ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

Antony Bryant and Lesley Semmens (Eds)

Methods Integration

Proceedings of the Methods Integration Workshop, Leeds, 25-26 March 1996

Paper:

Integrating Process Modelling and Soft Systems Analysis

D.W. Bustard and P.J. Lundy

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

Integrating Process Modelling and Soft Systems Analysis

David W. Bustard

School of Information and Software Engineering, University of Ulster at Coleraine
Coleraine, BT52 1SA
dw.bustard@ulst.ac.uk

Patrick J. Lundy

School of Information and Software Engineering, University of Ulster at Jordanstown
Newtownabbey, BT37 0QB
pj.lundy@ulst.ac.uk

Abstract

This paper describes and reflects on the experience of attempting to integrate process-oriented formal modelling with activity modelling in soft systems analysis. The work is part of a larger project at the University of Ulster to develop RACE, a new general purpose requirements engineering method for computing systems. Two distinctive characteristics of the integration approach used are that (i) formal modelling is offered as an optional facility, and so must be attractive; and (ii) the RACE method has been tailored to aid integration, in support of (i). The paper explains the relevant parts of the RACE method and the progress made so far to link its activity models with formal process models in LOTOS. The benefits and limitations of this work are discussed and proposals for an improved, tighter integration outlined.

1 Introduction

The development of the RACE (Requirements Acquisition and Controlled Evolution) Method [1] at the University of Ulster is an attempt to bring together a number of separate strands of earlier research and provide a coherent framework for future work. The method is evolving from a number of basic ‘beliefs’ about the desirable characteristics of a good requirements engineering method. One such belief is that “*RACE should support the use of formal description techniques*”. This view is based on the simple but compelling argument that formality is no more than a means of producing precise descriptions about system properties – descriptions that are necessary, in general, to ensure that everyone involved is clear about what is being attempted and that what is achieved matches what is specified. In practice, of course, it has proven difficult to obtain such benefits at reasonable cost [2] but there are various approaches that can be taken and not all are expensive [3].

Essentially, there are two main strategies to consider:

1. using formal descriptions in a *central construction role*, refining specifications from an initial abstract form down into an implementation; or
2. using formal descriptions in a *subsidiary support role*, producing formal descriptions to enhance and help analyse less formal descriptions.

The latter choice is cheaper to implement, which suggests that it should be the initial step even if a more central role is envisaged as the ultimate goal. RACE is certainly starting in this way. However, the underlying concern is to produce a good requirements engineering method irrespective of the individual techniques used and this will determine the approach ultimately adopted.

A subsidiary support role for formal techniques implies the existence of a basic ‘informal’ method that is enhanced with facilities for formal modelling. The enhancement might be achieved by an interleaving that makes formal modelling compulsory. However, in RACE, the decision has been made to have the use of formal techniques optional which implies providing for integration in a way that is attractive to developers. Optional use also implies that such formal modelling should not, strictly speaking, advance the analysis process but rather serve as a reinforcement of the development phases concerned.

The remainder of this paper describes and assesses the method integration work attempted so far in RACE. The first section summarises the relevant parts of RACE and indicates how the RACE method has been partly designed to help support process-oriented formal modelling. RACE has been built around Checkland and Wilson’s Soft Systems Methodology [4, 5, 6], which involves the development of activity models, so there is already an obvious link to process modelling which is largely concerned with describing behaviour. The second section of the paper then

explains a loosely coupled relationship that has been developed between soft systems models and corresponding formal models in LOTOS [7] (Language of Temporal Ordering Specification). The final section analyses the strengths and limitations of this approach, concluding with a discussion of how a tighter integration of the methods can be achieved.

2 Activity Models in RACE

RACE assumes that requirements analysis for computing systems involves two phases of investigation: (i) a *business analysis*, which establishes the business case for a computing system; and (ii) a *computing-oriented analysis*, which identifies how the computing system will operate in support of the business. The business analysis phase is based on Checkland and Wilson's Soft Systems Methodology (SSM) [4, 5, 6]. SSM helps to identify the purpose or purposes of a business and the activities necessary to achieve those purposes. The business is described in system terms using various models. These are described collectively as *activity models* in RACE. Typically, the models are first constructed for each distinct business perspective and then merged to form an overall system description. The analysis of such models in relation to the current operation of the business gives the basis for determining improvements.

Overall, the business analysis phase of the RACE method involves four stages of development as summarised in Figure 1.

Figure 1: RACE Analysis Stages

Activity models are built and refined in the second and third stages of business analysis. Each RACE activity model has three related elements: (i) a *root definition*, (ii) a *conceptual model*, and (iii) an *interaction model*. The root definition gives a succinct textual description of the characteristics of a system. For example, in the case of a simple cash-point banking system it might take the following form:

A Big Bank owned system to meet customer requirements for a convenient supply of cash by facilitating cash withdrawals from accounts, subject to the constraints that customers do not exceed defined credit limits and that unauthorised access is prevented.

A conceptual model then defines the activities required by the root definition and indicates, informally, relationships among activities. A conceptual model is derived directly from a root definition. For example, Figure 2 shows a simplified conceptual model for the Big Bank system, based on the above root definition.

Four of the necessary activities are identified: (i) opening an account; (ii) accepting cash deposits into the account; (iii) supplying cash from the account; and (iv) monitoring that the system achieves its purpose. Typically, top-level activities are elaborated as sets of lower-level activities, giving a hierarchical model definition. Often a full model will contain hundreds of individual activities.

As part of stage three of the business analysis phase of RACE a comparison is made between the conceptual models and the operation of the business to test the models for adequacy and identify deficiencies in the business. This increases the understanding of the analyst and will usually lead to revisions of the models. At the end of this iteration process the models will reflect how the business *should* operate.

Next in stage three, the conceptual models are enhanced to define more precisely how each activity is performed in terms of what it takes in and what it produces. The resulting descriptions are known as *interaction models* because they focus on the interaction of system elements. They are constructed directly and systematically

from conceptual models by defining the behaviour of each activity in terms of how it interacts with other system elements. New descriptive elements are introduced as necessary to account for each source of input and destination of output. For example, the activity 'open account' will need details from a 'customer' and those details will have to be stored in an 'account file' for subsequent reference. By convention, a new source or destination that is perceived to have an 'active' role, such as the 'customer', is called an *entity* while an element with a 'passive' role, such as the 'account file', is a *store*. Entities, stores and activities are known collectively as *processes*.

Figure 2: (Simplified) Conceptual Model for the Big Bank System

An interaction model is described in tabular form as suggested by the following partial description for the banking example:

A1. Open account: Obtain *customer details* (I1) from the *customer*, set up a new account using those details (I2), and issue the customer with an *account number* (I3).

Interaction	Linked Process	Data
I1 Obtain customer details	E1 Customer	D1 Customer details (IN)
I2 Initialise account	S1 Account file	D1 Customer details (OUT) D2 Account number (IN)
I3 Issue account number	E1 Customer	D2 Account number (OUT)

S1. Account file Respond to a request to open a new account (I2); record a cash deposit (I4); record a cash withdrawal (I5);.

Interaction	Linked Process	Data
I2 Initialise account	A1 Open account	D1 Customer details (IN) D2 Account number (OUT)
I4 Note deposit	A2 Accept deposit	D2 Account number (IN) D3 Cash amount (IN)
I5 Note withdrawal	A3 Supply cash	D2 Account number (IN) D3 Cash amount (IN)

These tables are a variant of the type of table used by Wilson to analyse information systems [5]. The behaviour of each process is expressed as a narrative, identifying individual interactions and the data associated with those interactions. The 'open account' activity, for example, involves three interactions: (i) an interaction with the 'customer' to obtain personal details, such as name and address, as needed to open an account; (ii) an interaction with the 'account file' to set up a new account with the customer's details and allocate a customer account number; and (iii) an interaction with the 'customer' to issue the account number. Note, of course, this description has been simplified in various ways to avoid clutter in the discussion. For example, it ignores the possibility that the request to open an account or withdraw cash may be rejected.

The system elements of an interaction model can also be shown as an *interaction diagram*, as illustrated in Figure 3. This is derived directly from Figure 1. A representation for the 'customer', the 'account file' and a 'cash stock' have been added and the arcs connecting the system elements now indicate the presence of one or more interactions between the processes concerned. The monitoring activity is not shown connected to other processes but might be based, for example, on a customer survey.

When the individual interaction models have been completed they are merged to produce an overall system model, which is then used as the basis for making recommendations for change.

This process is effective when used in the way described. However, it is argued that it can be further improved with the help of formal modelling, in that formal models can be checked for consistency and their meaning examined more thoroughly. This possibility is described in the next section in terms of an integration between the basic RACE business analysis process and formal modelling in LOTOS [7], a process-oriented formal description language.

Figure 3: Interaction Model for the Big Bank System

3 Formal Model Integration

The interaction models of RACE provide a bridge between conceptual models and computing-oriented descriptions such as, for example, data-flow diagrams [8] and object models [9]. However, they were also designed to support the development of formal process-oriented models such as CCS [10] and CSP [11] and were particularly influenced by the descriptive facilities of LOTOS.

Formal modelling facilities have been developed as an optional facility, with their intended role summarised in Figure 4.

Figure 4: The Role of Formal Modelling in RACE Model Development

The basic process is effectively one of successive model refinement, from root definitions through to computing oriented models. It is only at the interaction model stage that the levels of description are sufficiently

precise to make formal modelling worthwhile. The main purpose of such modelling, as indicated in the diagram, is to help validate the interaction model but also to assist in the development of the models that follow.

LOTOS was chosen partly because it has a standardised definition [12] and partly because it was designed to facilitate tool support. It combines two descriptive techniques:

- a *process algebra* formalism, based on CCS [10] and CSP [11], used to describe the temporal behaviour of a system; and
- an *abstract data type* formalism, based on ACT ONE [13], used to specify the data in a system and the operations on that data.

In general, a system is described in LOTOS as a hierarchy of nested process and type definitions. The behaviour of a process is defined by a *behaviour expression*. This is a combination of *atomic events* (or actions) and the instantiation of processes, linked using sequential, parallel and choice operators. Processes interact by sharing events and this may involve the interchange of data. The events through which a process can interact are declared as formal parameters in its definition – *event gates*; when a process is instantiated corresponding actual parameters are given. In a similar way data values can be passed to processes through parameters. A process can be recursively instantiated to specify repeated behaviour.

Because formal modelling is optional its use must be cost-effective if it is to achieve user acceptance. In effect, this means making the construction, analysis, and modification of formal models as simple as possible. An earlier paper described a largely automated route from interaction models to equivalent LOTOS models [14]. The transformation was based on three simplifying assumptions:

1. each process (activity, entity or store) of an interaction model can be described by a matching LOTOS process, giving a one-to-one relationship between the informal and formal processes;
2. each interaction between any pair of informal processes is an atomic ‘event’ and so can be represented by a LOTOS event gate; and
3. every process in an interaction model is considered ‘active’ at all times enabling it to be represented by a LOTOS process that is instantiated initially and continues indefinitely.

With these assumptions a skeleton for a LOTOS specification can be produced directly from an interaction model, mirroring the hierarchical structure of that model. In general, the derived LOTOS specification takes the following form:

```

specification system_name: noexit
    (i) definitions for each data type
behaviour
    (ii) a behaviour expression linking and instantiating the LOTOS processes
        representing the top level processes of the interaction model
where
    (iii) a set of LOTOS definitions for the processes identified in the behaviour
        expression
endspec

```

There are three parts to the description. The first is a set of definitions for the data types named in the interaction model. The second is a behaviour expression that connects and instantiates the processes identified at the first resolution level of the interaction model. The third is a set of LOTOS process skeletons for all of the processes in the interaction model. These are defined hierarchically to follow the structure of the interaction model.

The form of the process skeleton is illustrated in the following example, for the ‘open account’ activity in the banking model:

```

process open_account [obtain_customer_details, set_up_account, issue_account_number]:
noexit :=
    ...
    open_account [obtain_customer_details, set_up_account, issue_account_number]
endproc

```

Here each interaction with ‘open account’ is represented by an *event gate* in the process parameter list. By default, a recursive instantiation of the process is provided at the end of the LOTOS skeleton to represent infinitely repeating behaviour.

Each of the three parts of the description are then completed by the analyst using the information provided informally in the interaction model. Once complete the resulting LOTOS specification is then examined using an

animation tool such as [15, 16] and any flaws encountered rectified by modifying the interaction and/or LOTOS models accordingly.

This integration technique brings a number of practical benefits, including:

- the LOTOS models can be constructed quickly because the task is largely one of ‘filling in the blanks’;
- the close relationship between the LOTOS and interaction models simplifies the task of relating the animated behaviour in LOTOS to the matching behaviour described for the interaction model – indeed the animation could be presented through an interaction diagram in a similar way to that described in [16]; and
- the close relationship between descriptions also simplifies the task of mirroring changes when the models have to be modified.

This technique is certainly adequate for the initial construction of a formal model but it has become clear that a tighter linkage between the interaction and formal models is needed to support the modification of models – either during initial model development or later during maintenance. An approach to tackling this problem is described in the next section.

4 Tightening the Link

What has been achieved so far can be assessed by evaluating it against a concept of what is ideal. In this case ‘ideal’ means a situation in which the formal model is created without any analyst intervention whatsoever. In reality this is a practical impossibility because the informal description cannot be analysed automatically. However, it does serve to clarify that the main concern is to minimise the input of the analyst in developing formal models.

There are actually two issues here: one is to reduce, as far as possible, the work that the analyst is required to perform and the other is to dissuade the analyst from developing the formal model beyond what is given informally in the interaction model. In practice, with direct access to the formal description, there is a tendency for the analyst to include extra information, especially when describing data types. To address both issues a revised integration scheme has been developed. This is summarised in Figure 5.

Figure 5: Revised Integration Scheme

In this scheme the LOTOS model is constructed automatically from a combination of the interaction model and *supplementary details* supplied by the analyst. The analyst has no direct access to the LOTOS description and if changes are required they must be effected through changes to the supplementary details and/or the interaction model. In this way the analyst is encouraged to think about the LOTOS model as a *derived* view rather than a co-specification – a relationship similar to that between a high level language and the machine code into which it is compiled. The problem is then reduced to developing an effective means of providing the supplementary details.

There are four inter-dependent pieces of information that the analyst needs to provide:

1. data type information;
2. process behaviour definitions;
3. process value parameter definitions (data types defining the process state); and
4. values used in process instantiation.

An interaction model identifies data but does not make any direct attempt to define it. The main concern is to describe behaviour and an evaluation of the interaction model means assessing the validity of that behaviour. In principle, therefore, the meaning of data operations need not be defined and so the construction of explanatory axioms need not be supported. However, a basic requirement is to provide sufficient information to enable the resulting LOTOS description to be animated so that it may be analysed. Some animators do indeed use axioms during

execution to determine the results of operations, but this work will use other techniques, such as prompting the user for a value.

One approach would be to define a separate LOTOS type definition for each distinct data type named in the interaction model. The necessary information might be supplied through a descriptor of the form shown in Figure 6, illustrated with the ‘cash amount’ data type from the banking example.

Type cash_amount		
Operations	Inputs	Results
zero		cash_amount
deposit	cash_amount, cash_amount	cash_amount
withdraw	cash_amount, cash_amount	cash_amount

Figure 6: Data Type Descriptor

The signature for each data operation is defined as a function naming the types of its inputs, if any, and its result. In the case of ‘deposit’ and ‘withdraw’, the inputs are the account capital and the transaction amount, and the result is the revised capital. The nullary operation ‘zero’ returns a zero value cash amount.

The corresponding LOTOS definition for this ‘cash amount’ description would be:

```

type cash_amount is
sorts cash_amount
opns (* operations *)
    zero: -> cash_amount
    deposit: cash_amount, cash_amount -> cash_amount
    withdraw: cash_amount, cash_amount -> cash_amount
eqns (* equations - note none defined *)
endtype (* cash_amount *)

```

Similar definitions would be produced for each of the data types identified in the interaction model.

Although this provides a simple one-to-one relationship between RACE and LOTOS data types, an alternative approach is simply to generate a single LOTOS definition covering all of the types. This would combine the sorts and operations on these sorts, thus:

```

type data_types is
sorts cash_amount, customer_details, account_number, ...
opns (* operations *)
    zero: -> cash_amount
    deposit: cash_amount, cash_amount -> cash_amount
    withdraw: cash_amount, cash_amount -> cash_amount
    ...
    validate_account: account_number -> Boolean
    ...
eqns (* equations - note none defined *)
endtype (* cash_amount *)

```

For processes it is better to maintain a strict one-to-one relationship between the RACE and LOTOS definitions. Figure 7 shows the type of information that might be supplied to help build the LOTOS definition, using the ‘cash stock’ process as an example.

In general, the descriptor identifies:

- the name of the process;
- the data types and the particular operators and values used in the process definition;

- the state of the process (if any), represented by a sequence of zero or more data values; in LOTOS terms these are parameters to the process and the value shown in brackets after each data type is the instantiation value; and
- the behaviour description; this is largely standard LOTOS, simplified slightly by removing the event gates from recursive instantiations of the process.

Process	cash_stock
Uses	cash_amount (zero, deposit, withdraw)
State	stock: cash_amount (zero)
Events	deposit_cash, withdraw_cash
Behaviour	<pre> (deposit_cash?c:cash_amount; cash_stock (deposit (c, stock))) [] (withdraw_cash?c:cash_amount; cash_stock (withdraw (c, stock))) </pre>

Figure 7: Process Descriptor

Process descriptors thus provide the remaining information needed to construct process definitions and the behaviour expressions that instantiate and link the processes. The descriptor in Figure 7, for example, can be used to build the following LOTOS definition for ‘cash stock’.

```

process cash_stock [deposit_cash, withdraw_cash] (stock: cash_amount) : noexit :=
    (deposit_cash?c:cash_amount;
      cash_stock [deposit_cash, withdraw_cash] (deposit (c, stock)))
    []
    (withdraw_cash?c:cash_amount;
      cash_stock [deposit_cash, withdraw_cash] (withdraw (c, stock)))
endproc

```

5 Conclusion

This paper has described an integration between formal modelling in LOTOS and the use of less formal descriptions of behaviour in soft systems activity models. The approach taken uses LOTOS in an optional support role to help clarify the meaning of the informal models and to explore their implications. The integration goal has been to simplify, as far as possible, the task of constructing and maintaining the formal descriptions. This has involved developing ways of producing LOTOS skeletons directly from interaction models. Recent work has used the technique of defining and managing ‘supplementary details’ for an interaction model that enable a full LOTOS description to be produced without further analyst intervention. This has made the link between the models explicit and therefore clearly identifies the cost of achieving formality in this case – namely the cost of providing and maintaining the supplementary information. It also identifies where effort should be placed in the future to further reduce costs.

The descriptor approach, in effect, simplifies the LOTOS notation and so also makes the initial construction of LOTOS descriptions easier, especially for those new to the notation. Further simplifications seem possible, including the use of a graphical representation for behaviour, and these opportunities for tighter integration will be explored.

6 Acknowledgements

The authors are grateful to Raymond Oakes for his advice on the construction and presentation of the soft systems models for the banking example given in this paper. The research described was undertaken as part of the RUNES project (Requirements Understanding, Negotiation, Expression and Scrutiny), funded by the University of Ulster. RUNES is broadly concerned with investigating techniques for improving the requirements engineering process and with the development of tools to support that process. The RACE Method has emerged from this work.

7 References

1. Bustard DW. Progress Towards RACE, a 'Soft-Centred' Requirements Definition Method. In: Proceedings of 1st IFIP/SQI International Conference on Software Quality and Productivity, Hong Kong. Chapman & Hall, 1994, pp 29-36
2. Bowen JP, Stavridou V. Safety-critical Systems, Formal Methods And Standards. *Software Engineering Journal* 1993; 8 (4): 189-209
3. Lundy PJ, Bustard DW. Making Formal Methods Work: What's Being Done and What Can be Done. In: Proceedings of 2nd International Conference on Software Quality Management, Edinburgh. Computational Mechanics, 1994, pp 485-499
4. Checkland PB. *System Thinking, System Practice*, Wiley, 1981
5. Wilson B. *Systems: Concepts, Methodologies and Applications*, 2nd Edition, Wiley, 1990
6. Checkland PB, Scholes J. *Soft Systems Methodology in Action*, Wiley, 1990
7. Turner KJ (Ed). *Using Formal Description Techniques*, Wiley, 1993
8. Bustard DW, Oakes R, Heslin E. Support for the Integrated Use of Conceptual and Dataflow Models in Requirements Specification. In: Proceedings of Colloquium on Requirements for Software Intensive Systems, DRA Malvern. 1993, pp 37-44
9. Bustard DW, Dobbin TJ, Carey BN. Integrating Soft Systems and Object-Oriented Analysis. In: Proceedings of 2nd International Conference on Requirements Engineering, Colorado Springs, IEEE Computer Society Press, 1996, pp 52-59
10. Milner R. *A Calculus of Communicating Systems*, Springer-Verlag, 1980 (Lecture Notes in Computer Science No. 9)
11. Hoare CAR. *Communicating Sequential Processes*, Prentice Hall International, 1985
12. ISO - Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, ISO 8807, 1989.
13. Erhig H and Mahr B. *Fundamentals of Algebraic Specification 1*, Springer-Verlag, Berlin, 1985
14. Bustard DW, Lundy PJ. Enhancing Soft Systems with Formal Modelling. In: Proceedings of the International Symposium on Requirements Engineering, York. IEEE Computer Society Press, 1995, pp 164-171
15. van Eijk PHJ. The Lotosphere Integrated Tool Environment LITE. In: *Formal Description Techniques IV*, Parker K and Rose G (Eds.), North-Holland, 1991, pp 473-476
16. Winstanley A, Bustard DW: EXPOSE. An Animation Tool for Process-Oriented Specifications. *Software Engineering Journal* 1991, 6 (6), pp 463-475