

RoEnact: role-based enactable models of business processes

Keith Thomas Phalp^{a,*}, Peter Henderson^b, Robert John Walters^b, Geetha Abeysinghe^c

^a*Empirical Software Engineering Research Group, Department of Computing, Bournemouth University, Talbot Campus, Fern Barrow, Poole, Dorset BH12 5BB, UK*

^b*Declarative Systems and Software Engineering, Department of Electronics and Computer Science, University of Southampton, Highfield, Southampton SO17 1BJ, UK*

^c*School of Computer Science, Middlesex University, The Burroughs, London NW4 4BT, UK*

Received 19 February 1997; received in revised form 27 February 1998; accepted 2 March 1998

Abstract

This paper describes RoEnact: a process-modelling notation used to provide enactable models of process instances. The paper shows how RoEnact models may be produced which are equivalent to role activity diagrams (RADs). This allows the modeller to describe processes in a notation (RADs); which can be understood both by process consultants and process users, whilst retaining the ability to generate enactable process scenarios. © 1998 Elsevier Science B.V.

Keywords: Business process modelling; Role-based models; Role activity diagram; Enaction; Condition–action models

1. Introduction

The focus of this paper is the use of notations and tools to improve business processes. However, in describing approaches to business process modelling it is difficult to ignore the impact which software process modelling and the study of software process has had upon the discipline [1]. Many of the notations and tools used for business process modelling were originally developed for study of the software process [2]. The drive towards automation of the software process and towards project support environments has produced much software process technology. This process technology shares many features with modelling technology in other business domains [3–5].

This paper takes its rationale from another area of software process modelling, process programming [6]. Process programming argues that the software development process may be regarded as a set of activities, with associated inputs and outputs, that can be described in the same way that a software program describes the data and control to be captured in the final system [7]. Taken literally, a consequence of this view is that process models should use modelling languages like programming languages [8,9], and should attempt to codify and control human behaviour. Hence, some authors have argued vehemently against process programming [10], arguing that systems which involve

human behaviour cannot be codified or controlled so rigidly [11].

However, one of the intentions of the adoption of the phrase process programming was to provide an analogy about the way a software process should be developed [12]. The argument being that a development process should have a process life cycle [13]. Osterweil [6] states that ‘the various software processes should be viewed as having been created by process development processes’. This suggests that there needs to be a life cycle for developing processes, which includes phases for process requirements, process design, process construction, process testing, process evolution and process re-use [12].

These ideas may also be applied to the wider business process domain. Hence, business process models should have a requirements capture phase, a design phase, a debugging phase and so on. However, in applying such ideas to business processes the paper again draws on another idea from the software domain, that of executable specifications [14].

1.1. Executable specifications of business processes

Proponents of executable specifications argue that one of the main advantages of this technique is that it allows specification errors to be spotted far earlier. Since the cost of fixing problems late in the development process may be as much as 100 times greater than if they had been detected

* Corresponding author.

early, e.g. in specification [15], the early discovery of such problems would appear to be beneficial. However, despite such economic arguments there is still much debate about the utility of executable specifications [16].

This paper proposes that, as with the software development process, the early detection of specification errors will be equally cost-effective when the object being specified is a business process. Furthermore, it is suggested that the validation of business processes at the specification and analysis phase may be aided by the use of executable (enactable) models. Hence, this paper describes RolEnact, a notation for producing executable specifications of business processes. The paper also describes how this language may be combined with Role Activity Diagrams (RADs) [17], which are used for the initial gathering of process requirements.

Briefly, the method utilised is as follows: requirements capture and validation is facilitated by the use of RADs. These diagrams are translated to RolEnact code and RolEnact models are then run on a computer. These models have a simple Windows interface which allows modellers (or users) to experiment with the process behaviour (to run the executable specification of the business process). Hence, the model of the business process may be debugged before its implementation, and process specification errors captured far earlier.

2. Notations

Process modelling is an area that has seen a great deal of work over the last decade or more. A great many paradigms have been proposed to model processes in a variety of domains [18], and there exist many tools and notations for those wishing to attempt to model their business processes [19].

Curtis [20] classifies process modelling approaches as taking one of four perspectives; informational, organisational, procedural and behavioural. Of these approaches, the latter two, procedural and behavioural, account for the majority of process modelling currently being undertaken. Information approaches are too static, failing to capture the process or its dynamics, and organisational views are in many ways the antithesis of the process modelling movement, restricting the most efficient use of resources.

The following section argues that of these two remaining approaches, role-based models are more appropriate to the needs of many business process modellers.

2.1. An argument for role-based models

Procedural views of a process typically involve the production of data flow based models, based on notations such as Yourdon (see Ref. [21], used by Tate [22]), IDEF0 (see Ref. [23]) or ProcessWise WorkBench (see Ref. [24]). These methods describe processes in terms of activities

and the data or objects communicated between activities. Though using tried and tested techniques, it is very difficult with such models to abstract away from the details of process, and to capture the interactions between the people who carry out core activities¹.

When attempting to redesign or improve a process, the modeller should not be concerned with the mechanism of how this process proceeds. What is needed is a model that describes those activities that support the business goal. The activity of receiving and passing on information may really be superfluous to the process, or it may be core, but the mechanism by which this happens is irrelevant. It is argued [19,25] that when modelling at this level of detail it is harder to move away from the current mechanism of the process in attempting to redesign, and easier to assume that these mechanisms are actually essential aspects of the process. Furthermore, the necessity for activities to communicate via business objects introduces many artificial objects into the process description. Indeed, the depiction of the business process should not prescribe mechanisms since it is then more likely to inhibit change. Consider a manager communicating some project detail to a team member. From the business perspective it is important is that an interaction takes place and that further activities may then proceed. The business goal must be represented, not the mechanism that currently supports it.

A second problem with procedural models is that the activities that are to be carried out by individuals are often spread around the model, since the models tend to have decomposition related to function. For an individual (or group) in the organisation to carry out their activities, they need to know what activities they must take part in, in what order those activities must take place, and what other individuals or groups they must interact with [26].

Role-based models satisfy these requirements by grouping activities into 'roles', which describe the desired behaviour of individual groups, or systems [25]. 'A role involves a set of activities which, taken together, carry out a particular responsibility or set of responsibilities' [27]. Roles are like types or classes in that they describe behaviour that is then carried out by some actor (person) or agent. Customer behaviour may be described by a customer role, but a particular customer is an instance of that role.

2.2. Role activity diagrams

The example in this paper is described using a RAD. This diagram (Fig. 1) describes the example process. The reader may find it useful to refer to Fig. 1 whilst reading the

¹ For example, a particular person or group of people may be responsible for a number of activities within the process. A typical procedural approach would show how a particular activity may receive documents, take some action, and then pass them on to a further activity. There are two kinds of problem with this approach. Firstly, that it focuses on the mechanism of the current process and, secondly, that it does not describe the process with respect to those who will have to enact it.

following section which briefly describes RAD concepts and notation.

2.2.1. Roles

Roles group together activities that can be carried out by a group, an individual or a system (i.e. some actor or agent). The grouping of activities into a role reflects the fact that it represents some unit of responsibility. Roles are depicted as rounded rectangles surrounding activities.

Roles have a thread of control depicted by a vertical state line. The thread of control for the role allows for the description of sequential activities, parallel activities, and choice. Roles are types, e.g. they describe the behaviour of a class of individuals. Hence, there may be many instances of a particular role when the process is enacted; for example, there may be many customers. In addition, a single person may act out multiple roles; for example, a cashier may also act as a supervisor. A role is independent of other roles, but communicates through interactions. Instances of roles therefore act in parallel, with the interaction between roles being their only synchronisation mechanism.

2.2.2. State

A role has state. In carrying out an activity, it moves from

state to state. However, the notation does not require the modeller to explicitly label the state of a role, though some authors prefer to do so [19,28]. Therefore, the state may be viewed as a point on the vertical line that depicts the thread of control of the role. Consider the activity ‘complete application’ in the ‘Client’ role of the example. In carrying out the activity, the role moves from a state of ‘having received’ the form to being ‘ready to submit’ the form. The line vertically above the activity represents its current state, and the line below its new state. By labelling states the semantics of the role become clearer, and the labels help to make explicit the pre-conditions and consequences of each activity. However, the diagram becomes larger and this sometimes hampers understanding.

2.2.3. Activities

There are two types of activity in a role. An action is an activity that the role carries out in isolation. Carrying out an action moves the role from its present state to the next state. An action is represented by a small square. In this paper, actions are represented as solid black squares. An interaction is an activity that is carried out in sequence with another activity (or other activities) in another role (or roles). The consequence of an interaction is that all of the roles involved

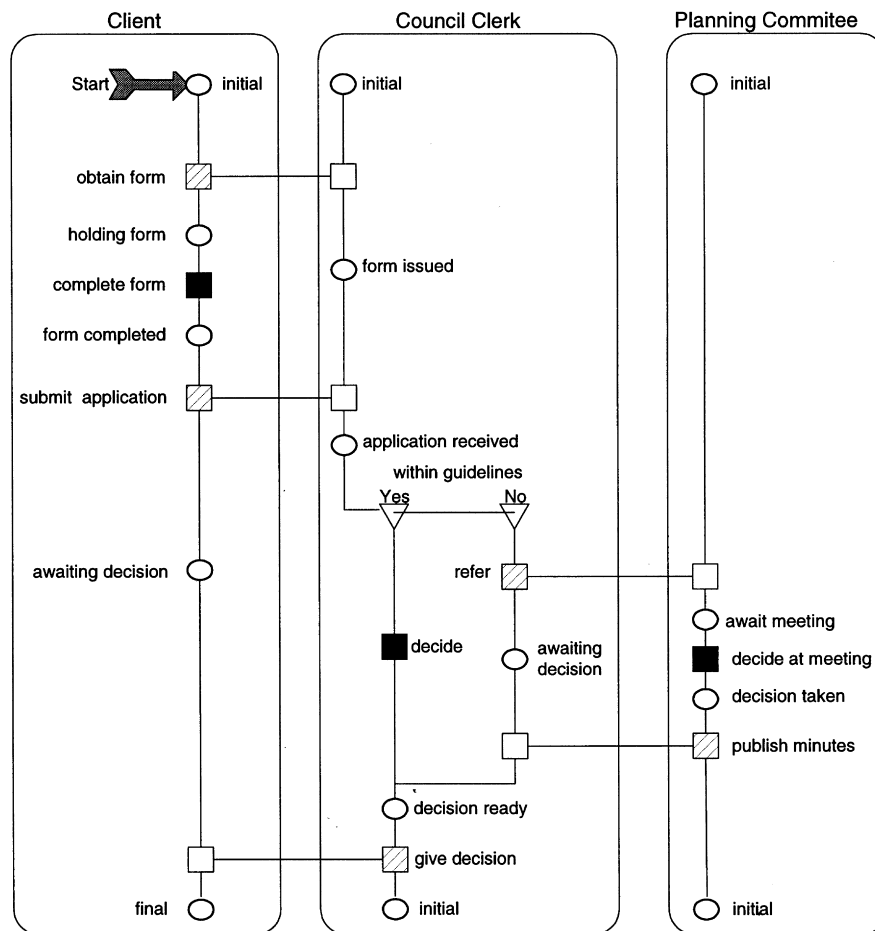


Fig. 1. Role activity diagram for an example.

move to their next state. Interactions are shown as small clear squares joined with horizontal lines. An interaction is always driven by some role, and in this paper this is signified by the square of that interaction being hatched or shaded.

2.2.4. Control

The thread of control in a role need not proceed sequentially. RADs have constructs to represent alternate paths (choice) and concurrent paths (parallel).

Choice is termed ‘case refinement’ [27,29], and is shown by the state line being split into two paths, the top of each path being marked with a downward pointing triangle (or in some tools, e.g. RADitor [29], a circle). There may be any number of alternative threads but only one of the threads (or cases) may be chosen.

Concurrent threads are termed ‘part refinement’ [27,29], each thread representing part of the path. The threads all join again after the split denoting that all paths have been completed. The points where the path divides are marked with an upward pointing triangle.

2.2.5. Iteration

Drawing a loop back to a previous point on the role normally shows iteration in roles. This signifies that the named state may be revisited. Typically, this looping is used when there is some checking or control mechanism within the business process.

2.3. RolEnact

RolEnact is a language for process modelling. It is based upon a condition–action paradigm. However, its primitives match those of role-based models (as described above). Thus, processes are described in terms of roles, the states of these roles, and the activities or events in which each role may take part. An instance of a role has state, and may move to its next state through an activity. This activity may be in isolation (an action) or may involve changing the state of another role or roles (an interaction or a selection).

However, it is not the fact that RolEnact brings together condition–action and role-based paradigms which is its main advantage, rather it is the fact that RolEnact models may be executed on a computer providing a simple Windows-based interface which users may use to experiment with processes. These enactable models are used by two main classes of users.

- Modellers, who produce and experiment with the models in order to understand the process description, to discover problems, and to analyse alternatives.
- Representatives of the client organisation, who interact with the models by taking the parts of users. These sessions can be used to validate the models, to experiment with process scenarios, and to provide a vehicle for process discussion.

3. Example problem and RADs

The example used in this paper describes the interactions between three roles: a member of the public wishing to make a planning application (the Client), a Council Clerk and the Council Planning Committee. This is a simplified version of processes encountered by the authors in modelling local government organisations, both in Europe and the UK [30].

3.1. An example as a role activity diagram

Fig. 1 shows the three roles ‘Client’, ‘Council Clerk’, and ‘Planning Committee’. Starting at the top of the member of the public, the ‘Client’ role there is an external event (shown as a horizontal arrow) which signifies the ‘Client’ deciding that planning approval is needed. The ‘Client’ obtains an application form from the Council offices and takes it away for completion. Once the form has been completed, the ‘Client’ returns it to the clerk at the Council Offices. The clerk examines the form to see if it can be approved without consulting the planning committee. If the application meets these criteria, the clerk can answer the application immediately. Otherwise, it is referred to the committee for a decision. When the clerk receives the response from the committee, that decision is passed back to the ‘Client’.

4. Moving towards enactment

RADs describe types, and thus they do not describe the synchronisation of instances of the roles. In order to move towards being able to run process simulations, some assumptions need to be made about the states of instances of roles. For example, this paper will assume that all roles start in an *initial* state and that, whilst the ‘Client’ role ends after a decision is received for an application, the ‘Clerk’ and ‘Committee’ roles return to the *initial* state ready to process another application. A working model of this example will also use an additional, fictitious role that will create the other roles.

RolEnact does not have explicit support for parallel threads within roles, but all roles act in parallel so that parallel threads within a role may be represented by separate roles. A new role (or roles) is created at the start of the parallel threaded section of the role to carry out each of the additional paths. These parallel paths rejoin by the use of an interaction. A consequence of this is that when RolEnact models are created from RADs they may have a greater number of roles. The advantage of this representation scheme (aside from providing a consistent mapping) is that it further decomposes the process such that for any instance of the previously parallel role, the parallel threads could now be assigned to different actors. However, a disadvantage is that the role with parallel threads may often be

a more representative depiction of the business process being modelled.

To aid comprehension, the features of the RolEnact language have been deliberately kept to a minimum. In some circumstances, this may mean that some ingenuity from the modeller is required to accurately describe the process under consideration. For example, although it is possible to create multiple instances of any role, a single role cannot simultaneously interact with multiple instances of another type. In this case, however, each instance would exhibit the same behaviour, moving from the same before state, to the same after state. Aside from the redundancy of such a description, this is not an intuitive way to think about business processes.

Hence, where a group of individuals act in this way, they are modelled as a single role. Consider the role Planning Committee role in the example process. Although formed of individuals the Committee act as one, and there is no requirement to understand their internal operation. The members of this Committee are best considered as a team acting in concert, i.e. as a single role. Each instance of this role is acted out by a number of staff (resource units), but these are not individual roles. Therefore, the model has a single role for the ‘Committee’ in place of multiple roles of a type ‘Committee_Member’.

5. RolEnact description

All RolEnact models can be made up of four basic types of behaviour: action, interaction, selection, and creation. These behaviours allow instances of roles to move from existing states into new states, to communicate with each other, to choose and then interact with other role instances, and to create new role instances.

The RolEnact behaviours correspond to those of standard RADs. RolEnact actions correspond to RAD actions, and move instances of roles from an existing state to their next state. RolEnact interactions also correspond to RAD interactions, in that they move all roles involved through from their existing states to their next states. RolEnact selections also correspond to RAD interactions. Selections are necessary in order for instances of roles to communicate consistently. Selections are interactions between roles that have not previously communicated. The distinction is that in order to ensure that the correct

instances communicate, selection sets up an identifier, so that each role instance can identify the other. Creation is another standard RAD construct and is reflected in RolEnact’s creation operation. This operation allows roles to create other role instances and to set up identifiers for future communication.

These four building blocks enable the modeller to build up the behaviour of standard RADs, but with the added advantage that the resulting model may be enacted on a computer.

5.1. Action

An action is as a process step that changes the state of the system. An *action* changes the state of its own role, from some *before* state to some *after* state (see Fig. 2).

The before state is a precondition for the action. That is, the action cannot take place if the role is not in the *before* state. As a result of the action, the role will be in the *after* state. The RolEnact syntax is shown below (keywords in bold):

```
Action Role.Action
      Me(before → after)
End
```

The Action keyword denotes that an action is to be described. The action name is prefixed with the name of the role that contains it, and then the name of the action itself follows the dot operator. The keyword *Me* refers to the current role instance which is invoking the action.

As an example, take the ‘complete form’ action of the ‘Client’ role. This is described as:

```
Action Client.complete_form
      Me(holding_form → form_completed)
End
```

5.2. Interaction

An interaction is a process step that takes place simultaneously in more than one role. As with RADs, the interaction has a driving role (which initiates the interaction), and all roles involved in the interaction move from their *before*

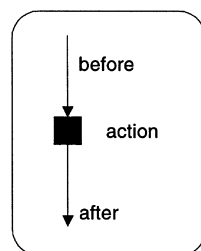


Fig. 2. A RolEnact action as a RAD.

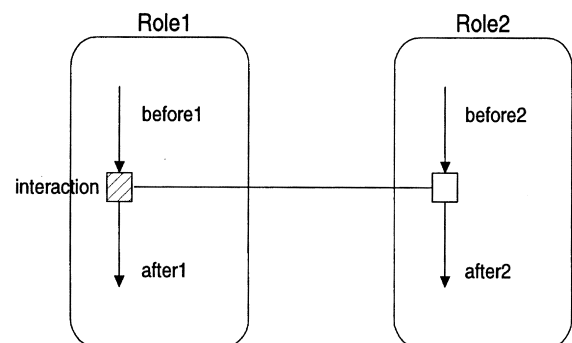


Fig. 3. A RolEnact interaction as a RAD.

state to their *after* state (see Fig. 3). For the interaction to take place, the driving role must have an associated role. This association will have been made in a previous selection or creation.

More formally, when the driving role (Role1) is in the state *before1* and its associated role (Role2) is in the state *before2*, the interaction may take place. The diagram above shows how this is depicted graphically (as a RAD). The interaction is named at the driving role end, which is depicted by having a shaded activity square. As a result of the interaction, the driving role will move to the state *after1*, and the associated role will move to the state *after2*. In RolEnact this is represented as:

```

Interaction Role1.Interaction
    Me(before1 → after1)
    Role2(before2 → after2)
End
    
```

The change of state for the roles is described as it would be for an action, i.e. before → after. Hence, in the above RolEnact description, the *Me* refers to the driving role, which changes state from *before1* to *after1*, and the *Role2* to the associated role, which changes state from *before2* to *after2*.

As an example consider the situation when a ‘Client’ has completed the form (and so is in the state ‘form completed’). The ‘Client’ may interact with a (previously associated) ‘Clerk’, only when the clerk is in the correct state (in this case ‘form issued’).

```

Interaction Client.submit_application
    Me(form_completed →
    awaiting_decision)
    Clerk(form_issued → form_received)
End
    
```

5.3. Selection

For an interaction to occur, the driving role must already have some associated role. Selection is one of the mechanisms by which association is made.

If one of the roles has created the other then there will already be an association from this creation, and the roles may simply communicate via an interaction. However, often a role needs to interact with another where there is

no existing association. In this case, the first interaction must be a selection.

In selection, the driving role selects another role, creates an association with that role and then behaves as in an interaction.

More formally when the driving role is in the state *before1* and there exists a role instance of type Role2 in the state *before2*, then the driving role performs the selection and moves to the state *after1* (see Fig. 4). The newly associated role is moved to the state of *after2*, and Role1 creates an association with Role2 and vice-versa.

The RolEnact for a selection is:

```

Selection Role1.Selection
    Me(before1 → after1)
    Role2(before2 → after2)
End
    
```

Note that this appears to be the same as an interaction, and indeed a RAD would make no distinction between a selection and an interaction. However, the *Role2*, in the above RolEnact description is a role type and not an instance. The selection states that a role instance of type *Role2* will be chosen. *Role1* will be associated with the chosen instance of type *Role2*, and then an interaction will take place. Although hidden from the user or modeller, RolEnact is generating the following association:

```

Me.Role2 := r      r.Role1 := Me
    
```

That is, the driving role identifies the chosen role *r*, as the role of type *Role2* to which it is associated, and the chosen role *r* associates the driving role as the Role of type *Role1* with which it is associated.

As an example, suppose that a Client wishes to obtain an application form from a Clerk. In this case, there would be no existing association. The RolEnact would be:

```

Selection Client.obtain_form
    Me(initial → holding_form)
    Clerk(initial → form_issued)
End
    
```

Note that this code appears to be identical to that for an interaction. However, the ‘Clerk’ Role referred to above is not an (associated) instance of a role. Enact is told to choose

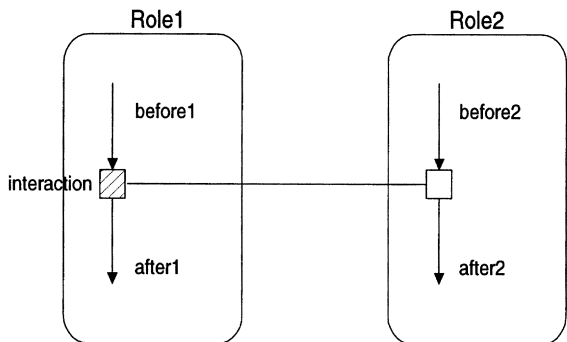


Fig. 4. A RolEnact selection as a RAD.

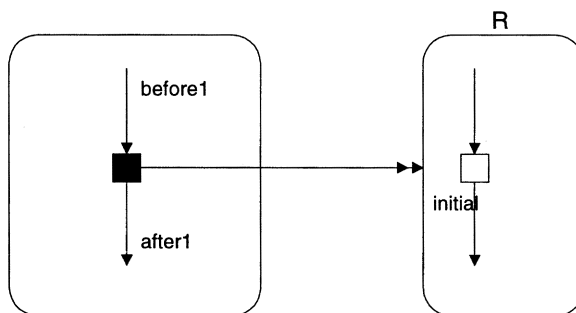


Fig. 5. A RolEnact creation as a RAD.

a role of the type ‘Clerk’ and to create an association. The association is invisible to the modeller, for whom interaction and selection may be treated as if they were the same. However, an association will be set up as follows.

```
Me.Clerk: = r      r.Client: = Me
```

That is, the driving role, *Client*, identifies the chosen role *r*, as the role of type *Clerk* to which it is associated. Similarly, the chosen role *r* (the *Clerk*) associates the driving role as the Role of type *Client* to which it is associated.

5.4. Creation

Creation is where a role creates a new instance of a role, and creates an association with that role. The creating role has the simple state change of all of the RolEnact constructs moving from its *before* state to its *after* state. The new role (*new Role2*), is declared, and will be created with the default state *initial*.

More formally when the role is in the state *before1*, it performs the creation and moves to the state *after1*, creating a new instance of type Role2 (see Fig. 5). In RolEnact, this is written.

```
Create Role1.Create
  Me(before1 → after1)
  new Role2
End
```

Again RolEnact will set up an association, just as for selection, and again this association is handled by the implementation of RolEnact, and hidden from the user.

```
Me.Role2: = r      r.Role1: = Me
```

Creation is shown as an action with a double arrowhead pointing at the created role. To enact the example, an additional role is needed. The extra role is not shown on the RAD, it used only by the RolEnact simulation, and it is not part of the business process being modelled. This role will be called *Control*. An example of creation is:

```
Create Control.newClient
  Me(initial → initial)
  new Client
End
```

The hidden association set up by Enact would be:

```
Me.Client: = r      r.Control: = Me
```

For the simple example described above, this association is not used, since the Control has no further interaction with any of the roles it creates. However, in principle, any role may create an instance of a role of another type and then communicate with that created role. For example, a superior will often create (instantiate) a subordinate role, and then interact with the subordinate on a number of occasions; giving initial instructions, checking on progress, being reported to and so on. Indeed, this is a scenario

that the authors have witnessed across many domains, and one that is naturally modelled using the creation construct.

6. The example in RolEnact

The RolEnact description of the example is now described in full. The RolEnact code mirrors the behaviour that one would expect when running instances of the example RAD.

6.1. Control

Control is an additional role and does not appear on the RAD of the example process (see Section 5.4). Its purpose is to create instances of the other roles in the model. In creating new instances of the ‘Client’ role, it simulates the external event that starts the process.

The Control role has three alternate paths, each consisting of a single ‘create’ activity and each returning the role to its initial state.

```
Create.newClient
  Me(initial → initial)
  new Client
End
```

```
Create.newClerk
  Me(initial → initial)
  new Clerk
End
```

```
Create.newCommittee
  Me(initial → initial)
  new Committee
End
```

The following descriptions, of Client, Clerk and Committee, will assume that role instances have been created (by Control), such that actions, interactions and selections may take place. However, no existing associations are assumed. Hence, the following role descriptions (Section 6.2–Section 6.4) model the behaviour as described by the RAD of Fig. 1.

6.2. Client

A ‘Client’ may select any ‘Clerk’ that is in an ‘initial’ state (see Section 5.3). The selection ‘obtain form’ moves the Client from an ‘initial’ state to the state ‘holding form’, and the ‘Clerk’ from initial to ‘form issued’.

```
Selection Client.obtain_form
  Me(initial → holding_form)
  Clerk(initial → form_issued)
End
```

The ‘Client’ role instance then carries out the action ‘complete form’ alone and consequently moves to the state ‘form completed’.

```
Action Client.complete_form
  Me(holding_form → form_completed)
End
```

The ‘Client’ is then able to interact with the previously selected ‘Clerk’ via the interaction ‘submit application’; moving to the state ‘awaiting decision’. Note that, the previously established association forces the ‘Client’ to interact with the correct ‘Clerk’; the one that issued the form.

```
Interaction Client.submit_application
  Me(form_completed →
    awaiting_decision)
  Clerk(form_issued →
    application_received)
End
```

Later, the ‘Client’ role is moved into its ‘final’ state by an interaction initiated by the ‘Client’ role.

6.3. Council Clerk

For brevity, the Council Clerk is referred to simply as ‘Clerk’ in the RolEnact model. Clerk (like all roles) starts in the default ‘initial’ state. The ‘obtain form’ selection (of the Client) moves the Clerk to the state of ‘form issued’. Subsequently an interaction with the same Client role instance, ‘submit application’, moves the Clerk from ‘form issued’ to the new state ‘application received’.

At this point the Clerk role has had two state changes, but has been passive in both. The clerk has now received the application and checks whether it is necessary to involve the Planning Committee in responding to the Client. If not then the Clerk makes the decision and moves to the state ‘decision ready’ (see below).

```
Action Clerk.decide
  Me(application_received →
    decision_ready)
End
```

If the application needs referral to the Committee, a selection² moves the Clerk from the state ‘application received’ to ‘awaiting decision’. This selection moves the Planning Committee (Committee) from the ‘initial’ state to the new state ‘meeting needed’, and sets up an association between the two role instances.

```
Selection Clerk.refer
  Me(application_received →
    awaiting_decision)
  Committee(initial →
    meeting_needed)
End
```

This will be followed by an interaction, driven by the Committee, which moves the ‘Clerk’ role into the state ‘decision ready’. Hence, both paths return to the same state; ‘decision ready’.

Finally, the Clerk may give the decision to the associated (correct) Client. As a result of this interaction ‘give decision’, the Client is moved to their ‘final’ state and the Clerk returns to an ‘initial’ state, in order to carry out further work.

```
Interaction Clerk.give_decision
  Me(decision_ready → initial)
  Client(awaiting_decision → final)
End
```

Note that the sequential nature of this process is of course highly inefficient, and the Clerk may become a bottleneck. Indeed, this is one of the aspects of process that modelling in notations such as RolEnact highlight, and one of the arguments for their usage.

6.4. Committee

The ‘Committee’ role, in common with the others, starts in an ‘initial’ state. The selection ‘refer’ of the ‘Clerk’ causes the ‘Committee’ to move to the state ‘meeting needed’ (see Section 6.3). From this state, the role is able to perform the action ‘decide at meeting’, and move to the state ‘decision taken’.

```
Action Committee.decide_at_meeting
  Me(meeting_needed →
    decision_taken)
End
```

Once in this state, the ‘Committee’ is able to initiate the interaction ‘publish minutes’, in which the ‘Committee’ returns to its initial state and the ‘Clerk’ is moved into the ‘decision ready’ state.

```
Interaction Committee.publish_minutes
  Me(decision_taken → initial)
  Clerk(awaiting_decision →
    decision_ready)
End
```

7. The RolEnact Windows interface

RolEnact is a Windows-based application written in Enact, a hybrid of object oriented and functional languages

² There is a need for the event involving the Committee to be a selection because it is the first association between the ‘Clerk’ and the ‘Committee’. In many real instances (and corresponding models) there will be only one ‘Committee’ and, hence, less need for the selection association.

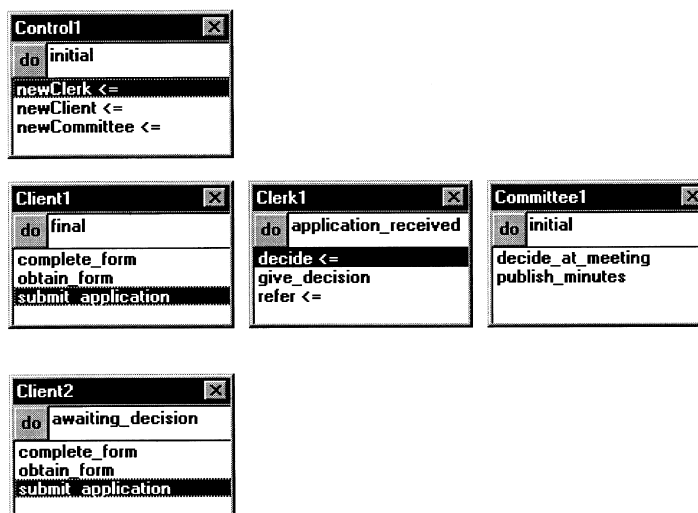


Fig. 6. The simple RolEnact interface.

[31]. A working model of RolEnact takes the form of a set of co-operating Windows programs each enacting an instance of a role in the business process being modelled [32].

Fig. 6 shows a point in the execution of a RolEnact model of the example process. Each role instance is shown as a separate window. Each window has four distinct parts: the name of the window, a menu displaying a list of possible actions that the role may perform, a text box which displays the current state of the role, and a 'do' button, which supports the enaction of a chosen action.

The window name takes the name of the role class followed by a digit (e.g. Clerk1). This is to allow multiple instances of a role. The first instance created will be post-fixed by '1', the second by '2', and so on (for example, 'Client1' and 'Client2'). The arrowed actions in the list are those available to be enacted in the current state of the system. For example, at the point in the process described in Fig. 6, neither Client is able to take part in any event. Client1 has reached the final state, and Client2 is waiting for a decision from the (only) Clerk³. The Clerk, being in the state 'application received', is about to either 'decide' or 'refer' the decision (these being the arrowed available actions). An action can be invoked by highlighting that event and pressing the 'do' button (or by double clicking). Suppose the Clerk makes the decision (enacts the activity 'decide'; the model user double-clicking on 'decide'). The Clerk instance will then move to their next state 'decision ready'. From this state, the Clerk may interact with Clients in the state 'awaiting decision', moving the Client instance to their final state, and returning to the 'initial' state.

It can be seen that each role instance in the enactable model acts as described by its type description given

³ Further Clients, Clerks or Committees may be created using the Control role.

in RolEnact (see above). Furthermore, the resulting behaviour is also in accordance with the RAD shown in Fig. 1.

8. Advantages of enactable role models

The advantage of being able to run a RolEnact model on a computer is that it allows both modellers and clients to check process understanding. Initially the modeller is able to check that they have an understanding of the current process, and to validate this by running through the model with representatives of the client organisation. Subsequent modelling allows for analysis of alternative process scenarios, finding sources of delay and so on. Finally, the model may be presented to or used by clients in order to make sure that it really reflects the intention of the Client Organisation. The following examples show the kind of issues that are easily illustrated by running a RolEnact scenario, and that are often likely to be missed by inspection of static models.

8.1. Correct assignment of responsibilities for driving the process

It is very easy in constructing the RAD to give little credence to the importance of which role drives the interaction. The interaction can easily be seen merely as a point of synchronisation for the roles, and a mechanism for communication. However, in the real world being modelled, driving the interaction is extremely important for the process as a whole to progress. It is necessary to assign the responsibility for driving the interaction to the correct role, since in the instantiation of the process it is necessary to assign the responsibility for initiating the interaction to an appropriate agent. Furthermore, since roles synchronise on

an interaction it may be important that it is clear which role initiates this interaction in order to minimise slack time in the process.

As an example, consider the ‘publish minutes’ interaction between the Committee and the Clerk (see Fig. 1). The diagram implies that the Committee drives the interaction. The Committee prepares minutes and publishes them to a (presumably waiting) Clerk, who extracts the data required to answer applications. However, it may be more efficient if the Clerk, knowing when the meeting is scheduled to take place were to request the minutes from the meeting. For example, it may be that the individuals comprising the committee consider the activity of generating and publishing minutes for their meetings to be a low priority. Giving the initiative for obtaining the decisions to the Clerk will ensure that the Clerk does not spend time waiting unnecessarily and that responses to applicants are not delayed.

Clearly, which of these scenarios is most useful depends upon the nature of other considerations within the process, such as resource availability, which actions are on the critical path and so on. However, a modeller would wish this to be visible to the client so that the correct decision is made. By using the RolEnact scenario, the role that is responsible for driving the interaction is made clear, since this and only this role which will include each interaction in its list of possible actions and interactions. Therefore, by running through such a scenario the client is forced to invoke each interaction from within the role to which it has been allocated, and this choice is made more explicit and visible.

9. Conclusions

The modelling notations presented in this paper share the features of a number of paradigms. The authors recognise that ease of use and understandability are important prerequisites for process modelling notations, particularly where such notations are to be used for process elicitation. That is, in order to find out about and understand the existing process, it is necessary to have notations that will be readily understood by non-technical users, with a relatively small entry cost. Diagrammatic methods appear to offer the best prospects for such use. Role-based perspectives appear to provide a natural mapping to business processes, and notations such as RADs offer both formality and understandability.

RolEnact appears to offer all of the advantages outlined above. It is based on a condition–action paradigm, yet it also has the advantage of providing a role-based perspective upon business processes. It is easy to use the notation to capture and describe business processes; particularly since it maps to RADs — a powerful and popular diagrammatic process modelling notation.

In addition, it is possible to generate RolEnact models which can be run on a computer to provide process

simulations so that users may experiment with processes. The paper has shown examples of how such experimentation can lead to increased process understanding, which users would be much less likely to gain using only static process models.

This use of RolEnact models is akin to the use of executable specifications in software development. The authors believe that by using enactable models of business process, process specification errors will be reduced, as will the costs of implementing processes, and process support. Furthermore, by having a more rigorous validation of the process specification the business processes, which are implemented, will better match the needs of client organisations.

References

- [1] M. Dowson, Are software processes business processes too, Panel Session of the Proceedings of the Third International Conference on the Software Process, Reston, VA, IEEE Computer Society Press, 1994.
- [2] R.A. Snowdon, A brief overview of the IPSE 2.5 Project, *Ada User* 9 (4) (1988) 151–161.
- [3] V. Ambriola, Related domains session, Proceedings of the Second European Workshop on Software Process technology, Vilard de Lans, Grenoble, France, February 1994, Lecture Notes in Computer Science, Elsevier Science, Amsterdam, 1994.
- [4] V. Gruhn, Software process management and business process (re-) engineering, Proceedings of the Second European Workshop on Software Process technology, Vilard de Lans, Grenoble, France, February 1994, Lecture Notes in Computer Science, Elsevier Science, Amsterdam, 1994.
- [5] B. Kramer, B. Dinler, Applying process technology to hardware design, Proceedings of the Second European Workshop on Software Process technology, Vilard de Lans, Grenoble, France, February 1994, Lecture Notes in Computer Science, Elsevier Science, Amsterdam, 1994.
- [6] L.J. Osterweil, Software processes are software too, Proceedings of the Third International Software Process Workshop, Breckenridge, CO, IEEE Computer Society Press, 1986.
- [7] L.J. Osterweil, Experiences with process programming, Proceedings of the Fifth International Software Process Workshop, Kennebunkport, Maine, USA, IEEE Computer Society Press, 1989.
- [8] L.J. Osterweil, Example process program code, coded in Appl/A, Proceedings of the Fifth International Software Process Workshop, Kennebunkport, Maine, USA, IEEE Computer Society Press, 1989.
- [9] A. Ohki, K. Ochimizu, Process programming with prolog, Proceedings of the Fourth International Software Process Workshop, Moretonhampstead, Devon, UK, IEEE Computer Society Press, 1988.
- [10] M.M. Lehman, Some reservations on software process programming, Proceedings of the Fourth International Software Process Workshop, Moretonhampstead, Devon, UK, IEEE Computer Society Press, 1988.
- [11] M.M. Lehman, Models in software development and evolution, *Software Process Modelling in Practice*, Kensington Town Hall, London, UK, Butterworth-Heinemann, 1993.
- [12] G.E. Kaiser, Constructing enactable models, Proceedings of the Fourth International Software Process Workshop, Moretonhampstead, Devon, UK, IEEE Computer Society Press, 1988.
- [13] B. Boehm, F.C. Belz, Applying process programming to the spiral model, Proceedings of the Fourth International Conference on the Software Process Workshop, Moretonhampstead, Devon, UK, IEEE Computer Society Press, 1988.

- [14] P. Henderson, Object-oriented specification and design with C + + , The McGraw-Hill International Series in Software Engineering, McGraw-Hill Book Company, New York, 1993.
- [15] B.W. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [16] A. Gravell, P. Henderson, Executing formal specifications need not be harmful, *Software Engineering Journal* 11 (2) (1996).
- [17] M.A. Ould, C. Roberts, Modelling iteration in the software process, Proceedings of the Third International Software Process Workshop, Breckenridge, Colorado, USA. 17–19 November 1986, IEEE Computer Society Press, 1986.
- [18] N.H. Madhavji, The Process Cycle, *Software Engineering Journal* 6(5) (1991) 234–242.
- [19] D. Miers, Use of tools and technology within a BPR initiative, in: *Business Process Re-engineering: Myth and Reality*, Kogan Page, London, 1994.
- [20] B. Curtis, M.I. Kellner, J. Over., Process Modelling Communications of the ACM 35 (9) (1992) 75–90.
- [21] E. Yourdon, *Modern Structured Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1989.
- [22] G. Tate, Software process modelling and metrics, *Information and Software Technology* 35 (6/7) (1993) 323–330. Special Issue on Process Modelling in Practice.
- [23] C. McGowan, L. Bohner, Model based process improvement, Proceedings of the 15th International Conference on Software Engineering, Baltimore, MD, IEEE Computer Society Press, 1993.
- [24] ICL, ProcessWise WorkBench User Guide, PWB/usrguide/S5.4, International Computers Limited, August 1995.
- [25] M.A. Ould, An introduction to process modelling using RADs, illustrated by reference to the case study, in: *IOPTCLUB Practical Process Modelling*, Mountbatten Hotel, Monmouth Street, Covent Garden, London, 1992.
- [26] C.B. Handy, On roles and interactions, in: *Understanding Organisations*, Penguin Modern Management Texts, Penguin Books, Harmondsworth, Middlesex, England, 1976.
- [27] M.A. Ould, *Business Processes Modelling and Analysis for Re-engineering and Improvement*, Wiley, New York, 1995.
- [28] G.K. Abeyasinghe, K.T. Phalp, Combining process modelling methods, *Information and Software Technology* 39 (2) (1997) 107–124.
- [29] Co-Ordination, RADitor version 1.5: Users Manual, Co-Ordination Systems, 1994.
- [30] GISIP, Geographical Information Systems Integration Process, 1996. Esprit project home page at: <http://dsse.ecs.soton.ac.uk/~kp/gisip.html>.
- [31] P. Henderson, Enact User Manual, April 1995, available at <http://dsse.ecs.soton.ac.uk/~peter/cv.html>.
- [32] P. Henderson, Making Models of Process Support in Enact, November 1995, available at <http://dsse.ecs.soton.ac.uk/~peter/cv.html>.