12

Coupling Process Models and Business Rules

Peter McBrien
King's College London
Strand, London WC2R 2LS, UK
Tel: +44 171 873 2469, email: pjm@dcs.kcl.ac.uk

Anne Helga Seltveit
The Norwegian Institute for Technology
Trondheim N-7034, Norway
Tel: +47 73593677, email: ahs@idt.unit.no

Abstract

Two techniques commonly used in the conceptual modelling of information systems are **process modelling** and **business rule modelling**. In this paper we propose a technique for associating certain types business rules with structures in a process modelling language. This **coupling** of the two models allows them to be used as complimentary languages in conceptual modelling; the process language being suitable when modelling how activities interact, whilst the business rule model is suitable when we need to make precise statements about a certain activity. The ability to model certain aspects of business rules within the process model is particularly important in distributed organisations, where the process model may be used as a means of communication between different parts of the organisation. The coupling also serves (1) to make apparent what effect **re-engineering** of one model has on the structure of the other model, and (2) indicate how the process model may be used to drive the creation of business rules.

Keywords

Conceptual modelling, Process modelling, Business rule modelling, Business process re-engineering.

1 INTRODUCTION

Process modelling is frequently used in the conceptual modelling of information systems, and allows the description of the dynamic nature of the **universe of discourse** (UoD), usually in the form of some variant of the **data flow diagram** (DFD) of (Gane and Sarson 1978) or (deMarco 1978). Although useful for describing a general picture of how the domain functions in terms of activities (processes) and exchanges of information between the activities (dataflows), the semantics of process models are not precise in their statement of facts about the domain. In particular it is not possible to generate any application code from a DFD model.

In part as an answer to the lack of semantic richness in process models, **business rule** modelling has received growing attention over the past few years, one research project TEMPORA (Loucopoulos, McBrien, Schumacker, Theodoulidis, Kopanas and Wangler 1991) being of interest to this paper, and commercial products using them being exemplified by UNIFACE 6. Business rules are usually expressed in a formal language, which allows precise statements to be made about the domain, and which contain sufficient information such that application code may be automatically generated.

A disadvantage of business rules is that they suffer from a lack of structuring mechanisms — the **rule base** is an essentially flat structure, and in large information systems this can become difficult to manage, test and maintain (Xiaofeng 1991). This will become more apparent in distributed organisations, where the flat rule structure must be mapped onto some structure in the organisation. By contrast, the ability to provide different levels of abstraction and to structure the model as communicating activities can be seen as advantages of process modelling, In particular, process modelling supports the notion of interaction between different parts of a distributed organisation.

In (Krogstie, McBrien, Owens and Seltveit 1991) it was proposed that the two models could be coupled, by defining how certain constructs in one model would indicate the presence of constructs in the other model. Rules were viewed as having the general structure

$$\textsf{when } \langle\text{trigger}\rangle \textsf{ if } \langle\text{condition}\rangle \textsf{ then } \langle\text{consequence}\rangle$$

where ⟨trigger⟩, ⟨condition⟩ and ⟨consequence⟩ are logical expressions which may contain several flow names. Such rules would imply the existence of a process which has triggering input (the flows contained in ⟨trigger⟩), non-triggering input (the flows contained in ⟨condition⟩), and output (the flows contained in ⟨consequence⟩).

In this paper we elaborate upon the approach in (Krogstie et al. 1991) to define how the structure of the rules may be more precisely defined within the process model. This tighter coupling facilitates the following activities:

- **rule capture** by the ability to express certain logical constructs within the process model.
- **process re-engineering** being carried out in the process model, since the consequences of such changes in the rule model are made apparent immediately.
- **business rule re-engineering** being carried out in the rule model, since the consequences of such changes in the process model made apparent immediately.
- **communication of business rules** between parts of a distributed organisation, since the relationship of rules to processes means that business rules can be located within a structural framework of the organisation as represented by the process model.

The remainder of this paper is structured as follows. Section 2 briefly describes the TEMPORA process and rule modelling languages. Section 3 describes how the two models may be coupled. Section 4 describes the rule capture process. Section 5 describes the use of the coupling mechanism in information systems re-engineering.

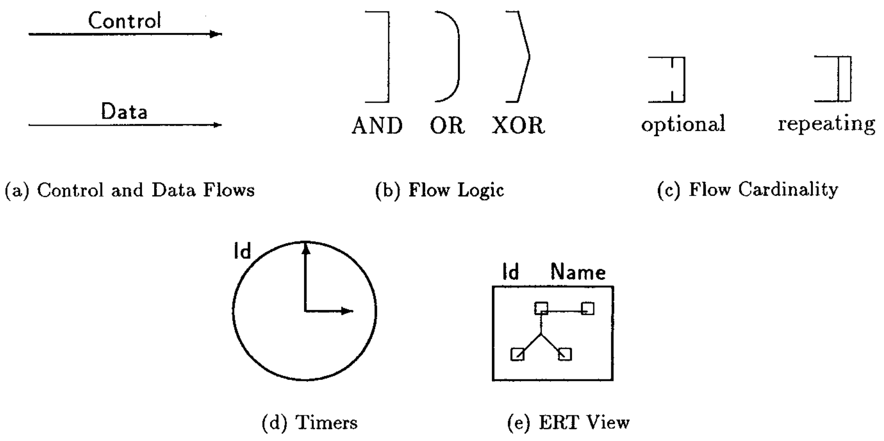


Figure 1 Enhancements on DFD found in the PID.

2 THE TEMPORA PROCESS AND RULE MODELS

In TEMPORA the capture of a conceptual model is made using three modelling languages:

- An entity-relationship model called the ERT (McBrien, Seltveit and Wangler 1992), containing various extensions to the ER model of (Chen 1976), including the ability model valid-time data model (Jensen *et al* 1984).
- A process modelling language called the PID (Krogstie et al. 1991), containing extensions to DFD modelling which we detail below, and based on the PPP modelling language (Gulla, Lindland and Willumsen 1991).
- A rule modelling language called the ERL (McBrien, Niezette, Pantazis, Thedoulidis, Tziallas, Seltveit, Sundin and Wohed 1991), which allows the specification of rules as logical constraints on the static ERT and dynamic PID models. The ERL uses as its basis the temporal **US logic** (Gabbay 1989), allowing the querying of the valid-time data model in the ERT. The use of the ERL as a mechanism for specifying logical constraints on the PID is the subject of Section 3.

So as to make this paper self contained, we detail a few key aspects of the PID and ERL in the following subsections.

2.1 An Enhanced Process Modelling Language: PID

The key differences between the PID model and the DFD model as proposed by (Gane and Sarson 1978) are itemised below, and the new constructs introduced in the PID are illustrated in Figure 1.

- The PID distinguishes between control and data flows. A **triggering flow** is drawn with a thick line. Each occurrence of a triggering flow causes the processes at which it ends to be *considered* for execution. A **non-triggering flow** is drawn with a thin line, and may only be used by a process that is already executing.
- A triggering flow only leads to a process being considered for execution. It will only *actually* execute if the **flow logic** is satisfied. The logic is specified using a combination of **AND ports, OR ports** and **XOR ports**. This indicates what logical combinations of input flows must be present in order for the process to execute.
- The **flow cardinality** is the number of instances of a flow that must be present for a valid process execution. By default it is assumed that one instance of each input flow is used in a process execution, and that one instance of each output flow is generated by the execution. The fact that there may be a lower bound of zero is indicated by a **conditional port**, and an upper bound of many is indicated by a **repeating port**.
- The notion that clock events may be generated at specified intervals is represented by the concept of a **timer**.
- The fact that data stores are views on an entity-relationship model leads to a new syntax for the object, which is called an **ERT view**.

2.2 A Rule Modelling Language: ERL

The ERL is intended to allow the semantic modelling of logical statements about the static model captured in the ERT, and the dynamic view in the PID. In conceptual modelling, we may interpret all rules as stating logical constraints on the models they are associated to, and all rules are based on a single structure:

$$\textsf{when}\ \langle\text{trigger}\rangle\ \textsf{if}\ \langle\text{condition}\rangle\ \textsf{then}\ \langle\text{consequence}\rangle$$

A rule holds when its ⟨trigger⟩ has just begun to hold (i.e. an event has occurred which makes ⟨trigger⟩ true), and its ⟨condition⟩ holds. If a rule holds then its ⟨consequence⟩ must also hold, and if this cannot be made the case then an exception should be raised. If either the ⟨trigger⟩ or ⟨condition⟩ part is omitted then that part is assumed to hold. Since trigger events must be modelled in the PID, a rule with a non-empty ⟨trigger⟩ must be associated to some process in the PID, in a manner we will detail in Section 3.

The ⟨trigger⟩, ⟨condition⟩ and ⟨consequence⟩ all have the syntax of an ⟨ERL expression⟩. The ⟨ERL expression⟩ can contain ⟨flows⟩, ⟨ERT access expressions⟩, ⟨logical connectives⟩, ⟨temporal connectives⟩, or ⟨aggregates⟩ (i.e. sets).

- ⟨ERT access expressions⟩ are ERL constructs which select data from an ERT model, where the classes are named, and relationships are used to restrict the selection of instances from classes in some way.
- ⟨flows⟩ name tuples of data (single or aggregates) selected from the ERT. They allow information to be shared between rules, and model the flows in the PID model.
- ⟨logical connectives⟩ are the usual first order logical connectives, together with an n-ary exclusive or. We list the connectives in Table 1, where $e_1, e_2, \ldots$ are instances of ⟨ERL expression⟩
- ⟨temporal connectives⟩ are temporal logic counterparts to the first order connectives, and for the purposes of this paper need not be detailed further. However, it should be

Table 1 Logical propositions and connectives.

ERL expression	*Semantics*
true	Always true
false	Always false
e_1 and e_2	Both e_1 and e_2 hold for some set of variable substitutions.
e_1 or e_2	Either e_1 or e_2 holds for some set of variable substitutions.
not e_1	e_1 does not hold for the set of variable substitutions applied to the expression which the connective is part of.
only_one_of($e_1, e_2, \ldots$)	Holds if exactly one of $e_1, e_2, \ldots$ holds.

noted that their presence, together with the coupling between ERL and PID, give the process model temporal semantics.

- ⟨aggregates⟩ of the variables $v_1, v_2, \ldots$ may be formed by the expression:

$$\{v_1, v_2, \ldots \text{for\_which } \langle \text{ERL expression} \rangle\}$$

The semantics of rules lead to the following requirements on what must be placed in each part of a rule:

- In the ⟨trigger⟩ part of the rule, we describe what must have just become true for the rule to be true. The trigger should contain the events which must have just occurred if the rule is relevant. The use of ⟨ERT access expressions⟩ or ⟨flow⟩ in the ⟨trigger⟩ will manifest themselves as triggering input flows in the PID.
- In the ⟨condition⟩ part of the rule, we describe what needs to be true for the rule to be considered. The condition should contain statements which must be true (and may have been true for some time) if the rule is relevant. If the condition is false, then the rule does not hold, no action or effect will result. The use of ⟨ERT access expressions⟩ or ⟨flow⟩ in the ⟨condition⟩ will manifest themselves as non-triggering input flows in the PID.
- In the ⟨consequence⟩ part of the rule, we state what must be true if the condition is true. The execution of a TEMPORA specification will attempt to ensure that the consequence is made true if the condition is true. If for any reason this can not be achieved, an error will raised in the runtime system. The use of ⟨ERT access expressions⟩ or ⟨flow⟩ in the ⟨consequence⟩ will manifest themselves as output flows in the PID.

3 COUPLING THE RULE AND PROCESS LANGUAGES

This section details how the connection between the ERL and the PID at the conceptual level can be made, using the semantics of the PID and the ERL as a basis.

Both the ERL and the PID can be used to describe the dynamics of a system, and thus there is clearly a degree of overlap in what the languages model in the domain. This motivates us to consider what precisely is to be contained in each model, and the manner in which concepts shared between the models cause one model to define what must appear in the other. We refer to this interaction as a **coupling** between rule and process modelling languages.

3.1 Defining the Basic Coupling

As already mentioned in Section 2.2, different parts of an ERL rule correspond to the use of flows in a PID model. We say that the rule is **associated** to the process which has as input and output the flows which the rule uses. The basic relationship is depicted in Figure 2 and can be described as follows:

- The *trigger* part of the process structure, i.e. the triggering flow α entering a process corresponds to the ⟨trigger⟩ part of an ERL rule.
- The *conditional* part of the process structure, i.e. non-triggering flow β entering a process, corresponds to the ⟨condition⟩ part of an ERL rule. The process logic γ should also be included in the ⟨condition⟩ part. Since the ERL expressions are logical formulae, they can be rewritten to **disjunctive normal form** (DNF), where an ⟨ERL expression⟩ e may be regarded as taking the form:

  $$e = (e_{11} \textsf{ and } e_{12} \ldots) \textsf{ or } (e_{21} \textsf{ and } e_{22} \ldots) \ldots$$

  Thus a condition c has a DNF $c_1 \textsf{ or } c_2 \textsf{ or} \ldots$ where each of $c_1, c_2, \ldots$ can be divided into an input part $c_{1,\beta}$ and a processing part $c_{1,\gamma}$, giving c the form:

  $$(c_{1,\beta} \textsf{ and } c_{1,\gamma}) \textsf{ or } (c_{2,\beta} \textsf{ and } c_{2,\gamma}) \textsf{ or} \ldots$$

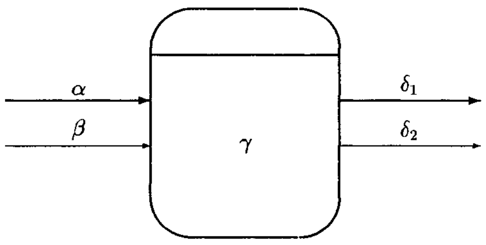
  The input part $c_{1,\beta}$ may contain a flow or name ERT objects in an ERT access expression, and the processing part $c_{1,\gamma}$ may contain ERL expressions excluding flows and ERT objects, such as arithmetic expressions.
- The *action* part of the process structure, i.e. the output flows, corresponds to the then part of an ERL rule. The action part may contain both triggering flows δ_1 and non-triggering flows δ_2.

We have now presented the basic connection between the ERL and the PID by describing the way in which ⟨trigger⟩, ⟨condition⟩, and ⟨consequence⟩ fields are reflected in the PID and vice versa. We now elaborate how the various constructs of the PID detailed in Section 2.1 may be associated to parts of an ERL rule.

Processes

A process requires information from its surroundings, that is, it interacts with other processes, ERT views, timers, and external agents via flows. A process may have one or more input flows and one or more output flows. A process' combinations of flows (input flows or output flows) is expressed using the concept of *ports* and control aspects are expressed using *triggering flows.* Triggering flows and ports make it possible:

- To represent permitted combinations of input/output flows for each process, i.e. the way the various elements of the ⟨trigger⟩, ⟨condition⟩, and ⟨consequence⟩ fields are logically related.



when α if β and γ then δ_2 and δ_1

Figure 2 Relationship between the PID and the ERL.

- To express knowledge about the sequencing of processes.
- To express which flows a process may only optionally use or produce, and which flows may be used or produced a multiple number of times.

For the moment, we will describe only how a process including ports may be described by a *single* rule expressed using the ERL. In the Figures 3 to 7, each construct of the diagram syntax of a PID process including ports is shown together with its logical semantics expressed in the ERL. The details of port construction in the PID are described in (Yang 1993).

The equivalences between AND ports and the ERL and logical connective are intuitive. However for OR, we have used a meta predicate ϕ with the following definition:

$$\phi(A, B) \equiv (A \text{ and not } B) \text{ or } (\text{not } A \text{ and } B) \text{ or } (A \text{ and } B)$$

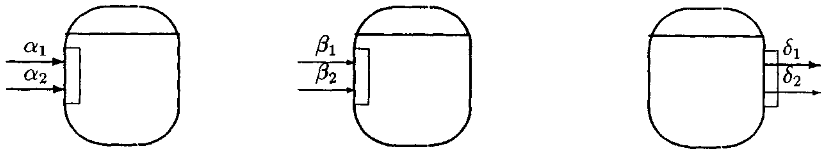
This is necessary since if a logical disjunction is executed using SLD resolution (Hogger 1990) (such as in Prolog execution), then the disjunction holds twice when both its disjuncts hold, whilst the semantics of OR are that it holds once when either or both its disjuncts hold. It is the XOR port which corresponds to logical disjunction executed using SLD resolution, i.e. the execution occurs once for each connective, but not together as a single execution.

Flows

A flow in the PID is described by a predicate notation in the ERL called ERL flow. For example, a flow new_part carrying parameters n and p is written as new_part(n,p). It may link rules described by two processes, a process and an ERT view, a process and an external agent, or a process and a timer.

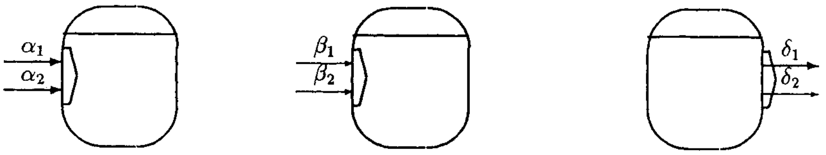
ERT views

An ERT view is defined in terms of an ERT access expression in the ERL. This selection constitutes a part of the ERT and is a structural representation of the contents of the ERT view. A process may read from an ERT view (corresponding to a flow from an ERT view to the process) or a process may write to an ERT view (corresponding to a flow from



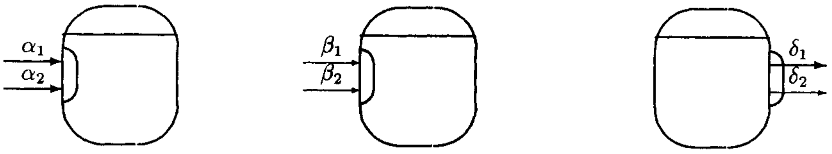
(a) when α_1 and $\alpha_2 \ldots$ (b) $\ldots$ if β_1 and $\beta_2 \ldots$ (c) $\ldots$ then δ_1 and δ_2

Figure 3 The semantics of AND ports expressed using the ERL.



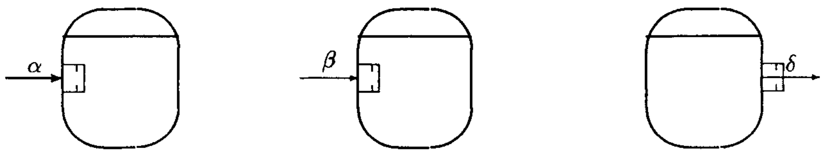
(a) when α_1 or $\alpha_2 \ldots$ (b) $\ldots$ if β_1 or $\beta_2 \ldots$ (c) $\ldots$ then δ_1 or δ_2

Figure 4 The semantics of XOR ports expressed using the ERL.



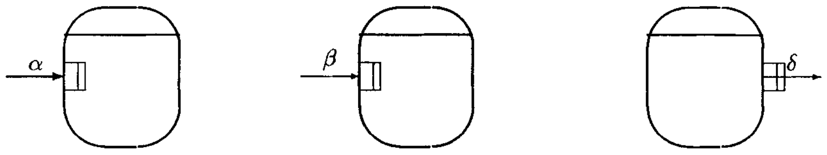
(a) when $\phi(\alpha_1, \alpha_2) \ldots$ (b) $\ldots$ if $\phi(\beta_1, \beta_2) \ldots$ (c) $\ldots$ then $\phi(\delta_1, \delta_2)$

Figure 5 The semantics of OR ports expressed using the ERL.



(a) Illegal structure (b) . . . **if** β **or true** . . . (c) . . . **then** δ **or true**

Figure 6 The semantics of the conditional ports expressed using the ERL.



(a) Illegal structure (b) . . . **if** $\{\beta\}$ . . . (c) . . . **then** δ**member_of** . . .

Figure 7 The semantics of the repeating ports expressed using the ERL.

the process to an ERT view). This may be expressed by ⟨ERT access expressions⟩ in the ERL:

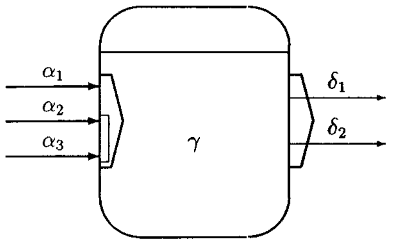
- From ERT view via triggering flow: **when** ⟨ERT access expression⟩ . . .
- From ERT view via non-triggering flow: . . . **if** ⟨ERT access expression⟩ . . .
- To ERT view: . . . **then** ⟨ERT access expression⟩

External Agents

An external agent may be represented by an ERL rule describing the objects which are passed from/to it in terms of ERT objects. These are expressed by ERL flows, and thus the coupling defined in Figures 3 to 7 apply to flows from external agents.

Timers

The details of timers can be expressed using temporal constructs in the ERL. The temporal constructs may contain temporal connectives such as **sometime_in_past** and **just_before**, references to time points through predicates such as **time_is** and references to time intervals through functions such as **today** and **this_week**. Details of the temporal constructs provided by the ERL may be found in (McBrien et al. 1991).



when α_1 or α_2 and α_3
if γ
then δ_1 or δ_2

(a) PID process and ports (b) ERL rule

Figure 8 Nesting of ports.

4 RULE CONSTRUCTION FROM A PID SPECIFICATION

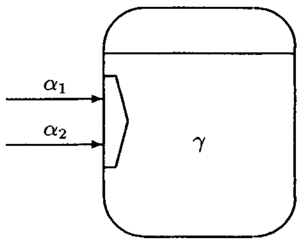
Having defined the basis of how a PID process can be described by a single rule expressed using the ERL, it should be clear that we can use the process model as a basis for rule capture. If the port structures in the PID are simple in nature, this will lead to rules which also have a simple logical structure. Again if ports of processes are nested in a complex structure, the resulting rule structure will also become complex. For practical purposes we need some method to reduce this complexity. Furthermore, we must deal with various levels of cooperating processes, that is, how rules are related to processes at various levels of decomposition. There are three issues we have to deal with when constructing rules from a PID specification:

1. How do nested input and output ports affect the rule construction.
2. How does the relationship between input flows and output flows affect rule construction.
3. How does rule construction vary when dealing with decomposed or non-decomposed processes.

4.1 Nested input ports and nested output ports

A process including nested input ports together with the resulting rules is shown in Figure 8. To construct the rules associated with this process, the ⟨trigger⟩, ⟨condition⟩, and ⟨consequence⟩ fields of a basic ERL rule are extracted from the process structure according to the definitions of the coupling in Figures 3 to 7. The rules are applied to the innermost ports first, being used to generate a 'flow' in the form of an ⟨ERL expression⟩ to be used in the construction for the next port moving outwards. For the example, the two triggering flows α_2 and α_3 use the rule in Figure 3(a) to yield α_2 and α_3, then we use this as input to use the rule in Figure 4(a) to get the ERL rule shown in Figure 8.

Clearly, the greater the depth of the nesting, the more complicated the resulting rule. However, we may simplify the situation by instead generating multiple rules. As already detailed in Figure 4(a), a process with two input flows to an XOR port may be modelled



(a) PID process and ports

when α_1
if γ_1
then ...

when α_2
if γ_2
then ...

(b) ERL rules

Figure 9 Generating multiple rules from an XOR port.

as a rule with a logical disjunction in the trigger part when α_1 or α_2.... As is the case in standard Prolog, this disjunction may be replaced by two separate rules as shown in Figure 9(b), each of which has as its trigger one of the triggers of the XOR port. This in practice will tend to be the more natural representation of the specification described by PID model in Figure 9(a).

If disjunctions appear between non-triggering flows (corresponding to the ⟨condition⟩ part of a rule) or between output flows (corresponds to the ⟨trigger⟩ part of rule), a composite rule can be split into simpler rules in the manner illustrated in Figure 9.

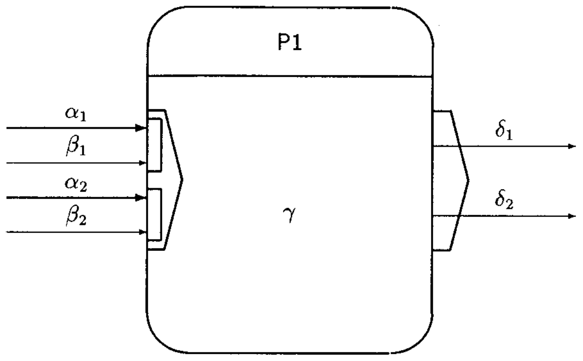
The process model shown in Figure 10(a) would lead to a trigger of the form when ($\alpha_1$and β_1) or (α_2 and β_2).... Again the technique of using separate rules for a logical disjunction we would get the rules shown in Figure 10(b). We could also split the action part of each of the rules shown, giving a total of four rules. However, this ignores what the process internals might indicate as far as the rules to be generated, which we will now consider.

4.2 Relationship between input flows and output flows

It should be noted that so far we have not taken the internals (i.e. γ) of the process into consideration. Thus, there is a certain ambiguity in the rules generated — i.e. there is part of the domain which the process model does not capture, and which must be specified in the rule language.

From what we have already described, for a process model such as that shown in Figure 10(a), the output δ_1 or δ_2 will be in the ⟨consequence⟩ part of both rules. However, it is quite probable that δ_1 is associated to only α_1 and that δ_2 is associated to only α_2, which would mean that the rules in Figure 10(c) would be a correct interpretation of the process model.

To decide which of δ_1 or δ_2, δ_1, or δ_2 should be in the ⟨consequence⟩, the relationship between the inputs and outputs must be considered. We may represent the way inputs and outputs of a process are related to each other in a decomposition of the process. In



(a) PID process and ports

when α_1 and β_1 if γ_1 then δ_1 or δ_2	when α_1 and β_1 if γ_1 then δ_1
when α_2 and β_2 if γ_2 then δ_1 or δ_2	when α_2 and β_2 if γ_2 then δ_2
(b) ERL rules	(c) Alternative ERL rules

Figure 10 Multiple rules and nested ports.

Figure 11 the internals of the process in Figure 10 are taken into account. By doing this, the ambiguity in the two ERL rules as described above is removed.

In Figure 12(a), we give a simple example of our method applied in a practical situation. We have modelled a process **staff_payments** which handles all payments made to staff, and have discovered that payments are based on **salary** or **expenses** records, and are triggered by a command from a clerk or the temporal event of it being midnight. The basic coupling between processes and rules would give the rule illustrated in Figure 12(b). This might seem over complex, and thus we may use the disjunction elimination to produce the four rules shown in Figure 12(c). At this point, the various possible behaviours of the process are made more apparent, and our analysis can check that all the rules produced correspond to the domain. It might be the case, for example, that we never produce salary cheques on the command of the clerk, and thus the rule

when pay_staff_no(...) if $\langle$ERT access expression$\rangle_{\text{salary}}$ then give_cheque(...)

may be eliminated — a restriction on the process logic specified in Figure 12(a) which

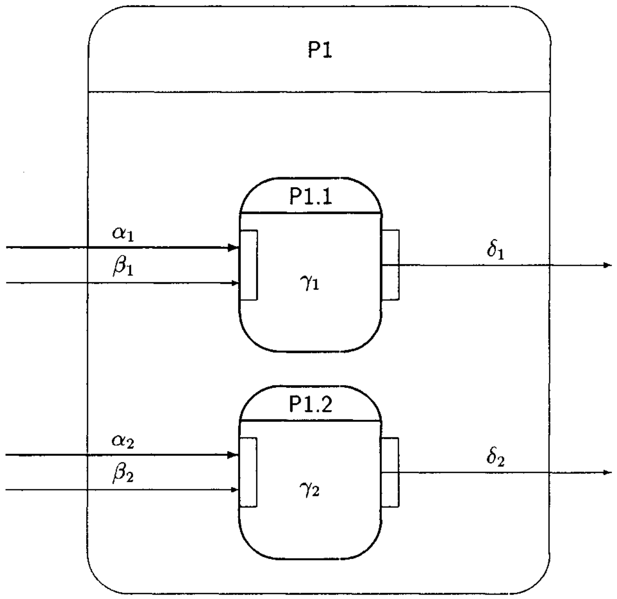


Figure 11 Example of a decomposed process and its corresponding ERL rules.

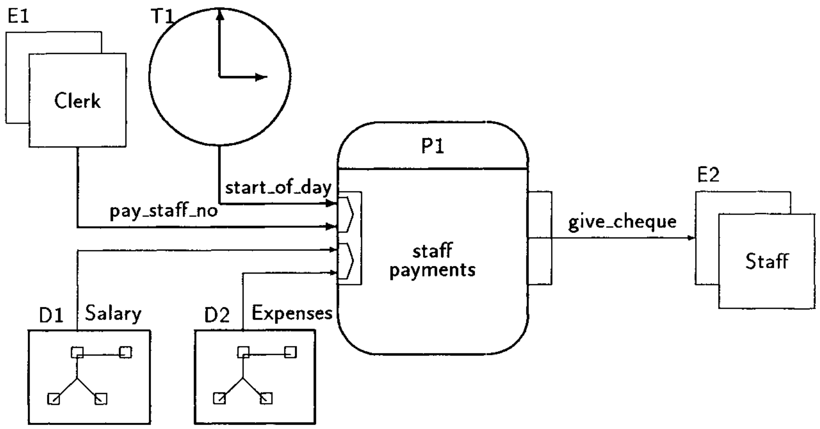
we may specify in the process model by decomposition or modifying the port structure. Once we are satisfied that we have a set of rules which correctly reflect the domain, we may proceed to specify the details of the process logic by expanding the condition part of the ERL rules.

4.3 Decomposed versus non-decomposed processes

Each non-decomposed process should have an associated set of ERL rules describing the behaviour of the process. In addition, one may optionally specify in rules the behaviour of decomposed processes, these rules being interpreted as constraints on the behaviour of the rules describing the non-decomposed processes. The rules associated with the processes at the lowest level of decomposition constitute the rule base which may be compiled to executable form. Rules associated with decomposed processes may be used as constraints on the execution of the rules of non-decomposed processes.

5 RE-ENGINEERING OF PROCESS AND RULE MODELS

The strong coupling between rules and processes described in Section 3 may be used during business process re-engineering to facilitate the modification of models. A particular step



(a) PID model to process payments due to staff

when start_of_day or pay_staff_no(...)
if ⟨ERT access expression⟩$_{\text{salary}}$ or ⟨ERT access expression⟩$_{\text{expenses}}$
then give_cheque(...)

(b) P1 represented as a single rule

when start_of_day
if ⟨ERT access expression⟩$_{\text{salary}}$
then give_cheque(...)

when start_of_day
if ⟨ERT access expression⟩$_{\text{expenses}}$
then give_cheque(...)

when pay_staff_no(...)
if ⟨ERT access expression⟩$_{\text{salary}}$
then give_cheque(...)

when pay_staff_no(...)
if ⟨ERT access expression⟩$_{\text{expenses}}$
then give_cheque(...)

(c) Disjunctions removed from the single rule

Figure 12 Example of ERL and PID Relationships.

in the process of business process re-engineering may involve changing either the process model or the rule model. We now detail how changes in one model reflect on the re-engineering of the other.

5.1 Process Model Updates

Modifications in the process model will imply changes be made in the rule model.

- The deletion of a construct in the PID implies the deletion of the construct in the rules associated to the process. The user may view these rules to ensure that the deletion of the PID construct was a correct procedure to follow.
- The addition of a process in the PID implies the creation of a rule in the rule model. The ports used to describe the process may be used to give a structural template of the rule to be created.
- The addition of a port or flow to a process in the PID implies the alteration of the logic used in the rules used to implement then process. The user may be prompted with the rules that need modification.

For example, if we decided in Figure 12(a) that payments to staff will now only be triggered by a direct command from the clerk, and not by a temporal event, then the two rules in Figure 12(c) that begin **when start_of_day** must be removed, and any details we have added to the condition part in the form of process logic must be incorporated in the two remaining rules.

5.2 Rule Model Updates

Modifications in the rule model will imply changes be made in the process model.

- The deletion of a rule which is associated to a PID process, and for which some construct of the process no longer has a rule associated to it, implies that construct should be revised. Either a replacement rule needs to be created, or the construct deleted.
- The alteration of the logic used in a rule implies the alteration of any associated process logic.
- The creation of a rule which includes a ⟨trigger⟩ component implies that the rule be associated to some process in the PID. This process may already exist, or it may be required that some modelling activity must proceed in order to produce a new process.

For example, if we decided in Figure 12(c) that payments to staff which exceeded a certain amount would be made by multiple cheques, then we would need to alter the process model to make the output from P1 a repeating port.

6 SUMMARY & CONCLUSIONS

The motivation for the coupling between the process based and rule based approaches was to reduce the fragmentation of specifications when multiple languages are used for conceptual modelling. In achieving this, we utilises each language's strengths and used them as complementary languages for information systems modelling. Three particular areas where the strengths of one language served to enrich the other were:

- The process model gives a flexible representation and presentation of the rule model. This allows us to have both a form based representation (in the ERL) and a graphical representation of rules (in the PID). The form based representation uses a formal language, allowing the precise statement of the domain, and suited to the use of automated application generation. The graphical representation of rules uses a less formal language, but allows the structure of specifications to be made readily apparent.
- The ERL language has temporal connectives for the querying and update of a valid-time data model. With the ERL-PID coupling in place, this serves to give the process model temporal semantics.
- The rule model serves to define the process logic of the process model, and hence turns it into an executable specification. We may use the rules as a basis for animating the model, as well as generating application code.

Thus, in combining the process based and rule based approaches we exploit the *user-friendliness* and the *structural properties* of a process model and the *expressive power* and *formality* of a rule model. We thus allow development and maintenance to take place at the correct level of detail. The use of rule modelling in distributed environments is supported by the structuring capabilities of the process model. Re-engineering precise statements about activities is made on rules, and the process model altered to reflect the changes to rules. Re-engineering of how activities interact takes place in the process model, and the rule model then altered to reflect those changes.

7 ACKNOWLEDGEMENTS

The work reported in this paper was partly funded by the Commission of the European Communities under the ESPRIT R&D programme. The TEMPORA project (number E2469) was a collaborative project between: BIM, Belgium; Hitec, Greece; Imperial College, UK; LPA, UK; SINTEF, Norway; SISU, Sweden; University of Liège, Belgium and UMIST, UK.

REFERENCES

Chen, P. P. S.: 1976, The Entity-Relationship model: Toward a unified view of data, *ACM TODS, vol. 1, no. 1.*

deMarco, T.: 1978, *Structured Analysis and System Specification*, Yourdon Press.

Gabbay, D.: 1989, The declarative past and executable future, *Temporal Logic in Specification: Altrincham Workshop 1987*, LNCS, Springer-Verlag, pp. 409–448.

Gane, C. and Sarson, T.: 1978, *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall.

Gulla, J., Lindland, O. and Willumsen, G.: 1991, PPP an integrated CASE environment, *Proceedings of the Third Nordic Conference on Advanced Information Systems Engineering*, Vol. 498 of *LNCS*, Springer-Verlag.

Hogger, C.: 1990, *Essentials of Logic Programming*, Vol. 1 of *Graduate Texts in Computer Science*, OUP.

Jensen *et al*, C.: 1984, A consensus glossary of temporal database concepts, *SIGMOD Record.*

Krogstie, J., McBrien, P., Owens, R. and Seltveit, A.: 1991, Information systems development using a combination of process and rule-based approaches, *Proceedings of the Third Nordic Conference on Advanced Information Systems Engineering*, Vol. 498 of *LNCS*, Springer-Verlag.

Loucopoulos, P., McBrien, P., Schumacker, F., Theodoulidis, B., Kopanas, V. and Wangler, B.: 1991, Integrating database technology, rule-based systems and temporal reasoning for effective software: the TEMPORA paradigm, *Journal of Information Systems.*

McBrien, P., Niezette, M., Pantazis, S., Thedoulidis, B., Tziallas, G., Seltveit, A., Sundin, U. and Wohed, R.: 1991, The TEMPORA external rule language, *Proceedings of the Third Nordic Conference on Advanced Information Systems Engineering*, Vol. 498 of *LNCS*, Springer-Verlag.

McBrien, P., Seltveit, A. and Wangler, B.: 1992, An entity-relationship model extended to describe historical information, *Proceedings of CISMOD '92*, Bangalore, India.

Xiaofeng, L.: 1991, What is so bad about rule-based programming?, *IEEE Software* **8**(5), 103–105.

Yang, M.: 1993, *COMIS - A Conceptual Model for Information Systems*, PhD thesis, IDT, NTH, Trondheim, Norway.

BIOGRAPHIES

Peter McBrien received a B.A. in Engineering and Computer Science from Cambridge University in 1986. After working at Racal and ICL, he joined Imperial College as a Research Associate in 1989, where he obtained his PhD in 1992. Leaving Imperial College in January 1994, he spent two months working at The Norwegian Institute of Technology (NTH) as a visiting academic before joining the Department of Computer Science at King's College London as a lecturer in April 1994. His research activities at present cover the area of temporal databases, rule based programming, algebras to support conceptual modelling, and modelling techniques for temporal and rule based systems.

Anne Helga Seltveit obtained her siv.ing. (MSc) degree from NTH in 1987. She then worked as a research assistant in the Informations Systems Group lead by Prof. Sølvberg. She obtained her dr.ing. (PhD) degree in January 1995. In 1988 she had seven months study leave, working with Andersen Consulting in Chicago. Her research activities include work on the process, rule and temporal database modelling languages.