3 Dynamism in the process

INTRODUCTION

At the end of Chapter 1, our Tutor pointed out two important messages: instantiation and concurrency. Things happen because many concurrent things happen at the same time. Remember the chevron of geese. Think of a bee-hive. When we draw a RAD we are essentially drawing a static model that shows the relationships between types of things: types of roles, types of actions, and types of interactions. But more importantly it describes the potential *behaviour* of an organisation when it carries out the process that we have modelled. That behaviour arises from instantiation. In this chapter we shall look more closely at how a RAD captures concurrency. We shall do this by showing how we can pick up a RAD and 'run' it – a sort of paper animation. Such a paper exercise is useful for understanding or validating a process model; if we want to enact the process on a BPMS then the topic becomes central as we shall see in Chapter 13.

As we looked at each concept in the RAD notation in Chapter 2 – case refinement, part refinement etc – we found it useful to animate the fragments of RAD in order to understand the sorts of behaviour that we had effectively defined. In this chapter we shall look at the business of animation on a larger scale: that of the entire RAD. How do we look at a RAD and animate it to see what the process *does*?

A RAD describes a process in terms of the relationships between *types* of roles, types of actions, types of interaction, and types of events. When we animate a RAD to see how it works, we look at *instances* of roles, actions, interactions and events, and we are interested in the actual *states* of role instances.

A REMINDER ABOUT STATE

Each line (other than those indicating interactions) represents a potential state of a role instance. Because a role instance can have more than one thread of work active at any one moment, we can be stricter and say that each line represents a potential *sub-state* of a role instance. For instance, when a role instance enters a part refinement with three threads, we can think of its state becoming the 'sum' of the sub-states on each of those three threads. We showed the state of an instance by placing tokens on the appropriate state lines. As the role instance does its work – carrying out actions, taking part in interactions, responding to external events – so the positions of the tokens change: the marking changes to reflect the change of state of the role instance. We can imagine a software tool on our PC that shows us our role instance – RAD-style – with tokens on the state lines to tell us where we have got in carrying

out that role. As we do items of work, so the marking changes to reflect the way we are moving through the process.

If we stood back and looked at a running process we would see a set of role instances, each in its current state. If we asked the question 'What is the state of the process?' we could answer by saying that it is the 'sum' of the states of all those separate role instances. This matches real life: 'How far have we got with dealing with that insurance claim?', 'Well, the loss adjuster is currently waiting to arrange a meeting at the claimant's house, the clerk dealing with it has checked with the police, and we're sorting out liability with the lawyers right now.'

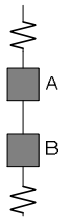
We will be able to tell when the process has reached its goal when the desired states are reached in certain of the role instances.

HOW ROLE INSTANCES 'START'

Let's remember that there are two ways for an instance of a role to arise. The simplest situation is where a role has one or more pre-existing instances. Those instances are in place when the process starts and can begin work. Other roles need to be instantiated once the process is running. Suppose that a process has just started and there is a pre-existing role instance for one of the roles, or a role has just been instantiated in a running process (e.g. a Task Force has been set up). What happens to that role instance? To answer this, let's step back for a moment.

Inside a role we draw all the things a role does: actions and part-interactions. For each of these we can define the activating condition: the state which allows the action or part-interaction to start. We can define those states by making the post-condition of one thing the activation of the next: by connecting them with a line. So, in Figure 3-1, when we have done *A* we can do *B*: or, in our jargon, the post-condition of *A* is the activating condition of *B*. (Strictly, once the instance of *A* has completed, an instance of *B* is created and can be acted by the current actor of the role instance.)

Figure 3-1 – 'After doing A we can do B'



Suppose now that we have a thread of activity whose beginning is as shown in Figure 3-2. What is the activating condition of action *A*? The rule is that, at the head of a thread, the state is 'true', and since the value of this is always 'true' we can deduce that action *A* is always ready to run. In other words, for any instance of the role, there is always a token sitting on the state preceding *A*, as shown in Figure 3-3.

Figure 3-2 – The start of a thread in a role on a RAD

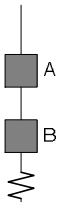
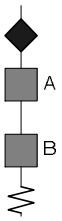


Figure 3-3 – A role instance thread ready to start in a running RAD

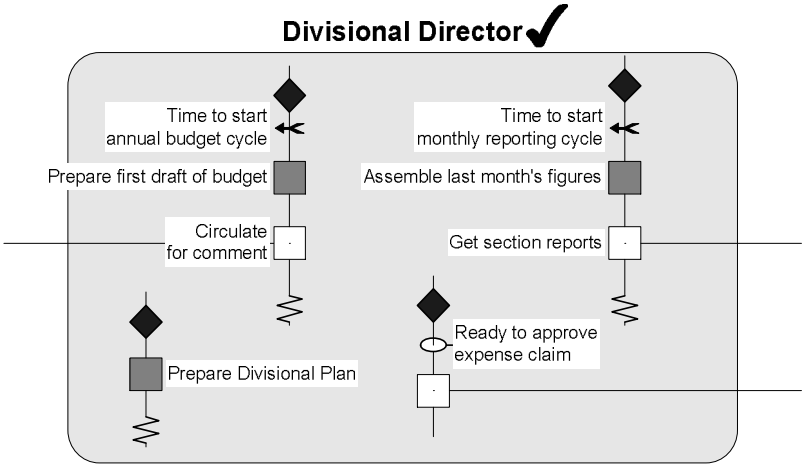


This generalises of course to each such thread in the role: when a role instance starts (in particular when it has just been created), it has a token at the start of every separate thread. What happens on each thread then depends on the first thing on the thread:

- It waits in front of an action, in which case it is up to the actor to decide when to start the action.
- It waits to take part in an interaction.
- It waits for an event to happen.

Let's revisit the fragment of a process we saw in Figure 2-56. When the *Divisional Director* role instance in that process starts there will be a token at the top of each thread, as shown in Figure 3-4. What we observe is the Divisional Director waiting for the moment to come when they must start work on the annual budget; waiting for the moment when it is time to start working on the monthly report; waiting for an interaction with someone wanting an expense claim approved; but able to get straight on with preparing their Divisional Plan. If we restrict ourselves to this small fragment, we have a role instance that is ready to start four threads of activity.

Figure 3-4 – A multi-threaded role instance ready to start in a running RAD



Note how important it becomes when we draw a RAD to mark roles with pre-existing instances if we really want to understand how much concurrency there is at the outset and hence how the process can unfold.

HOW A PROCESS STARTS AND RUNS

When we look at a RAD we want to be able to see how and where it starts. Simple. We just ask ourselves 'Which roles have pre-existing instances?' Each of those instances then starts in the way we have just described: with a token at the head of each free thread. We need say no more. We simply use the rules we developed in Chapter 2 to see how the tokens flow. When the role instance comes across a part refinement, more threads will become active: the concurrency of the role instance will increase. Some threads will merge and others will come to a dead end. If the role instance comes to a replicated part refinement, a multitude of concurrent threads can be started up.

This is a good moment for a caution. When you draw your first RADs you will find yourself unable to resist two things: joining everything up, and creating unnecessary sequences. If you succumb to either of these temptations you will deny the existence of concurrency. For instance, do those four black-box actions you have drawn in a sequence actually have to be done in sequence, or is some parallelism possible?

HOW ROLE INSTANCES 'END'

We have no special notation in *Riva* to mark the 'death' of a role instance in a RAD, no symbol that says 'delete this role instance'. When we want to show this we simply use a black-box action with an appropriate caption: e.g. *Close down the Task Force*, *Close off responsibility for Project Managing*, or *End role of Expense Claimant*. If this troubles you, remember that a role's activity can be made up of many concurrent threads. It is not always a matter of reaching the end of a single thread.

HOW A PROCESS 'ENDS'

A process ends when there are no role instances with something to do … assuming of course that they cannot be revived into activity by an external event that they are waiting on.

Later, next to the water-cooler

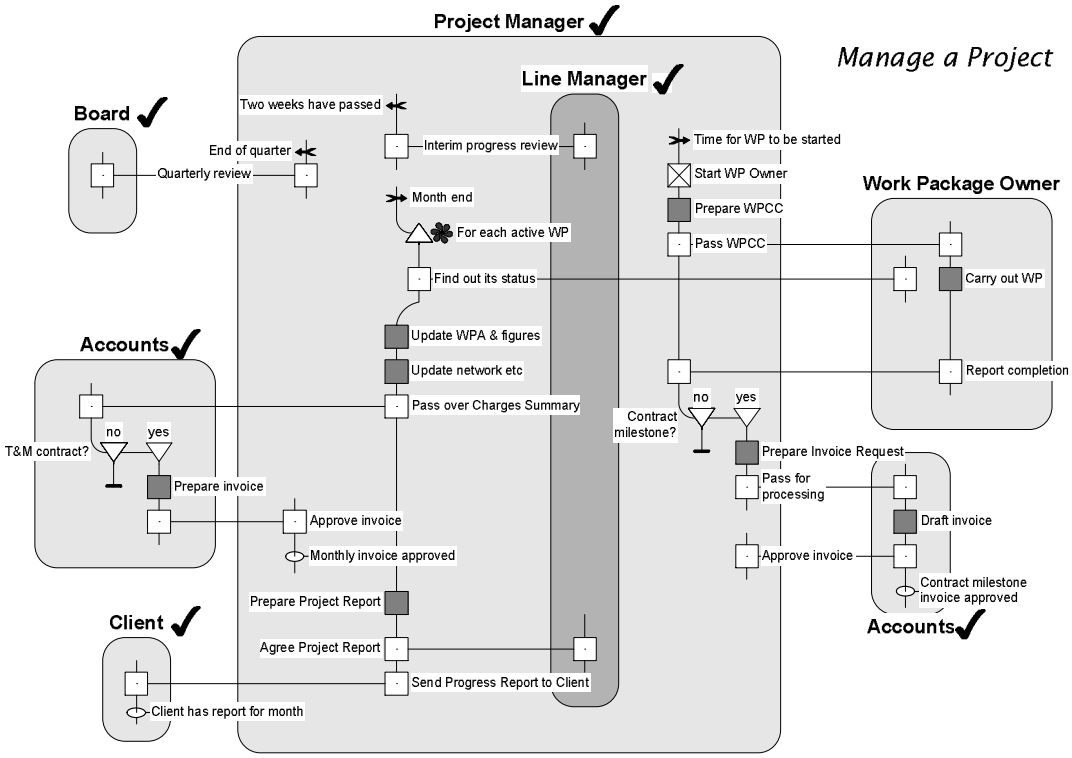
Tutor: I hope you can see now why a RAD is not a flowchart.

Pupil: Well, a flowchart – and, I guess, all the variations on the theme that I've seen being used – represents a simple, single, sequential flow of activity, a single thread, with no possibility of multiple concurrent activity. A swim-lane diagram typically does no more than show the single thread trundling between roles. Even if we allow a thread to divide we aren't getting the true concurrency that happens in the world.

Tutor: Exactly. It is all too easy to model a process as if it is some kind of simple sequence – processes are hardly ever like that. If they were, most people in the building would be standing still at any one moment, waiting for their turn to come. Instead, as a process unfolds, there is a constantly changing flux of instances that ebbs and flows. We cannot think of a process as a sequence of activities, some beads strung on a string, pieces of meat on a kebab stick. We must accept that it is more like … well, like the workings of an organisation! So although a RAD is a static model that shows relationships between types of things (roles, actions, interactions), it actually captures the **potential** dynamics, what happens when things run and instances happen.

Let's look at the RAD in Figure 3-5. Tell me about the concurrency in it.

Figure 3-5 – 'Manage a project'



Pupil: OK. When the process starts there is a Project Manager, a Board, an Accounts Department, a Client, and a Line Manager. The model seems to be mostly about the Project Manager, who has six separate threads. One thread is triggered at the end of a quarter and they then have a quarterly review with the Board. Another triggers every two weeks to have an interim progress review with the Line Manager. Then there is another at the end of the month which is to do with getting the monthly report written and sent off to the client.

The most interesting seems to be one to do with handing out pieces of work – 'work packages'. Each work package has a responsibility associated with it in the form of a *Work Package Owner* role. When one of those owners is started they get a WPCC – some sort of terms of reference I guess – and they do the work – no details about that – and finally tell the Project Manager when it's finished – the Project Manager waits to hear about that.

Tutor: So tell me about the concurrency in this process.

Pupil: That work-package-related thread can fire as many times as necessary, so, in principle, at any one moment the Project Manager could be having a quarterly review with the Board, having an interim progress review with the Line

Manager, preparing the monthly report – which itself could involve as many threads as there are active work packages – and looking after as many threads again for the work packages themselves. So if there are N current work packages, the Project Manager could have 2+2N concurrent threads of work in hand – that sounds like project management!

And of course there are N instances of *Work Package Owner*, each of which has two threads going: one doing the work package and the other ready to report on status to the *Project Manager*.

Tutor: Right. But this amount of concurrency is still not enough. So far, we have only talked about the concurrency possible in one process. We now have to step back and get a handle on how the collection of different processes work together and we shall then add whole new levels of concurrency.

KEY POINTS

When an individual process (instance) runs there is concurrent activity.

Each role type can have zero, one or more independently active instances at any one moment.

Within each role instance, there can be zero, one or more threads of activity operating independently and concurrently at any one moment.