1 Basic process concepts

WHAT HAPPENS IN THE WORLD?

Our analysis of the needs of the process modeller leads us naturally to the basic concepts that we will be looking for in our process modelling language – the vocabulary that will help us say the things we want to say to answer the questions we want to answer. This chapter will look at those basic concepts, before we go on to model them in Chapter 2.

Processes are divided over *roles*

Central to our notion of a process is that it is about *people* doing business – how they do it, how they think they do it, how they are supposed to do it, how they might do it better or differently, and so on. People are central. So should we model a process by saying what Mavis does, what Bill does, and what Charlie does? Perhaps we should draw a box labelled 'Charlie' and in that box describe, in some way, what Charlie does. Similarly we might have boxes for Mavis and Bill, and for the other people involved in the process. This doesn't feel right of course, because, in an organisation, people do things not because they are themselves but because they have a *responsibility* in the organisation; they are perhaps paid to carry out that responsibility: they have a *role* in the organisation. So those boxes on our diagram should really be about roles, responsibilities that are carried out – *acted*, as we shall say – by individuals or perhaps groups of individuals.

For now, let's just think of a role as a responsibility of some sort, perhaps a hat that someone – an *actor* – can wear. I go to work, and as I walk in the door of my office I put on my hat of responsibility.

In the process of getting a book published, we find roles such as the author, the publisher's editor, the typesetter, the copy editor, the publicist, the printer and the bookseller. Each has their own responsibility.

Some roles are carried out by individuals: Martyn Ould is (carrying out the responsibility of) the author of this book. Other roles might be carried out by a group of people: the responsibilities of the Accounts Department are carried out by the people who work in the Accounts Department.

Finally, a role has the things necessary to do its actions. They can include its physical environment, work in progress, materials, resources in various forms, and tools.

We shall call these the role's *props*. Those props might reside permanently within the 'body' of the role; for instance, the Project Manager has a set of plans that they use during the project. Or props might be passed to the role; for instance, the Project Manager receives terms of reference for the project they are to manage.

Individuals *do* things following rules

In carrying out my responsibility – in performing my role – I do things. We shall call those things *actions*. So, roles carry out actions.

In the business of getting a book published, various people write the book, prepare the index, draw the diagrams, check the copy, set the type, print the book, and get copies to the shops. By predicting the target market place, the publisher decides whether the book will come out first in hardback or paperback. The designer decides on typographical issues such as layout and typeface. The printer prints the pages and binds the books, ready for the retailer to sell to the public. Each of these actions does something in a way that we hope adds value and contributes to the business of achieving the goals of the process.

Things are done in a particular order: we cannot label a product until we have the label and the product; we cannot typeset a book until the copy is ready; we must lay the foundations of the house before we build the walls; we need to get the budget approved before we spend money.

Sometimes, the way things are done is determined by the outcome of decisions about the state of things: if the customer is late in paying, we charge interest to their account; if the house being insured is in a flood plain, we estimate the insurance premiums differently; if it is the weekend, we charge double; if the e-mail system is down, we send a fax.

Sometimes, how we do things is governed strictly:

- *Policy*. Our company might have a policy that nobody can approve their own work, or that all product tests must be carried out by someone independent of the production group.
- *Procedures*. Some of our work might be regulated and defined in the form of procedures. For example, in order to control financial commitment closely we might have procedures for planning projects, reporting project status and purchasing. We might have procedures to make interfaces efficient: all requests for training follow the same procedure so that people requesting training do not need to invent how to make a request, and the people handling requests know in what form they will arrive. Procedures might say who can sign off purchases above what value, who can authorise a change in

production schedules, or who can cancel a project. (Note that I have used the word 'procedure' here in the strict ISO sense of standardised activity.)

- ☞ *Standards*. A standard might be laid down to define a common appearance or content for something produced during the process. We might require that project plans and reports conform to a particular layout. This might be for efficiency (everyone knows what a report looks like and where to find the information they are interested in) or for control (we want to ensure that certain topics always get covered and certain information is always included).

All these are *business rules* that govern 'how things get done around here'.

Individuals within a group *interact*

Also central to our notion of process is that people, in carrying out their roles, sometimes do things *together*: they collaborate. To collaborate, they carry out some actions together. We shall say that they *interact* and that they *have interactions*. For instance:

You and I discuss something.

You and I negotiate.

I contract with you to do something.

I pass you some information.

I delegate a task to you.

I ask you for something.

I give you authority to do something.

You and I agree on an action.

You and I jointly approve something.

You report your status to me.

I oversee something you are doing.

You pass me the results of your work.

I instruct you.

You and I work on something together.

I wait for you to do something.

I chase the progress of your work.

Note how rich interactions can be. An interaction isn't just about the locus of activity moving from one role to another; real business-oriented, value-adding things can happen in an interaction. Interactions are just as vital to the process as the actions that

individual roles carry out on their own. Ineffective interactions can be as damaging to a process as ineffective actions. Slow interactions can affect cycle times as much as slow actions.

So, a role involves a set of actions and interactions which are governed by rules which, taken together, carry out a particular responsibility. A process is the sum of the contributions of the individuals acting as individuals and collaborating as a group. If we get the sum right, the process achieves its goal. So …

Processes have *goals*

A process is done for a reason: it has a *goal*. Sometimes the goal might not be reached and there is some other *outcome*, perhaps an undesirable one, some sort of failure.

For instance, the goal of a process might be to deliver a computer system, to maintain positive cash flow, to organise the efficient use of a piece of plant, to provide a medical care service to a customer, or to manage a research budget. It must be possible to see from our process models how a process is achieving the goals set for it, and ideally to be able to identify the point(s) in the process where those goals can be said to have been achieved.

KEY POINTS

A *process* is a coherent set of actions carried out by a collaborating set of roles to achieve a goal.

A *role* is a responsibility within a process.

An *actor* carries out a role.

A role carries out *actions* following business rules.

A role has *props* which it uses to carry out its responsibility.

Roles have *interactions* in order to collaborate.

A process has *goals* and *outcomes*.

These concepts are central to *Riva*: role, actor, action, interaction, goal and outcome. Let's examine each in more detail.

ROLES

Suppose we walk into a supermarket company and identify the things that we might think of as roles, in some yet-to-be-defined sense. Here are some: *Store Manager*, *Shelf Stacker*, *Checkout Assistant*, *Shop-floor Assistant*, *Security Guard*, *Finance Clerk*, *Warehouse Person*. If we walked into a publishing company we might choose *Author*,

Editor, *Commissioning Editor*, *Marketing Manager*, *Copy Editor*, *Production Planner*. If we walk into a software engineering company we might find *Project Manager*, *Programmer*, *System Tester*, *Chief Architect*, *Designer*, *Configuration Controller*, *Finance Director*.

Roles of this sort are rather like job-titles, the sort of thing an individual might have printed on their business card. Or they might be boxes on the organisation chart. Or both. For instance, there will be a box on the organisation chart labelled *CEO*. There might be several labelled *Store Manager* — one per store. Such roles have a part to play, responsibilities to carry out, in the organisation. However, there will be no box labelled *Author* on the organisation chart for our publishing company, even though authors have a lot to do with the processes of a publishing company. *Author* is not a post in the organisation, but a real author might view it as their job-title, what appears in their passport as their occupation, or on their business card.

Moreover, I could stand back from the organisation a little and see it in terms of rather 'larger' roles: *Finance*, *Production*, *Editorial*, *Marketing*. Here I am clearly spotting functional *groups*, each of which has one or more responsibilities in the organisation. What is the relationship between these sorts of roles and the 'smaller' roles we identified above? We might expect that the *Finance Director* carries out some of the responsibilities carried out by *Finance*, and that some other parts are carried out by (say) *Finance Clerk*. Of course, we can't divide up all the responsibilities of *Finance* and hand the bits to the 'smaller' roles because responsibility isn't like that: in many cases a responsibility of *Finance* can only be carried out through the cooperation of a number of roles operating within the Finance Department. I am suggesting here that it is dangerous to think of roles nested in some sort of hierarchy: hierarchies smack strongly of decomposition and if we try to decompose responsibility by cutting it up into lumps we shall lose the notion of cooperation and collaboration which is what makes so much happen in organisations. Things will fall apart.

There is probably only one Finance Department in our supermarket company. But we might identify *Store* as a role in the organisation and there will be lots of them. A bank may have many branches; each branch plays the same role and has the same responsibilities.

There are other sorts of role which are neither posts, nor job-titles, nor functional groups. Take *Customer* for instance. When I walk into a shop, I take on the role of customer, and in that role I interact with roles in the shop. I don't have 'Customer' in my passport, any more than I have the role 'Expense Claimant' on my business card. But there are times — month-ends typically — when I do put on the hat of *Expense Claimant* in order to claim my expenses. Both of these are *transient* roles that I take

on at appropriate times. Here are some similar transient roles: *Job Applicant*, *Hospital Patient*, *Complainant*, *Employee*, *Defendant* and *House Vendor*.

So far, our roles have been rather tangible things: you can kick them, or at least draw some sort of line round them somewhere. But we could take the notion of role-as-responsibility a step further and identify rather abstract things — pure responsibilities — as roles. For instance, *Large Claim Approval* might be a responsibility in an insurance company. When claims over a certain size are about to be paid, we might require that they are scrutinised independently before going through. The responsibility for carrying out that scrutiny has a reality — though we might not be able to kick it — and an identity. In practice we shall probably allocate that responsibility to a role of one of our other, more concrete types, a particular post or job-title, say.

Let's pull these threads together and list the different sorts of roles that we might have:

- A unique functional position or post: e.g. *Head of Analysis Department*, *CEO*. Such a position or post is unique in that there is one and only one in the organisation. There is only one Analysis Department and that brings with it the responsibility of heading it which we wrap up in the unique post of Head of Analysis Department.
- A generic functional position or post: e.g. *Head of Department*, *Divisional Manager*. Each Department brings with it the responsibility for heading that Department. Each Division brings with it the responsibility for managing that Division.
- A unique functional group: e.g. *Document Registry*, *Accounts*, *The Government*. Such a group is 'unique' in that there is one and only one in the organisation. It has a part to play in the organisation's activity.
- A generic functional group: e.g. *Department*, *Branch*, *Subsidiary*. There may be many Departments or Branches. They all have the same responsibility in the organisation.
- A generic type of person: e.g. *Trade Union Member*, *Customer*, *Purchaser*, *Expense Claimant*. Typically a rather transient role — one that comes and goes — but there may be many at any one time.
- An abstraction: e.g. *Progress Chasing*. Such an abstract role is almost a definition of the responsibility itself, rather than a label of a post that brings with it a responsibility.

KEY POINTS

Some roles are unique in the organisation and some are generic and replicated.

Some roles are concrete and some are abstract.

Roles are types; role instances are acted

We have so far talked rather glibly of roles, and of roles being acted. But we need to introduce an important subtlety here.

In the world of cars, there are many different types, for instance a Rover, or a Saab. Given a type of car, we will find on the roads a number of cars of that type. I have a car of the Saab type; a colleague has a car of the same type; our two cars are both *instances* of the Saab type.

These notions of a *type* and the *instances* of the type are very important in the *Riva* method. In fact, a role in *Riva* is actually a *type*. That is, in principle there can be a number of different *instances* or *occurrences* of a role type active at any one time within an organisation. In this book I shall use the word *instance* in this sense of 'occurrence'. (Software engineers will recognise the terminology of object orientation.)

As an example, in a pharmaceutical R&D company – let us call it Hill Pharm – we might have a number of projects running at any one time, one for each new drug that is undergoing development. Suppose we define the role *Project Manager* for drug development projects: it is the role to do with the responsibility for managing a project. We then know that there must be an *instance* of the *Project Manager* role for the Xanthropol project, another instance for the Bisintifil project, and yet another for the Viniliton project. Each project has a separate responsibility associated with it.

When the Xanthropol project started, the responsibility for managing it was created. Suppose we appointed Bill to be the project manager. In *Riva* terminology, 'We assigned Bill *as actor* of the instance of *Project Manager* for the Xanthropol project.' We might even say 'We cast Bill as actor of the instance of *Project Manager* for the Xanthropol project.'

Perhaps Bill then resigned, and Jill took over. The instance remained constant, but the actor changed. Jill stepped into Bill's shoes; she took over the responsibility. But perhaps, just for a day after Bill's resignation, the project had no Project Manager. In other words the Xanthropol instance of the role *Project Manager* had no actor that day, no-one to take responsibility. The important point to note here is that as long as a project exists, the responsibility for managing it also exists, *whether or not anyone is carrying it out, or acting it*. In other words, the role instance exists whether or not it has an actor.

So, we need to be careful to distinguish between the role instance — which has its own existence — and its current actor, if any.

Let's take a further example, *Prime Minister* is a role type. I can point to several instances of this role around the world: 'Prime Minister of the UK', 'Prime Minister of Australia', and 'Prime Minister of Canada', to name three. These role instances exist independently of whomever is acting them at any one time. As I write, a man called Tony Blair is acting the instance 'Prime Minister of the UK'. This instance was formerly acted by John Major. In fact, Tony Blair acts a number of other role instances concurrently, one being that of 'Member of Parliament for Sedgefield', itself an instance of the role type *Member of Parliament*. In the *Riva* view of the world, an election is simply a way of choosing an actor for a role instance. If an MP dies, the role instance they were acting continues but there is no-one to act it, no-one to carry out the responsibilities of being MP for the area, until there has been an election.

If you approached an employee going into the publisher's building and asked them what they did, they might say 'I'm a Copy Editor.' They are saying 'I wear the hat of Copy Editor.' At home they do not wear that hat. (Let's assume they don't take work home.) So, as they walk into the building they don that hat and start carrying out the responsibility it implies.

Quite clearly, the role instance is separate from the person who acts it. That employee we stopped: when they go into a shop they don the hat of Customer. When they leave, they take the hat off. When they re-enter the publisher's building they don the hat of Copy Editor again. And while they are wearing that hat they take part in various processes in which they have a part — a responsibility — to play.

Note how strong the theatrical analogy is: actors play roles. You might say 'I play the part of *CEO* in this company,' in the same way that someone else might say they played the part of Hamlet in a recent performance.

Now, an actor can take many forms. It might be a single person, a group of people, a computer, a person or group assisted by computers, a machine tool, a company — indeed, any agent in the real-world capable of carrying out the work in the role.

When we model a process, we might (or might not) be concerned to understand where role instances come from and how actors get allocated. For instance, if we are modelling how a new drug development project gets underway, we will at some point model how the project comes into being ('is born') and with that will come the creation of the responsibility for the management of that project: an instance of the role *Project Manager* will be created. We may also model, as part of the process, how a person is chosen to act that role instance — how they are 'cast' for the role; indeed, we shall allocate the responsibility of choosing someone to another role in the process, perhaps a

Therapeutic Area Director. And we may go further and model how a replacement Project Manager is found if an existing one leaves the project.

A word of warning is in order here. Our first inclination when choosing roles might be to use names like *Store Manager*, but this can make it too seductive to identify roles solely with post or job titles, forgetting that job titles are invariably bundles of responsibilities in different roles in a number of different processes. For instance, the post of CEO could be treated as a role, but it is clearly a post that has a part to play in many processes in the company's activity: as authoriser of large purchase orders, as setter of the company's strategy, and as an important player in maintaining the company's relationship with its clients. Put another way, the CEO has a number of responsibilities in the organisation. If we show the role *CEO* on the process model for, say, the process of setting company strategy, we must remember therefore that we are only saying 'This is the responsibility that the post of CEO has in this process' or 'This is the role of the CEO in this process'; we are not saying 'This is the CEO role.'

Equally, it is all too tempting to identify parts of the organisation as roles: departments, divisions, sections, or whatever. For instance, a Finance Department, though forming a readily identifiable group of people, might actually participate in a number of separate though related processes including remunerating staff (by paying people), purchasing (by placing orders and paying suppliers), and handling the company's cash-flow (by invoicing clients, chasing bad debts and negotiating with the bank). It has a host of responsibilities.

Whether or not we associate posts or job titles or parts of the organisation with roles will, like so many similar decisions, depend on why we are modelling the process, and later in the book we shall look at the various situations in which different approaches are appropriate. Some process models that I have prepared for clients have had, on a single page, roles from each of the different forms.

KEY POINTS

The *role type* defines the role.

A role type can have a number of *instances*.

Role instances operate independently and concurrently.

Each instance can be *acted* by one or more people or no-one at a given moment in time.

If appropriate, we can model the creation of new role instances.

If appropriate, we can model the casting of actors in role instances.

The dynamics of roles, role instances, and actors

Let's explore the relationship between roles, role instances and actors a little further.

In the case of Hill Pharm projects, we would expect to see only one instance of the role *Project Manager* being acted on a given drug development project, i.e. within one 'running' of the process that we might call Develop a New Drug. This is a situation where at most one instance of the role will exist when a process runs. Across the company there would of course be a number of instances of the role, one per active project.

The single instance of *Project Manager* could be acted by different people at different times during the life-time of its project: Jack might be acting as project manager until March 19th, when he hands over the role (instance) to Jill. In a well-organised company we would not expect to discover that at some moment both Jack *and* Jill were acting the role, or a moment when nobody was acting it. So here is a case where we expect to find a one-to-one mapping between role instance and actor at any one time, though that mapping might change over time.

Let's think about a software product company which has a number of development projects in progress at any one time. Each project is responsible for the development and bringing to market of one new software product. Let's take one such project. Within that project, there could be several instances of the role of *Designer*, each being acted by a designer-type person, and, in a large project, perhaps hundreds of instances of *Software Programmer*, each being acted by a software-programmer-type person. So, when the Develop a New Software Product process is running for a particular software product, we would find one *Project Manager* instance, a number of *Designer* instances, and many *Software Programmer* instances. When we draw the model of the Develop a New Software Product process, we shall only describe the role *types*, and hence imply that all programmers, for instance, follow the same procedure. If, in the real-life process, some programmers follow one procedure and others follow another, we would expect to see two different role types in our process model. For instance, novice programmers might be constrained by a rather detailed procedure to ensure that they do not do anything that threatens the project (and incidentally to train them in good practice), whilst experienced programmers might follow a less rigorously defined procedure which recognises their greater expertise and grants them greater discretion. Our process model might therefore contain two role types that have the same goal but operate differently: *Experienced Software Programmer* and *Novice Software Programmer*.

In our Develop a New Software Product process, we might well have somewhere a role with responsibility for controlling change to specifications, designs, code and so on: let us call it the *Change Controller* role. The *Change Controller* role will probably only have one instance on a project and that single instance might be acted by a whole

team of people. Conversely, one (physical) actor might be acting several role instances simultaneously. For instance, on a small project, one person, Jill, might be acting the single instance of *Project Manager*, the single instance of *Change Controller*, and perhaps one of two instances of *Programmer*. Here is a case where the mapping from actors to role instances might be one-to-many.

Let's now look at how those role instances may come and go, and to do this let's revisit the different forms that a role can take and look at how the number of instances and actors can change over time for each.

☞ A unique functional group: e.g. *Document Registry*, *Accounts*.

Within a given organisation we would expect to find only one instance of a (named) functional group such as these. In other words the role type (e.g. *Accounts*) has only one instance and that instance is effectively permanent; that is, we can take it as permanent for the purposes of the process model in which the role appears. The actors of the single role instance are of course the people (and perhaps computers) who make up the actual Accounts Department, and, while the instance has a 'permanent' existence (with the same qualification as above), the people who act it will change. Today I might be a member of Accounts, helping act the role; tomorrow I might have resigned and you might have taken my place.

Suppose the 'organisation' we are concerned with is a nation. Then *The Government* will be a role, it will have one instance, and the actors will be those currently in power. Moreover, unless anarchy breaks out, the single instance is there for all time, with a succession of different actors acting it.

Summarising, there will be a single permanent instance of such a role, with variable actor(s).

☞ A unique functional position or post: e.g. *Head of Analysis Department*, *CEO*.

The situation here is similar to the unique functional group case: there is only one instance of the type. For instance, there is only one Analysis Department and it has only one Head of Department post; similarly there is only one CEO post in this organisation. The holder (actor) of such a post can change of course, but the post (role instance) is probably permanent for the purpose of the model.

Summarising, there will be a single permanent instance of such a role, with variable actor(s).

☞ A generic functional group: e.g. *Department*, *Branch*.

Such a group will appear in a process model when we want to refer to *any* Department, Branch etc and do not want to be specific about *which* Department we are concerned with, or we want to allow *any* department to play a particular part in the process. When the process runs, there may be any number of instances of the role – the branch in Oxford, the branch in Chicago, the branch in Auckland, etc. These instances will be permanent for the duration of the process, unless of course the

process is about the creation and closure of branches. The actors of this sort of role can of course change: staff at a given branch come and go.

☞ A generic functional position or post: e.g. *Head of Department*, *Divisional Manager*.

The situation here is slightly different. At any one time, there will, according to the organisational structure, be a fixed number of such posts: Head of Research Department, Head of Marketing Department, Head of Production Department, and so on. We might want to consider each of these as an instance of the role *Head of Department*. And each role instance – each Department Head post – will have an actor: the current holder of that post.

Summarising, in general the set of instances will remain fixed for a given process, but there will be a change in actors as people are put in those posts and leave them.

☞ A generic type of person: e.g. Trade Union Member, Customer, Purchaser, Expense Claimant

This is like the generic functional group: when a process runs, one or more instances will be created, but each will be identifiable with an individual, and the role instance is in a one-to-one relationship with that individual.

If an instance of *Customer* comes into being it will be associated with and acted by a single person; the actor will not change. During the handling of a complaint from Mr Bloggs we do not expect to see Mrs Featherstonehaugh take over at some point: the *Complainant* instance's actor remains constant.

In the process Sell a House, we shall have a role *Vendor*, representing the seller of the house concerned. For a given house, there is one vendor, so the role will have at most one instance during the process. And the actor of that instance, the person or group selling the house, is very unlikely to change during the sale.

On the other hand, in the process Find and Buy a House, the role *Vendor* will represent the sellers of houses we might buy. Instances of *Vendor* will come and go, as houses come onto the market and come off it during the process. Each instance will be acted by a fixed actor: the owner of the house concerned.

The *Expense Claimant* role is the hat we put on when we want to claim expenses; it is the role we act to claim expenses. Our main role might be *Supervisor*, but once a month we 'slip into' the role of *Expense Claimant*.

☞ An abstraction: e.g. Progress Chasing, Project Managing

This is a case where a gerund (an -ing word which is a noun) makes a good name for the role. It can be particularly useful to name an area of responsibility, rather than a box on the organogram for the organisation, such as a post or a department. I might start *Progress Chasing* an invoice but

someone else might take over at some point if I ask them to; and there could be any number of us progress-chasing various items at any one time. Be careful not to think of this as an action: it is a responsibility within which certain actions will be carried out, but we have yet to look inside the box and examine actions.

One apparent flavour of role that we need to be careful with is a *rank* or *job title*: e.g. *Principal Analyst*, *Senior Engineer Grade 5*. This is a badge that people have, that they carry round the organisation, and that brings responsibilities or entitles them to carry out certain responsibilities. When we label a role in a process with such a 'badge' we are saying 'Anyone acting this role – carrying out these responsibilities in the process – must have this badge.'

KEY POINTS

A role instance generally exists independently of an actor to act it.

Generally, the actor of a role instance can change.

At a given moment, there might be no-one acting a given role instance.

Role instances start other role instances

We have seen how some roles have *permanent* instances, permanent in that they are there when the process concerned starts and still there when it finishes. (Remember we are looking at a single process at the moment.) Suppose we are modelling the Prepare the Annual Budget process. We might expect that the unique role *CEO* plays a part. When the process of preparing the budget starts, that instance already exists; (moreover, when the process is over and the budget is ready, the instance is still there). To be precise, we shall say that 'The role *CEO* has a pre-existing instance for the process.'

However, some roles may not have instances when the process starts. Suppose we are modelling a process called Develop a Software Product. We could imagine that during the early phases of the process/project, we would want to create a role instance to manage change requests, and appoint someone to carry out that role instance. Once the instance is in place and has an actor assigned, the management of change requests can begin. Clearly in this situation we will need instances to be created during the process, and it will not come as a surprise that one role instance can cause the creation of new role instances to create new responsibilities. This corresponds to what happens in real-life: one responsibility can create another responsibility. We shall say that one role can 'instantiate' another. For example, in Develop a Software Product, the (single)

instance of the *Project Manager* role might instantiate the *Change Manager* role. This is what happens when a Project Manager creates the responsibility for managing change.

In the pharmaceutical R&D industry, when a chemical compound looks as though it has promising therapeutic properties, a team is often set up to champion that compound through the long process of getting it to market. That team has responsibility for managing the compound's life thereafter. So the *Compound Management Team* role is instantiated to carry out that responsibility. During the compound's development, many batches of raw drug material will be made for formulation for clinical trials. The (abstract) role of *Making a Batch* is instantiated for each batch – each time a batch is to be made there is a responsibility created for that; in fact there may be any number of instances of that role at any given time, depending on how many batches are in the process of being made at that time. (Remember that *Making a Batch* is not the name of an activity: it is the name of the responsibility for making a batch.)

Any role that is not permanent as far as the process is concerned will have to be instantiated at some point – we have called these 'transient' roles.

Note that creating a new role instance does not imply any allocation of real people or machines to act the role instance, as far as a RAD is concerned. The *actual* team of people who manage the life of a candidate compound may well change over the many years during which the compound is developed, but the role instance remains. When a batch of compound is to be made, the role instance is started and a process research chemist is allocated to actually make the batch, i.e. to act the role instance – it could be John or it could be Jill. Again we are being careful to separate a role instance from its actor. Also, if the new role instance needs resources (props) to carry out its responsibility, we might be interested in modelling how it acquires them. When I start work as the manager of a project, what props can I expect to find on my desk, where did they come from, and which props do I have to acquire for myself?

KEY POINTS

Some role types have pre-existing instances in a given process.

Other role types must be instantiated during the process: this equates to creating a new responsibility during a process.

Instantiating a role type only creates the responsibility; allocating actors to carry out that responsibility is a separate matter.

A role type has an *instance profile* in a given process:

☞ It can have one or more pre-existing instances.

- ☞ Those instances might be fixed for the duration of the process.
- ☞ The actor of a role instance might change during the life of the instance.

That 'in a given process' above is important: a role's instance profile will depend on the scope of the process context. As far as the process of taking a proposed law through Parliament is concerned, the role *Member of Parliament* can be considered to have 635 pre-existing instances whose actors are of no concern to us. But if we are concerned with the process of electing new Members of Parliament, then the actors and their allocation is very much a matter of concern. If we are concerned with the (unusual) process of redefining parliamentary boundaries and hence with creating and deleting instances of *Member of Parliament*, then we shall have a certain number of pre-existing instances, and potentially a different number of instances on completion of the process; actors will not be our concern here.

Later, next to the water-cooler

Pupil: Something puzzles me. You're saying – and I understand this – that responsibilities come and go in the organisation. To take the example earlier, when a new project comes along in Hill Pharm, that creates a new responsibility for managing the project. Once the project is finished that responsibility is gone. So there is a flux of responsibilities washing through the organisation – right?

Tutor: Right. In fact I like the word *flux* – a continuous succession of changes … that's what's happening: responsibilities are coming and going.

Pupil: But the organogram stays more or less constant. In particular, the organisational structure doesn't change with the flow of responsibilities. What's happening?

Tutor: Simple: every time a new responsibility arises it gets allocated to a post or a functional group, or one of the other concrete – shall we say 'real-life'? – roles. When a purchase order is raised, the responsibility for checking and approving it is created. That responsibility is an abstract role. I would call it *Purchase order approving*. As you say, as time passes there is a flux of such role instances, coming and going. Yet all invoices are actually handed to the CFO for checking and approval. So we are allocating all the *Purchase order approving* role instances onto the single instance of CFO. Try the idea with a different situation: suppose someone calls the Helpdesk.

Pupil: OK, a call comes into the Helpdesk. That creates a responsibility … to answer it satisfactorily, say. I guess you'd want to call that role *Call handling*. One

way or another the call is routed to a desk. I guess the desk represents a concrete role. But you would want me to be more accurate: the desk represents an *instance* of a concrete role – call it *Help Desk Station* – and the person sitting at that desk is acting that instance. So we have mapped that instance of *Call handling* onto that instance of *Help Desk Station*.

Tutor: Exactly! Now do the same with another situation: the fire service gets a report of a fire.

Pupil: Right, well, the report creates a responsibility – to put out the fire. The responsibility is taken on by the fire crew leader. So the responsibility which we might give the abstract role name *Fire handling* has been mapped onto the concrete role *Fire crew leader*. Presumably, when we model all these processes, we have a choice between modelling the abstract role – a transient role that comes and goes – or the concrete role that the responsibility is mapped onto, which is rather more permanent?

Tutor: Precisely. Let me try and capture this. We identified six types of thing we might find 'playing a part' in a process : unique post, unique functional group, etc. Five of them are organisational realities, the sixth is an abstraction: a 'pure responsibility'. In many situations we create responsibilities on the fly. We'll see later that such a responsibility typically 'goes with' what we shall call a 'unit of work', it's the responsibility for that unit of work. But that responsibility has to be allocated to something in the organisation: one of the five organisational types of role.

Pupil: So when a customer order arrives we might say that the responsibility to deal with it is created, and that we give that responsibility (initially at least) to, say, a Customer Order Clerk. On the RAD, we could choose to model the abstract role – the responsibility – *Customer Order Handling*, or the organisational entity: *Customer Order Clerk*.

Tutor: A good example. A 'post' is another example. The organisational post 'CEO' means nothing on its own. It is given meaning by the responsibilities that we allocate to it. The post CEO takes on many responsibilities in many processes. Some are permanent, many are transient. By acting as CEO, a person takes on that flux of responsibilities.

Pupil: One final point: organisations depend a great deal these days on their computer systems. They have systems … often massive systems … that – shall I say – play a part in the process. Would you consider such a system as a role?

Tutor: In a sense, yes, but I'm going to be a bit cautious about the idea. Later on we shall indeed see some process models that have such systems as roles, playing a part in the process. I'm cautious simply because it's hard to say that the system 'takes responsibility' for anything – it is just a heap of metal and plastic after all.

You're looking tired?

Pupil: You really have whittled away at this idea of a 'role'. Why should we get involved with all these subtleties?

Tutor: Have patience. We shall have some important questions to answer and these categories of role will help us to answer them:

Do we understand the dynamics of each role?

Do we need to model how instances arise? In particular, how do responsibilities get identified and created and by whom? Who has the authority to create a new responsibility in the organisation?

Do we need to model how actors become connected to the instances they are acting? The relationship between actor and role instance is what the scheduling of staff and resources is all about, and this itself might be part of the process or indeed another process altogether.

Pupil: OK. I can see how it would be all too easy when looking at processes to worry just about what is done, and to forget how things get organised to get done, and how responsibilities are created and handed around.

Tutor: Indeed, but there's more to it than that. A constant theme of *Riva* is that organisational activity is massively concurrent: there are many, many things going on at the same time. If we want to get our heads round it and get a true and full understanding of that organisational activity, one that captures that concurrency, then we'll need ways of describing it in the language we use. Role instances are one of the sorts of concurrency that we find. Role instances have their own lives: they operate concurrently. We'll see later in this chapter that they also operate independently except where they collaborate through interactions.

From now on, we must take care to differentiate between a *role type*, a *role instance*, and the *actor* of a role instance, unless the sense is clear from the context.

(If you are familiar with the notion of *use cases* in UML, you will need to make a concept switch from UML's use of *role* and *actor* to *Riva*'s. UML confusingly defines a

use case as '*The specification of* [my italics] a sequence of actions … that a system (or other entity) can perform, interacting with actors of the system,' an *actor* as 'A coherent set of roles that users of use cases play when interacting with these use cases. An actor has one role for each use case with which it communicates.' and a *role* as 'The named specific behavior of an entity participating in a particular context.' See www.omg.org/uml.)

ACTIONS

When we look at a single process, we know that we chunk everything that gets done in the process into roles, and that each role represents an area of responsibility within the process. We must now look inside each role.

Actions are what actors do on their own in their roles to carry out their responsibilities. Let's take some examples:

- ☞ In the process of developing a new pharmaceutical drug, actions might include *Choose lead molecule*, *Carry out a clinical trial*, *Prepare the submission to Regulatory Authority*, and *Clean the pilot plant*.
- ☞ In the process of buying a house we might find actions such as *Choose estate agents (realtors)*, *Obtain finance*, *Obtain planning permission*, and *Negotiate the price*.
- ☞ In the process of developing a software system, actions might include *Prepare the project plan*, *Prepare use case model*, *Carry out a proof obligation*, *Transform an algorithm*, *Verify a code module against specification*, *Build the system*, and *Add a component to the object library*.

Note how we name actions with verbs: 'prepare', 'draw up', 'verify', etc.

An action needs to be well defined, in particular we need to know what makes it start and what makes it stop, what state the world is in when it starts and what state the world is in when it stops, i.e. when and why an action is done. Let's look at this in more detail.

Like roles, actions are defined in *Riva* as types that have instances. If we are speaking informally about a process we shall say 'Action *A* starts'; if we want to be formal and precise we shall say 'An instance of action type *A* is created.' An instance of an action type is created when the organisation's process enters a particular state: we call this the *activating condition* for the action. This is a sufficient condition for the action instance to start. For example, in an organisation that has a policy of paying invoices three months after receiving them, the action *Pay invoiced amount* would have as its activating condition the fact that payment of an invoice has been due for three months. In an organisation that pays on receipt, the activating condition would

be the fact that an invoice has been received. The *post-condition* of an action is the *state* of the world when the action (instance) has finished. *Pay invoiced amount* would probably have as (part of) its post-condition the fact that a cheque for the invoiced amount had been sent to the supplier.

Not surprisingly, the post-condition of one action will often be the activating condition for another. So, in the Purchase Materials process in our prompt-paying organisation, the post-condition of the action *Receive invoice* – probably *invoice received* – will be the activating condition of the action *Pay invoiced amount*. In this case we might say that 'the action *Pay invoiced amount* follows the action *Receive invoice*' or that 'the action *Pay invoiced amount* is consequent on the action *Receive invoice*'.

There may be other conditions that are also true when an action is started and which we want to note (though they are not what makes the action start); these are collectively referred to as the action's *pre-condition*. They are necessary but not sufficient conditions for the action to start. Thus, in the late-paying organisation, the action *Pay invoiced amount* might have as a pre-condition the fact that payment has been authorised. Authorisation does not activate payment: 'due for three months' does. Payment cannot occur unless there has been authorisation.

(Strictly, a post-condition is a necessary but not sufficient condition for completion of the action. We can also define a *stopping condition* of an action, which is the sufficient condition for the action instance to stop.)

As an example in Develop a Software System, suppose we have an action (type) called *Compile component source code*. The activating condition for this could be *component source code successfully syntax-checked*; a pre-condition could be *access available to validated library*; the post-condition could be *corresponding object file available in the object directory*.

Finally, let's note that an action can have alternative activating conditions: there can be a number of different situations that cause me to send an email, or write out a cheque, or compile some source code. Each has a different activating condition, but they will all share the same pre-conditions.

Actions change the state of things

Pupil: I've noticed that you've made no attempt to define an action in terms of 'inputs' and 'outputs'. I've seen actions described as ways of 'transforming inputs into outputs'. Why don't we do that in Riva?

Tutor: Because that manufacturing-oriented way of thinking about actions and processes isn't helpful. We end up distorting the idea of an input and an output – and hence our understanding of an action – just to stick with the metaphor. The compiled object code could certainly be thought of as an 'output' of

the action *Compile component source code*, a product of the action. But it's nonsense to say that the source code was 'transformed' into the object code. It clearly wasn't. The source code of a software component isn't 'consumed' by the action – it still exists after the compilation – so calling it an input is strange. A dirty car is not 'consumed' by the act of washing it.

Pupil: No, I guess not. The purpose of washing a car is to change its state from 'dirty' to 'clean'. Ah … I described it in terms of states. But couldn't we say that a dirty car was the input and a clean car was the output?

Tutor: We could, and one way I could implement that is to crush and dispose of the dirty car you give me and hand you back a clean car. I didn't *transform* the car you gave me but I satisfied the input–output specification. What sense does it make to say that a purchase order is 'transformed' into goods? The whole input–transform–output metaphor leads us into absurd statements and is best avoided.

It is much more natural and less forced to think of the *state* of the world before an action starts – there is a particular software component that is uncompiled – and its *state* after the action has finished – there is an object code file associated with the component source code in the development library. The purpose of the action is to change the state of (that part of) the world. This all becomes much more obvious when we are dealing with desired outcomes such as 'a happy customer'. I would rather say that an action leaves a customer in a happy state, rather than that it outputs a happy customer as though they have popped out of the side of the box smiling having gone in the other side looking glum. And I would certainly not want to say that an action 'transforms a purchase order into a happy customer'. That sounds like science fiction.

Pupil: One example that comes to mind is the process for curing someone of an illness. The input – forgive me – is a sick patient, and the output is a well patient. What's wrong with that?

Tutor: Remember my constant message: when we think about processes, we are thinking about *dynamics*. When we think about actions, we want to know *when* things happen, *when* actions start. States do that for us. When we say '*C* is the activating condition for action *A*' we are defining the dynamics of that part of the process: *C* is what makes *A* start. Defining the inputs of *A* (whatever it means) doesn't tell us what makes *A* start. Inputs and outputs might capture data dynamics, but they don't capture process dynamics. States do.

> If you tell me that the action *Cure a sick person* has a sick person as its input and a well person as its output, I'm no wiser about what starts the activity, which seems to me to be vital information. I need to know that its activating state is *Sick person waiting in reception*, and that its post-condition is *Well person has returned home*, or possibly – let's be realistic – *Dead person is in mortuary*. (Of course, if we do allow that other possible post-condition, we have named the action badly.)
>
> Finally, we shall see in a moment the importance of *goals* of a process. Goals are, of course, just desired states, so we shall need to have the language in place for talking about states.

Actions have relationships

Actions not only have important properties of their own, they also relate to each other in different ways. There are three ways:

- Action *A* might always follow action *B* in role *R*: a cheque cannot be sent until the expense claim has been approved. So actions may be *ordered* and follow a particular sequence. The action *Approve expense claim* must precede the action *Send cheque for expenses* within the role *Finance*.
- Either action *A* is carried out or action *B* is carried out in role *R* depending on whether some condition *C* holds: if the expense claim is over £1,000 it is paid by electronic transfer, otherwise by cheque. So actions may be *conditional*. The action *Pay expense claim by electronic transfer* can only start (be instantiated) if the condition *expense claim exceeds £1,000* is true; the action *Pay expense claim by cheque* can only start (be instantiated) if it is false.
- At some point both action *A* and action *B* can proceed in parallel in role *R*: once the expense claim has been approved, the money can be paid to the claimant and the relevant department budget can be debited. So actions may be *concurrent*. The action *Pay expenses to claimant* can proceed concurrently with *Debit departmental budget* within the role *Finance*.

In our process model we will want to be able to show where actions are sequential, conditional or concurrent. These will be one way in which business rules will be represented.

KEY POINTS

An action is carried out by a role on its own.

The activating condition of an action is a state that causes an action to start (be instantiated).

The post-condition of an action is the state of the world when the action has finished.

We define the dynamics of roles in terms of state changes.

INTERACTIONS

Roles 'chunk' the activity in a process. Roles carry out actions on their own account. But we started from the important axiom that the main way things happen in a process, especially in terms of 'moving the process on' or 'making progress', is through the *interactions* that take place between roles, such as when a manager delegates a task to a subordinate, or a price is negotiated.

In the process of Develop a Software System for a Client, the role *Project Manager* will want to interact with the roles *Designer* and *Programmer* to obtain status reports on work completed, and the *Designer* role will want to pass specifications of programs to be written to the *Programmer* role. In the process Develop a Portfolio of Products, the *Board of Directors* will want to pass a statement of direction to the *Product Strategy Board* along with a budget level and targets. In return the *Product Strategy Board* will present the *Board of Directors* with information on the chosen portfolio, and progress reports against budget and targets.

In *Riva* terms, an interaction is neutral and has no implied direction – it is just some *coordination* between roles, a collaborative act. But an interaction might involve the transfer of something – what we shall call a *gram* – from the body of one role to that of the other. For example, the *Divisional Manager* role interacts with the *Project Manager* role so that the former can pass the latter some terms of reference for the project they are to manage. But an interaction need not involve the transfer of a gram: for instance, you and I might interact simply to agree on something – 'nothing changes hands'. For instance, the Sales Team, the Marketing Team, and the Production Group of a product company might collectively decide when a new product should go to market: that interaction might consist of a discussion around a table.

As with actions, we will think of interactions in terms of states and state changes, not in terms of inputs and outputs. When I reimburse you with your expenses, I see it in terms of your change of state: before the interaction you didn't have your expenses,

after the interaction you did; before the interaction I had the money, afterwards you had it.

An interaction can be two-party – involving two role instances – or multi-party, involving a number of role instances. However many parties are involved, an interaction is always *synchronous*:

- ☞ it starts at the same moment for each party, as soon as they are all ready;
- ☞ it completes at the same moment for each party, as soon as they have all finished.

One way of thinking about an interaction is as *an alignment of states*. For you and me to interact, we must both be 'waiting' in the required state beforehand, we go through the interaction together, and then we go to our respective after-states.

In some cases an interaction might physically take a few seconds (I give you a memo containing some terms of reference), in others months (a vendor and purchaser agree on the contractual terms of a sale). As elsewhere in our modelling notation, we do not capture absolute time: there is no time axis on our models. We might choose to annotate actions and interactions with their duration in some way, but no more.

KEY POINTS

Interactions between roles are the way that collaboration happens in a process.

Interactions are the way that role instances coordinate their activity.

In some interactions grams change hands.

Interactions align the states of the interacting parties.

PROCESS GOALS

Point-wise goals

Processes are there for a reason. For instance, the goal of a process might be to deliver a computer system, to provide a medical procedure to a patient, or to manage a research budget. It must be possible to see from our process models, how a process is achieving the goals set for it, and ideally to be able to identify the point(s) in the process where those goals can be said to have been achieved or maintained.

In the simplest case we will be able to identify some point in the activity of a particular role where the state of the process is 'goal achieved'. After a particular action or interaction has completed we can recognise that the goal has been achieved. For

instance, in a process for handling a reported credit card loss, we might say that, once there has been an interaction with the customer in which the latter has been sent a new credit card, the goal 'client has been sent replacement credit card' has been achieved. We can identify the role in which, at some point, that state has been achieved, i.e. the state at that point *is* the goal. Reaching that state is reaching the goal.

The goal of an insurance company in its New Policy Applications Department might be to respond to a customer with a proposal within seven days of receipt of their application. Each step in the process can contribute to the successful achievement of that 'point-wise' goal: there comes a point when we can say 'Here the goal of the process has been achieved.' The goal of the process of carrying out a software development project for a client is to satisfy the client with a timely delivery of working software. The only point at which we can check whether we have been successful is the point of delivery: is it on time and does it work? Until that point we can only make predictions.

In some situations there may be several goals, leading not to one state in one role but a combination of states in a number of roles. For instance, the goals of the Handle a Reported Credit Card Loss process might be not only that a new card is sent to the customer, but also that the credit card fraud bureau is informed, and that the card number has been entered on the list of lost cards that is circulated to retailers.

The New Policy Applications Department in a life insurance company wants to get a correct policy proposal out within seven days of receipt of the application; it might also want to ensure that the business it takes on is good business, i.e. that the premiums it charges adequately cover the risk; and it might also have the goal of making the terms offered available to those assessing the competitiveness of the company's products.

On the way towards achieving a 'final' goal, we can often identify 'sub-goals' which represent milestones of some sort. Issuing the insurance proposal to the customer requires that, at some point en route, the risk be satisfactorily assessed and that approval be obtained for the premium offered.

This indicates, I hope, how important it is to see a process model as a description of *the way that the organisation changes state*, from its initial state (someone wanting a service or whatever) to its final state (service delivered).

Steady-state goals

When I run my life in financial terms, one of my aims is to regulate my earning and spending so as to keep my bank balance in credit. At any moment in time I prefer to have the bank looking after my money rather than the other way round. I don't like paying interest. The critical bit is 'at any moment in time'. I want to see a steady state in which I am in credit.

The steady-state goal is more complicated. By definition it says that something is true at all times, or – softening this a little – that it is true at a number of points in the process (the times when we choose to look, say). I might want to be sure that at all times everyone has the latest information on product features, or that expenditure is always kept within the budget level set at the start of the year. The process of managing cash flow in a company has the same goal: that of maintaining a steady state in which the company keeps a positive cash flow as expenditure and revenue rise and fall. If the process is successful we should be able to observe the company at any moment in time and note a positive cash flow. We shall unpick this a little more later.

KEY POINTS

Processes have goals.

Goals are simply desired states.

Goals can be 'point-wise' or 'steady-state'.

ENTITIES

Pupil: Why isn't *data* or *information* in our list of concepts?

Tutor: The simple answer is that in this approach to business processes we concentrate unashamedly on what people *do*, rather than on what people do it to or what they do it with. Once we have chosen a structure for the organisation and the processes it will operate, then we can decide on the information needed by individuals and groups to perform those processes in that organisational structure.

Process precedes information.

Once we have a process model – a description of how the business does its business or plans to do its business – we can start to investigate the information needs of the process: Who needs what information to do that, or to make that decision? And how does that information get to that person? We might think of documents, in particular, as the oil that makes the wheels turn, but they aren't the wheels!

Pupil: You've made a lot of disapproving noises about 'inputs' and 'outputs'. And now you're keeping low on information. Do you deny the existence of *things* altogether? For instance, a manager will prepare a *Plan*, write a *Report*, draw up *Terms of Reference*, or approve a *Document*. *Plan*, *Report*, *Terms of Reference*, and *Document* are entities. A programmer uses a *Specification* during

the programming action and produces a *Program*. A warehouse person uses a *Picking List* during the packing action.

Tutor: Yes, we shall indeed talk about 'entities'. And we'll use the word 'entity' for anything that is the subject matter of an action. Or indeed an interaction: when I delegate a task to you I perhaps pass you some *Terms of Reference*; and when you have finished the work you will pass me the *Results*. The grams of an interaction are frequently entities.

Like roles and actions, *Riva* entities come as *types* which can be instantiated. In fact if an action instance produces an output we consider it to be *instantiating* the relevant entity type.

When we define an entity, we will want to ascribe properties to it. Amongst those that are covered by *Riva* are the parts from which a compound entity is composed if it is a composite thing, and 'invariants', i.e. things that are always true about the entity. For example, a *Technical Specification* might be defined to be made up of a *Contents List*, *Document Control Section*, a *Scope Section*, a *Control Flow Section*, and a *Data Flow Section*, followed by *Performance Details*. A *Production Plan* might be made up of a *List of Input Resources*, a *Timetable*, and a *Definition* of the process to be used. An invariant of the entity *Production Lathe* is that it is *up to date in its maintenance schedule*.

Pupil: And entities do play a part, don't they? If I ring up someone who is doing something for me, and ask them how things are getting on, they'll often tell me about the state of the objects that are involved: 'Your application forms have been approved,' 'Your car will be ready in half an hour,' 'The catalogue you ordered is on its way to you,' and so on.

Tutor: That's right. And more generally, when we define the activating condition, pre-conditions, or post-condition of an action we will often do it in terms of the states of entities. For example

The activating condition of the action *Send Invoice* is that there is an Invoice and it has been approved.

A pre-condition of the action *Send Invoice* is that the Goods have been received in good condition.

A post-condition of the action *Send Invoice* is that the Database has been updated with the date the invoice was sent, that the invoice has been sent, and so on.

Pupil: So we will have entities, but information – presumably about those entities – is simply their state? And we shall talk about state? and we shan't concentrate on information per se?

Tutor: Exactly. If our task is to understand a process then worrying about the information of the process is a bit like trying to understand a document by worrying about the typeface that a document is written in.

KEY POINTS

Entities are the subject matter of actions and interactions.

Entities are instantiated, just like any other types.

Entities have states, some of which are used in defining actions and interactions.

Information consists of state descriptions.

THINGS ARE COMPLICATED

Pupil: You've outlined the ideas behind each of the central concepts – roles, actions, interactions, and so on – but I sense that there is a more general message that you're getting at. This notion of 'instance' seems important.

Tutor: Yes, in fact there are two important 'inner' messages here. The first is the central place of 'instantiation': the making of new instances. It's important because it's what drives the *dynamics* of a process. We've seen how roles in particular are instantiated: new responsibilities are created dynamically during a process. We'll look more closely later at the instantiation of actions and interactions.

Pupil: OK, so an organisation at any one moment is just a mass of instances, and an organisation viewed over time is a flux of instances. What's the second inner message?

Tutor: It's a closely related message: processes are about *concurrency*: lots of things are happening at the same time. When we walk into the building and watch the mass of organisational activity, we don't see a simple flowchart being followed, with someone's finger tracing down the boxes. We see many instances of many different processes in progress simultaneously. Within each process instance we see many role instances active at the same time. We see role instances coming and going as responsibilities arise and cease. And within

each role instance we see potentially many threads of activity in progress. In summary, there is a massive network of inter-related threads operating.

Pupil: So, all those instances mean concurrency. And presumably what an organisation achieves it achieves because of all that concurrency – things don't happen because of sequential activity.

Tutor: Exactly. Let me give you an analogy which my colleague Clive Roberts embodies in the logo of his company Co-ordination Systems. When geese fly long distances, each bird uses the same tactic: fly in a specific relationship to the next bird. The result is a chevron of birds. That chevron is an *emergent behaviour* resulting from the interactions and concurrent activity of all the birds. Yet no bird looks at the overall shape and decides how to respond. It follows its own tactic. The effect of the organisation is an emergent property resulting from the interactions and concurrent activity of many role instances across many process instances.

To truly understand our processes as dynamic objects we must grapple with instantiation and with concurrency and in particular find ways to model them. And, if we want to design sets of processes that we can actually execute on a Business Process Management System of the sort we shall explore in Chapter 13, we shall need to be able to design those dynamics that give the right emergent behaviour.

Instantiation and concurrency are central themes of *Riva*, and when we have developed the argument we shall turn to the idea of *mobility* of processes.