. Let's think through what's happening. When I activate a process it's for a reason: I want something to happen. To make something happen, I create a responsibility for making it happen. The lead role is that responsibility. By instantiating the lead role we have created the responsibility. The process begins. We don't instantiate other roles or actions or whatever … not until there time comes.

Having instantiated the lead role, the activating process *A* might immediately assign actor(s) and hand them resources – their props – in an interaction. We might choose to model this . . . or we might not. By instantiating the lead role, we are – in effect – instantiating the activated process *B*. The instances of *A* and *B* then have their own lives, operating concurrently, interacting as necessary. In some situations the activated process instance will have a shorter life-time than the activating process instance, perhaps 'living' only as long as is necessary to do a job and give the results back to the activating process instance. In Chapter 6, when we construct the organisation's process architecture, we shall see how this sort of 'service' relationship underpins much of the dynamics in an organisation.

Earlier in this chapter I said that organisational activity is the *operation of a changing network of concurrent, interacting process instances*. It should be clear now how such a dynamic network can be seen as the operation of process activation and process interaction.

KEY POINTS

One process activates another by instantiating the latter's lead role.

Subsequently the two process instances operate independently and concurrently.

ENCAPSULATION

Given that our functioning organisation is a network of interacting process instances, we can see that hierarchical decomposition would be a totally inappropriate way of modelling organisational dynamics. To say that 'process *A* is subprocess *A1* plus subprocess *A2* plus . . .' is to ignore reality. It is hard to give any meaning to such a statement. What exactly do we mean by 'plus'?

However, when we draw a RAD there are occasions when we might wish to 'summarise' a whole mass of activity in a single black-box action, not wishing – in this particular perspective – to get into the detail of how it is done. Similarly we might wish to summarise a complex interaction as a simple interaction, not wishing – in this particular perspective – to worry about the detail. We should therefore allow ourselves the possibility of such a modelling convenience. But note that this is a *modelling convenience*, and we are not pretending that processes are hierarchically structured. We must take great care not to imagine that we can accurately model a process by decomposition into smaller and smaller process 'units' or 'subprocesses'.

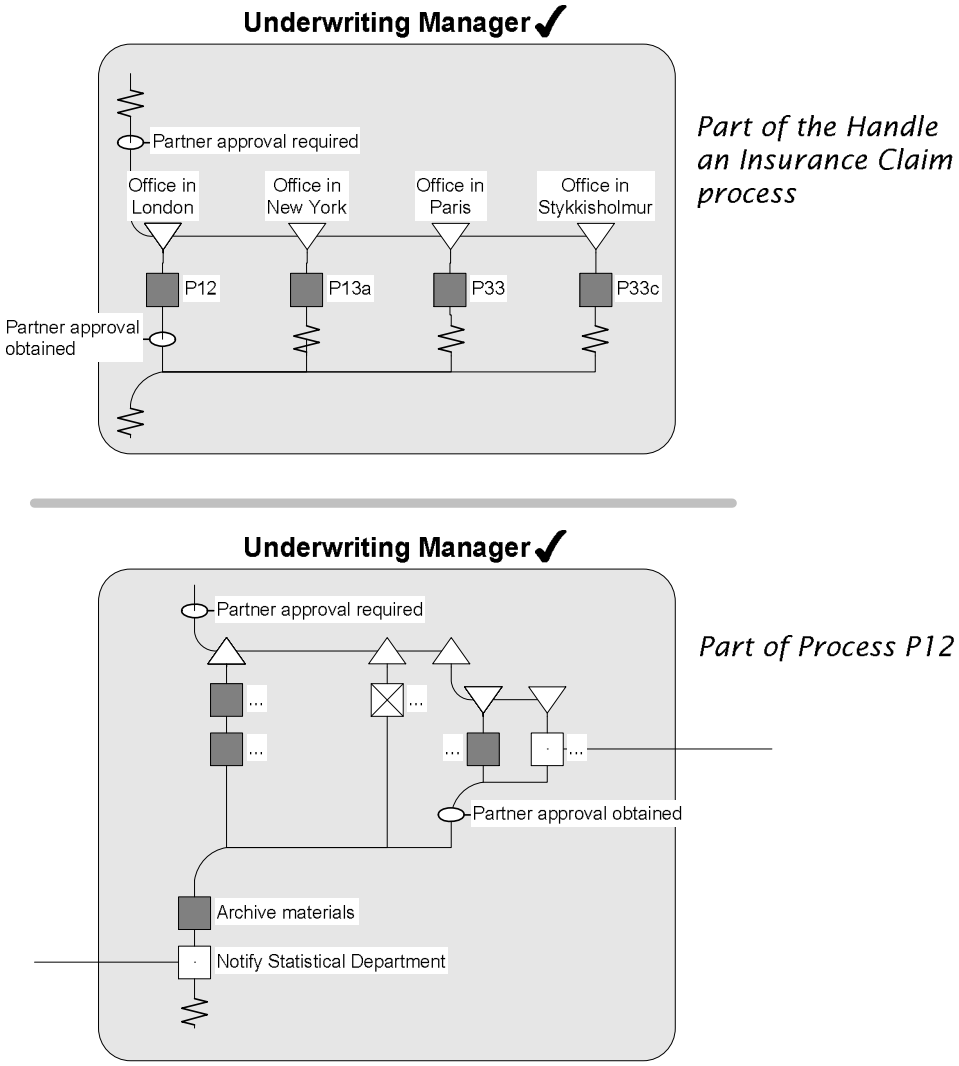
Opening up a black-box action

The question is 'What happens if we "open" a black box on a RAD?' Rather than talking about the 'decomposition' of a black-box action, I prefer to say that we are looking through a window and seeing part of the world from another, more detailed, perspective. We need to understand how that new perspective relates to the black box on the original RAD. In particular, we must remember that an action (like an interaction) takes place over some period of time and at a particular place in the process, and so we need to understand especially the temporal relationships between the two perspectives, the 'main' RAD where we see a black box and the 'encapsulated' RAD which we see when we look through the window.

For instance, take the action *Produce design* back in Figure 2-2. Suppose we open this black box. When we look through it, we might find a whole world of activity involving new roles such as *Client*, *User*, *Quality Assurance*, and *Chief Engineer*. These do not appear on the first RAD, nor are they 'part of' the role *Designing*. But they are all 'encapsulated' in the action *Produce design*. In treating *Produce design* as a process in its own right, we are starting to look at new parts of the world, parts that we were not interested in when we were drawing up the model in Figure 2-2. So we are now going to draw two free-standing RADs whilst showing the relationship between them.

To understand how we open up a black box on a RAD, let us look at a simple example. Suppose we are a life insurance company and suppose that the way the Underwriting Manager handles an application for an insurance policy depends on the office where the application is received; each office has its own procedure for getting partner approval for something. If we were not concerned with the detail of each office's procedure but did want to identify which procedure they used, then we might draw the relevant part of Handle an Insurance Claim as in the upper part of Figure 4-8, showing the *Underwriting Manager* using procedure *P12* in the London office, procedure *P13a* at the New York office, and so on, with each shown as a black-box action.

Figure 4-8 – Action *P12* and *Process P12*



Suppose now that we want to go into detail about the different office procedures, but, for modelling convenience, to show it on its own RAD. How would we do this? Basically, we draw a new RAD for each office procedure's black box. In other words we treat each black box as if it were a complete process and give it a RAD of its own; for instance action *P12* on the main RAD becomes **Process P12** on its own RAD. Remembering my earlier warning about the dangers of decomposition, we have to ask what precisely we mean by saying 'Action *P12* has its own RAD as **Process P12**.'

Firstly, what 'starts' **Process P12**? In **Process P12** we will naturally expect to see the role *Underwriting Manager* from the 'main' process, the role that carries out action *P12*; and,

moreover, we can assume that one instance of *Underwriting Manager* exists when process *P12* runs. This is the 'lead role' of the process: it is the one that pre-exists and that picks up the thread at the beginning. So, the activating condition of action *P12* is also the state at the start of the lead role in Process P12. We will show this by labelling the states accordingly: *Partner approval required*.

Secondly, when we know that *P12* is complete the *Underwriting Manager* is in the state *Partner approval obtained*. This too must appear on the RAD for Process P12, and we have shown this in the lower part of Figure 4-8. But, when we say that 'Action *P12* has its own RAD as Process P12,' do we require that *all* of Process P12 has to be completed before the black box for action *P12* is deemed complete and the main process can proceed? If the answer were 'yes' it would be like treating process *P12* as a 'subroutine' which must 'complete' before 'control passes back to' the main process, to use software jargon. Again, there is a temptation (especially for the software engineer) to impose a tidy block-structured simplicity on the world; but the world is rarely so clean. Instead we are saying that Process P12 must reach *a specified state* before action *P12* is complete. We recognise that there will be some state reached during Process P12 which is the same as the post-condition of action *P12*.

In the second RAD fragment in Figure 4-8, we show part of the RAD for Process P12: the process might be quite complicated, but at some point that same state – *Partner approval obtained* – is reached, at which the thread in Handle an Insurance Claim can proceed, even though there is clearly further procedural activity in Process P12 before it finishes: the Statistical Department must be informed, archiving must be done, and so on. The *Underwriting Manager* in Handle an Insurance Claim does not have to wait for all these things to be done before proceeding. (Process elements have not been labelled in Figure 4-8 where they are not pertinent to the example.)

So there is no sense in which Process P12 and action *P12* are the same thing. All we can say is that some of Process P12 constitutes action *P12*.

Later, next to the water-cooler

Tutor: To demonstrate why I am nervous about this idea of encapsulation, and the danger of imagining there are things called 'sub-processes', let me tell you my suspicion about this Handle an Insurance Claim process: it is that the partners in this insurance company have a stream of requests for approval coming to them, and that that stream of work has to be managed, prioritised, and perhaps resourced, with approval requests being forwarded to partners on the basis of availability, loading etc. In other words that stream of requests is managed somewhere.

Pupil: That sounds like a process itself to me.

Tutor: Exactly. So the RAD we have drawn is a lie: it shows none of that flow management. If we are in a process improvement project and we don't show it we can never recognise it as a potential bottleneck. If we are designing a new process and don't show it we will have a gap in our processes. If we are planning to enact these processes on some form of BPMS, then we shall need to be very clear where such case management occurs. So by thinking in terms of 'sub-

processes' we have made a major modelling error. That's one of the reasons why I am nervous about encapsulation and its use.

Riva's solution to this is the *process architecture*. This is the way we find all the processes in the organisation, in particular the flow management processes. In Chapter 6 we shall see how to expose all the processes, and, as a result, we should see that encapsulation should only be necessary – as a modelling convenience – in a very few cases.

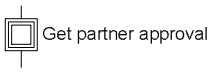
Modelling action encapsulation

Let's summarise the steps in representing the encapsulation of an action:

- ☞ In the 'main' process RAD, show the black-box action *A* to be opened in the appropriate role *R*, label its pre-state, say *ST*, and label its post-state, say *SP*.
- ☞ In the 'encapsulated' process RAD, label the state at the top of the main thread of *R* *ST*. Show the state *SP* at the appropriate place in *R*.

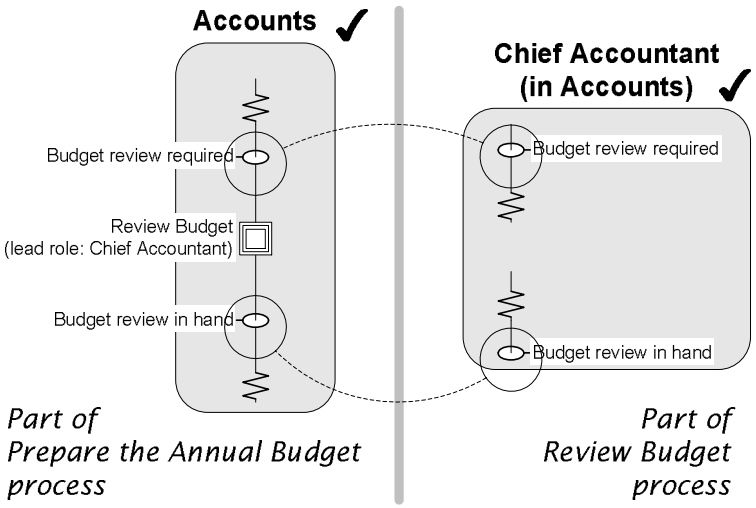
Spend a moment thinking through how the two states – both shown on both RADs – tie the two processes together, much as we might solder ends of wire together to connect electrical circuits.

Finally, let's note that there is a special symbol for an action that has an encapsulated process for it on another RAD:



When we move from the main process to the encapsulated process we might find it useful to work with a sub-role of the role in the main process. So, although we might have the role *Accounts* carrying out the action *Review Budget* in the Prepare the Annual Budget process, when we model the encapsulated 'process' Review Budget in its own RAD, we might show the *Chief Accountant* starting it off – a role that is 'within' *Accounts*. Since we want the RADs to be free-standing, we would tie them together with some extra captioning on the action in the main process RAD, as in Figure 4-9. The dotted lines emphasise the way in which states are used like solder points to connect the main and the encapsulated processes.

Figure 4-9 – Using a sub-role in an encapsulated process



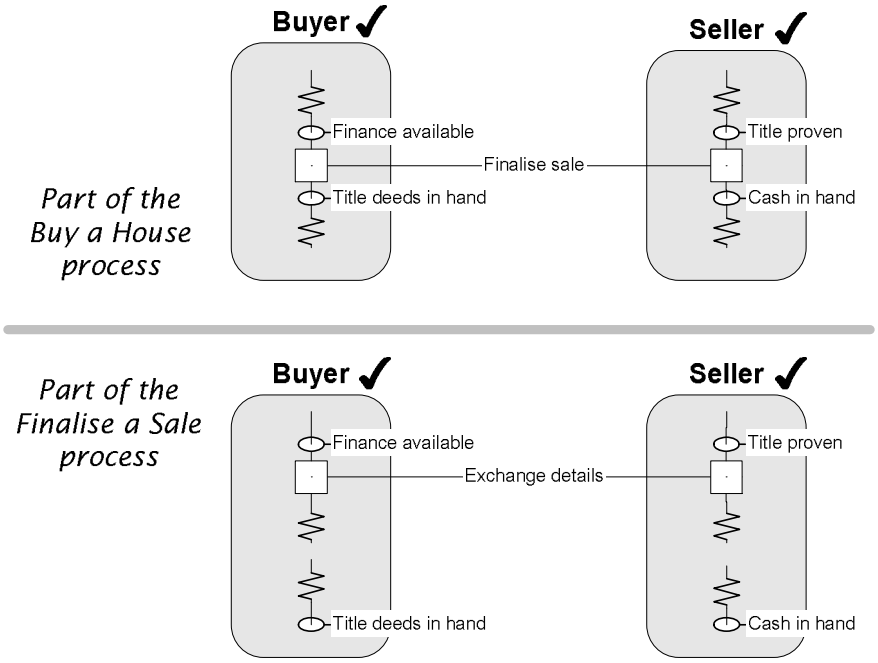
Opening up an interaction

In a RAD we might choose to summarise a complicated interaction in a single element as a modelling convenience. As an example, take a simplification of the interaction I used earlier: 'Two parties meet to discuss, negotiate and agree the price of a piece of work, drawing up the agreement as legal document and obtaining financial securities from a bank.' On a RAD, we might choose to represent this as an atomic interaction between the roles *Buyer* and *Seller*, atomic in that we are not interested – in that model – in any further detail; we simply want to say that *Buyer* and *Seller* have that interaction with that result, and we don't mind how they do it.

Suppose we now choose to 'open up' this interaction. Rather than showing just more detail of what happens between *Buyer* and *Seller* (in the way that we noted when we looked at interactions as 'conversations for action' in Chapter 2), we might look further and find other roles involved, roles which did not appear on the first RAD: *Bank Manager*, *Lawyer* and *Auditor* for example; we might also find new actions that they carry out, and new interactions between them. We have not 'decomposed' the interaction: we have opened it up like a window again and looked at this part of the world from a new angle, an angle which introduces new roles and activities, all of which were of no interest to the first RAD.

As when we opened black-box actions, we need to understand the relationship between the interaction and its 'expansion'. Not surprisingly, we do it by equating the pre-states of the two part-interactions with corresponding starting states in the roles in the expanded process. The post-states of the part-interactions are similarly dealt with. Spend a moment tracing through the example in Figure 4-10. In the expanded Finalise a Sale process, much can happen between the initial and final states.

Figure 4-10 – Encapsulating an interaction



Modelling interaction encapsulation

Let's summarise the steps in representing the encapsulation of an interaction between roles *R1* and *R2*:

- In the main process RAD:
  - show the part-interactions *P1* and *P2* of the interaction to be opened in *R1* and *R2*;
  - label the pre-states of each part-interaction, say *preP1* and *preP2*;
  - label the post-states of each part-interaction, say *postP1* and *postP2*.
- In the encapsulated process RAD:
  - label the state at the top of the main thread of *R1 preP1*;
  - label the state at the top of the main thread of *R2 preP2*;
  - show the states *postP1* and *postP2* at the appropriate places.

Again, spend a moment thinking through how the four states – all shown on both RADs – solder the two models together.

KEY POINTS

Don't use encapsulation.

If you think you must, don't until you have fully understood the process architecture and cannot find the encapsulated 'process' in it.

If you still think you must, ask first whether you have uncovered another 'unit of work' as described in Chapter 6.

If you finally do, use states to show how the beginning and end of the action/interaction being encapsulated translate into the 'expanded' process.

When you have done it, remember that encapsulation is only a modelling convenience and probably doesn't reflect anything in the real world.

Finally, reconsider whether you really should have done it!