Department of Computer Science
University of Manchester
Manchester M13 9PL, England

Technical Report Series

UMCS–93–6–1

Jin Sa and Brian C. Warboys

A Formal Description of the ISPW-6 Software Process Example

A Formal Description of the ISPW-6 Software Process Example*

Jin Sa and Brian C. Warboys

Department of Computer Science
University of Manchester
Oxford Road, Manchester, UK.
jsa@cs.man.ac.uk brian@cs.man.ac.uk

Technical reports issued by the Department of Computer Science, Manchester University, are available by anonymous ftp from `m1.cs.man.ac.uk` (130.88.13.4) in the directory `/pub/TR`. The files are stored as PostScript, in compressed form, with the report number as filename. Alternatively, reports are available by post from The Computer Library, Department of Computer Science, The University, Oxford Road, Manchester M13 9PL, UK.

Abstract

Recently, software process modelling has attracted an increasing research interest. For the purpose of comparing and contrasting different approaches to software process modelling, a common problem, referred to as the ISPW-6 software process example, was defined. Many solutions to this example have been provided by different people using different approaches.

In this report, a solution to the ISPW-6 software process example by an approach, called BM, is presented. The suitability of BM for software process modelling is assessed against the criteria proposed at the sixth International Software Process Workshop.

Keywords: The Software Process, Formal Methods

1 Introduction

There has been an increasing research interest in software process modelling. Many different approaches have been developed in this field, for example, PML [25], Adele2 [3], SPELL [6], PROCESS WEAVER [9], AP5 [11], STATEMATE [14] and SLANG [2]. It was thought that in order to compare and contrast the different approaches, it would be useful if a common problem were used to demonstrate the different approaches. Therefore an example was defined for such purpose at the sixth International Software Process Workshop. This example was later referred to as the ISPW-6 software process example. We shall abbreviate it as the ISPW-6 example in the rest of the report.

In this report we shall model the ISPW-6 example using a method called BM which was developed as a specification method for concurrent object-based systems at the University of Manchester. Details of BM are described in [22]. An initial study of the feasibility of the BM application to process modelling was carried out and reported in [23]. This report further investigates the suitability of BM to software process modelling.

In this report, instead of simply providing a BM description of the ISPW-6 example, a method for modelling the example in BM is first of all defined. This method is then used to produce a BM description of the ISPW-6 example in a systematic way. We feel that it is beneficial to have such a method so that it is easy to re-model the example as it is inevitable that it will be (and has been) changed and modified.

The rest of this report is organised as follows. Section 2 provides a brief description of the ISPW-6 example and Section 3 contains an overview of BM. The essential elements in the ISPW-6 example are listed in Section 4 and in Section 5 corresponding BM notations for these elements are identified. A method for modelling the ISPW-6 example in BM is defined in Section 6. Section 7 contains the BM description of part of the ISPW-6 example. Some related works are discussed in Section 8. In Section 9 some issues in the ISPW-7 extensions are briefly addressed. Finally the conclusion is drawn in Section 10.

2 Brief Description of the ISPW-6 Example

The core problem focuses on the designing, coding, unit testing, and management of a rather localized change to a software system. This is prompted by a change in requirements, and can be thought of as occurring either late in the development phase or during the support phase of the life-cycle. In order to confine the problem, it is assumed that the proposed requirements change has already been analyzed and approved by the configuration control board and only a single code unit is to be affected. The problem begins with the project manager scheduling the change and assigning the work to appropriate staff. The example problem ends when the new version of the code has successfully passed the new unit tests. During the software process, the following steps may be taken by various personnel: Schedule and Assign Tasks, Modify Design, Review Design, Modify Code, Modify Test Plans, Modify Unit Test Package, Test Unit and Monitor Progress. A full description of the example is given in [16].

3 Overview of BM

BM [21] is a specification method aimed at the specification of concurrent object-based systems.

3.1 General Concepts

In BM, a system is considered to be composed of components which may be executed in parallel. Each component provides a number of operations and contains some state variables which can only be accessed by the operations. An operation may call other operations in other components. There are two kinds of operations: active and passive. An active operation of a component is started by the component itself, and a passive operation is only started when it is being called by another operation in another component (or a person). The behaviour of a component is specified by the pattern to which its operations, when executed, conform. Such patterns are called **operation patterns**. The execution of a component is as follows: if the next allowed operation in the **operation pattern** is an active one, it can be executed without waiting for any external trigger; if it is a passive one, the component must wait until it is being called by another operation in another component (or a person). A component may only communicate with another component by calling the operations of the other component.

3.2 Underlying Model

The semantics of BM constructs are defined using a linear time temporal logic. A brief overview of the underlying model is described below. A detailed definition is given in [21].

The behaviour of a component is defined as the set of all the possible execution sequences of the component. An execution sequence of a component is a sequential combination of its operation sequences. An operation sequence is a sequence of events. There are four kinds of events:

- `send` a request to call an operation;
- `receive` a result (or end of operation indication) from the called operation;
- `accept` a call request;
- `return` a result (or end of operation indication) to the caller.

The sequence of an active operation may contain a list of `send` and `receive` events. In addition to a list of `send` and `receive` events, the sequence of a passive operation also contains an `accept` event and a `return` event at the starting and finishing points respectively.

The model matches `send`s to `accept`s and `receive`s to `return`s when composing the full specifications of a system. The behaviour of a system is defined as the set of all possible execution sequences of the system. Each step of an execution sequence of a system, one consisting of several components, is interleaved from execution sequences of the components unless there is a synchronisation between some of the components. Two components are synchronised when one is calling the other. In the case of synchronisation, the corresponding events from the two components, e.g. `send` and `accept`, must take place in parallel.

3.3 Basic Constructs

To specify a system, BM provides two constructs - a single object and a composed object.

A single object is used to describe the behaviour of a component by specifying:

- the behaviour of the possible (sequential) combinations of the operations, i.e. the operation pattern;
- the behaviour of each operation provided by the component; the specification of an operation consists of three parts;
  - *call templates*: specifying the interaction events between this operation and other components,
  - call pattern: specifying the orders of these interaction events,
  - pre- and post-conditions: specifying the functionality of the operation.

A composed object is used to describe the behaviour of a collection of components (executing in parallel) by specifying:

- the behaviour of each component (as described above);
- and how the components may interact with each other, this is specified using a BM notation called *synchronisation template*.

4 Elements in the ISPW-6 Example

The ISPW-6 Example consists of the following elements.

Agents: These are the personnel, e.g. project manager, design engineers, QA engineers.

Steps: These are the actions performed by the agents, e.g. modify design, review design. Each agent may perform several steps and a step may be performed by several agents.

Inputs/Outputs: These are the data required and produced by each step, e.g. current design, design review feedback and modified design.

- The format of Input is:
  Data(where from)(communication means)
  where from may have the following possibilities:
  - file(s) and paper(s), e.g. the software design document,
  - agent(s), e.g. design engineer,
  - step(s), e.g. review design.
- The format of Output is
  Data(where to)(communication means)
  where to may have the same possibilities as **where from**.

(In this report, the **communication means** is not modelled.)

Constraints: these describe the allowed sequencing of the steps.

A general scenario is that when the constraints are fulfiled and the inputs are available, an agent takes some inputs to perform a step and produces some outputs. Each agent may perform several steps and each step may be performed by several agents.

5 BM Representation of the ISPW-6 Elements

Each agent is modelled as a single object. Such an object will be referred to as an agent object in the rest of the report. Each step is also modelled as a single object, such an object will be referred to as a step object in this report. An agent object is said to take up a particular role if it accesses a particular step object. An agent object may access several different step objects, and vice-versa, a step object may be accessed by several different agent objects. The former allows an agent to take up different roles and the latter allows team work. The whole ISPW-6 example is modelled as a composed object containing all the agent objects and the step objects.

5.1 Agent Object

An agent object may be engaged in three kinds of activities: it may deliver artifacts[1] to some steps; it may receive artifacts from some steps; it will perform some steps. The following three kinds of operations are used to model these activities:

supply operation: delivers artifacts to a step by calling the receive operation of the appropriate step object. A supply operation of an agent object is modelled as a passive operation. It is invoked by an agent.

receive operation: receives artifacts from a step. These operations are called by the supply operations of the appropriate step or agent objects. Receive operations are modelled as passive operations.

step interface operation: calls the corresponding step object to perform (part of) the step. Step interface operations are invoked by the relevant agents. They are also passive operations.

In addition to the above three kinds of operations, each agent object should also have an operation pattern which defines the constraints on the ordering of the operations for that agent. There is no general common constraint for agent objects.

5.2 Step Object

In order to perform a step, a step object usually needs to get some artifacts (inputs) from either some agent objects or some other step objects or both. On completion of a step, some artifacts (outputs) may be delivered to either some agent objects or some other step objects or both. Therefore step objects also need to provide three kinds of operations:

receive operation: receives the input data from either an agent or a step object. These operations are called by the supply operations of the appropriate agent or step objects. Receive operations are passive operations.

supply operation: delivers the output data to either an agent or a step object. These operations call the relevant agent or step object. A supply operation of a step object is active because the operation is started by the step object itself when it is ready to deliver data.

step operation: performs the step. This operation is called by the relevant agent objects. Step operations are passive operations.

As in an agent object, in addition to the operations, each step object should have an operation pattern. Usually the operation pattern for a step object consists of a few receive operations, followed by, possibly an iteration of, the step operation and followed by a few supply operations.

[1] The term *artifact* is used in accordance with the definitions given in [8] and [18]. *Artifacts* mean both input and output data for steps.

5.3 Input and Output flows

In the ISPW-6 example, the scopes of the data are not explicitly described. In this document, we assume that files and reports are available to their step objects only, therefore they are modelled as state variables of the step objects. (Data may be modelled as objects if they are globally available.)

- Data from file(s) and paper(s):

  The data is obtained by reading from the state variable(s).

- Data from another agent or step:

  The data is obtained through the appropriate receive operation.

- Data to file(s) and paper(s):

  The data is delivered by writing to the state variable(s).

- Data to another agent or step:

  The data is delivered to the agent or step object by calling the appropriate receive operation, through the relevant supply operation, with the data as the parameter(s).

5.4 The Entire Example

The entire ISPW-6 example is modelled as a composed object. The interactions among the agent objects and the step objects are defined using *synchronisation templates*. For each pair of objects A and S, if A may call operations of S, the *synchronisation template* for the two objects lists the names of the operations that are provided by S and potentially called by A.

5.5 Relationships Between the Different Objects

To summarise:

- The supply operations of agent objects call the receive operations of step objects and/or agent objects.
- The step interface operations of agent objects call the step operations of step objects.
- The supply operations of step objects call the agent and/or step objects receive operations.
- The agents, i.e. the project participants, invoke the operations provided by the relevant agent objects.

5.6 Rationales

This section explains the reasons for the BM approach.

5.6.1 Agent Objects

In many approaches agents are modelled as parameters or attributes of the steps. It was decided that agents should be modelled as objects for four reasons. Firstly agents perform activities. It seems a natural way to model an agent as an object which performs some steps. Secondly information (or artifacts) may be sent to or received from agents. This can be easily modelled with objects. Thirdly each agent may perform a variety of steps, again this can be naturally modelled by having the agent object calling the corresponding step objects. Fourthly an agent may perform only part of a step. By defining an operation, in the appropriate step object, which only performs part of the step, such behaviour can be modelled by having the agent calling the (partial step) operation.

5.6.2 Step Objects

Each agent performs some steps. If an agent is modelled as an object, it seems that steps should be modelled as operations of the agent performing them. However problems occur when a step is performed by several agents, in which case the step should be available to all the agents performing it. This consideration leads to the current approach in which a step object is associated with every step. The step is modelled as a step operation of the step object. Thus a step may be performed by more than one agent.

5.6.3 Supply and Receive Operations

Each step takes some input data. Since each step is modelled as an operation (or operations in order to allow partial execution of a step), it may seem to be natural to treat the input data as parameters of the operation. However there are two problems with this. Firstly the input data may come from an agent (or a step) different from the one performing the step, and secondly the data may need to come from several agents (and/or steps) for a step. In the approach presented in this report a receive operation is defined for each input required by a step and a supply operation is defined for each agent (or step) that supplies the input data for the step. The input data is then received only through the supply operation of the agent (or step) supplying the input data.

6 A Method for Modelling the ISPW-6 Example in BM

In the previous section, we identified the BM notations corresponding to the elements in the ISPW-6 example. In this section we describe a method which enables us to model the ISPW-6 example in a systematic way. The method is as follows:

1. Identify the agent objects.
2. Identify the step objects.
3. For each agent object:
   - identify the supply operations
     one supply operation for each step to which the agent delivers an artifact
   - identify the receive operations
     one receive operation for each step from which artifacts are received
   - identify the step interface operations
     one step interface operation for each step the agent performs
   - define the operation pattern for the agent object
   - define each operation
4. For each step object:
   - identify the receive operations
     one receive operation for each step or agent from which input data is received
   - identify the supply operations
     one supply operation for each step or agent to which output data is delivered
   - name the step operation
   - define the operation pattern for the step object
   - define each operation
5. Define the entire ISPW-6 example as a composed object.

   For each pair of objects that communicate with each other a *synchronisation template* is defined.

7 BM Description of (part of) the ISPW-6 Example

In this section we will use the method defined in the previous section to model a small part of the ISPW-6 example. The entire example can be found in Appendix A.

7.1 Identify the Agent Objects

The following table lists all the agent objects in the example.

Agent Name	Agent Object Name
Configuration Control Board	CCB
Project Manager	PM
Design Engineers	$DE_{[i:DEset]}$
QA Engineers	$QAE_{[i:QAEset]}$
Software Engineers	$SWE_{[i:SWEset]}$

The notation $A_{[i:set]}$ denotes an array of $A_{[i]}$s ranging over set. $DEset$, $QAEset$ and $SWEset$ contain the groups of design enginers, QA engineers and software engineers respectively.

7.2 Identify the Step Objects

The following table lists all the step objects.

Step Name	Step Object Name
Schedule and Assign Tasks	SAT
Modify Design	MD
Review Design	RD
Modify Code	MC
Modify Test Plans	MTP
Modify Unit Test Package	$MUTP$
Test Unit	TU
Monitor Progress	MP

7.3 Description for Each Agent Object

For simplicity, we shall only model two agent objects: the Configuration Control Board and the Project Manager. The specifications of the other agents can be found in Appendix A.

Configuration Control Board—*CCB*

- identify the supply operations

  The following table lists the supply operations.

Description	Operation Name
requirement change to step object SAT	$reqtoSAT$
authorization to step object SAT	$authtoSAT$
Notification of cancellation to PM	$cantoPM$
cancellation to MP	$cantoMP$

- identify the receive operations

  None

- identify the step interface operations

  None

- define the operation pattern

  The description of the ISPW-6 example given in [16] does not mention the constraints on the operations of *CCB*. Therefore the operation pattern is not defined here.

- define each operation

  The precise definitions of the three supply operations are not significantly relevant to the objectives of this report, and therefore they are not defined here. The main purposes of the three supply operations are to pass the required artifacts to the step object *SAT* and the agent object *PM* by calling the appropriate receive operations provided by *SAT* and *PM*.

Project Manager—*PM*

- identify the supply operations

  None

- identify receive operations

Description	Operation Name
cancellation from CCB	*canfrCCB*

- identify the step interface operations

Step Name	Step Interface Operation Name
SAT	*doSAT*
MP	*doMP*

- define the operation pattern for *PM*

  The operation pattern for *PM* is

  $doSAT; (doSAT \oplus doMP)^{+}; (canfrCCB; fin \oplus fin)$

  In this report, the symbols used in BM are explained as they appear in the example. In the above expression, the symbols denote:

  P;Q specifies the behaviour that satisfies P followed by Q.
  P $\oplus$ Q specifies the behaviour that satisfies either P or Q.
  P^{+}; Q specifies the behaviour that satisfies P a number of times and then satisfies

Q.
fin indicates the end of computation.

The above expression states that after assigning the first task, the project manager may repeatedly assign more tasks (*doSAT*) or monitor the progress of the assigned tasks (*doMP*) until either the end of the project, i.e. the unit testing is successfully completed, or a cancellation is receive from the CCB (*canfrCCB*). The detection of a successful unit testing is further explained later in this section.

- define each operation

  The step interface operation *doSAT* will call the step object *SAT* to perform the Schedule and Assign Tasks step. The other step interface operation *doMP* will call the step object *MP* to monitor the progress.

  Below is the definition of *doMP*. As mentioned before the specification of each operation consists of three parts: *call templates*, call pattern and the pre- and the post-conditions.

  The *call templates* specify what operations are called by this operation. The call pattern defines in what order these operations are called. The pre- and the post-conditions define the functionality of the operation.

  doMP only calls one operation, i.e. *doingMP*. Hence there is only a single *call template*:

calldoingMP	
prop	*function*
obj	*MP*
op	*doingMP*
apar	*someparameters*

  In the above template,

  - *calldoingMP* is the name of the *call template*.
  - `prop` indicates the type of the call. In this case it is of type `function`. The function call returns a result.
  - `obj` identifies the object being called. In this case it is the step object *MP*.
  - `op` identifies the operation within `obj` being invoked. In this case *doingMP* is the name of the operation.
  - `apar` specifies the actual parameters of the call.

  Since there is only one single call, the call pattern is also simple, i.e.

$calldoingMP; fin$

Let *scheduleinfo* and *progressinfo* denote two variables containing scheduling information and progress information respectively, the pre- and the post-conditions of *doMP* are as follows:

$doMP() \quad \triangleq$

pre $\neg complete(progressinfo)$

post $progressinfo' = update(progressinfo, res\text{-}MP\text{-}doingMP) \wedge$
$scheduleinfo' = scheduleinfo$

Where *res-MP-doingMP* is a variable used to record the result received from the call to *doingMP*, *complete* is a predicate stating that the unit testing has been completed successfully and *update* is a function which takes the current progress information and the information received from the call to *doingMP* and returns the updated values of the progress information.

The pre-condition states that the unit testing has not been completed. The post-condition states that the progress information is updated and the scheduling information is not changed.

7.4 Description for Each Step Object

Only Schedule and Assign Tasks, Modified Design and Review Design are modelled here. The other step objects can be modelled similarly.

Schedule and Assign Tasks—*SAT*

- identify the receive operations

Description	**Operation Name**
requirement change from CCB	*reqfrCCB*
authorization from CCB	*authfrCCB*

These receive operations will be called by the corresponding supply operations of *CCB*, i.e. *reqtoSAT* and *authtoSAT*.

- identify the supply operations

Description	Operation Name
notification of assigned tasks to all affected personnel	$notetoPM$
	$notetoDE_{[i:DEset]}$
	$notetoQAE_{[i:QAEset]}$
	$notetoSWE_{[i:SWEset]}$
requirement change to all other steps	$reqtoMD$
	$reqtoRD$
	$reqtoMC$
	$reqtoMTP$
	$reqtoMUTP$
	$reqtoTU$
	$reqtoMP$
Notification of completion of the step to *MP*	$endSATtoMP$

These supply operations will call the corresponding receive operations of the relevant agent or step objects.

- name the step operation

  Let *doingSAT* denote the name of the step operation.

- define the operation pattern for the step object

  The operation pattern for *SAT* is:

$reqfrCCB; authfrCCB;$
$(doingSAT;$
$(info2DesGrp \oplus$
$info2QAGrp \oplus$
$info2CodeGrp \oplus$
$info2TestGrp \oplus$
$info2MonitorGrp))^+;$
$endSATtoMP; fin$

where $info2DesGrp$ denotes $reqtoMD; notetoDE_{[d]}$

$info2QAGrp$ denotes

$reqtoRD; notetoDE_{[d]}; notetoQAE_{[q]}; notetoSWE_{[s1]}; notetoSWE_{[s2]}$

$info2CodeGrp$ denotes $reqtoMC; notetoDE_{[d]}$

$info2TestGrp$ denotes

$reqtoMTP; reqtoMUTP; reqtoTU; notetoDE_{[d]}; notetoQAE_{[q]}$

$info2MonitorGrp$ denotes $reqtoMP; notetoPM$.

where d, q, $s1$ and $s2$ are variables containing the names of the most recently assigned design engineer, QA engineer and software engineers respectively.

The operation pattern of *SAT* states that after receiving the requirement and authorization from CCB, the SAT object is allowed to repeatedly schedule and assign tasks and supply the relevant information to the concerned personnel and step objects.

- define each operation

  Each supply operation will call the corresponding receive operation, so a call template will be defined for it, and the call pattern is simply the call followed by *fin*. The pre- and the post-conditions are all *True* since the supply operations are not engaged in any other activities.

  The receive operations in this example do not call any other operation, hence no call template to define. Again the pre- and the post-conditions are all *True* since the receive operations are not engaged in any other activities.

  The step operation does not call any other operation, again there is no call template to define. It schedules and assigns tasks based on its parameters. The pre- and the post-conditions are defined as follows:

$doMP(inputinfo) \quad \triangleq$

pre $\neg complete(progressinfo)$
post $progressinfo' = progressinfo \ \wedge$
$\quad scheduleinfo' = scheduled(scheduleinfo, inputinfo)$

where *scheduled* is a predicate describing the state of the schedules.

Modify Design—*MD*

- identify the receive operations

Description	Operation Name
requirement change from *SAT*	*reqfrSAT*
design review from *RD*	*revfrRD*

- identify the supply operations

Description	Operation Name
(modified) design to *RD*	*destoRD*
(modified) design to *MC*	*destoMC*
(modified) design to *MUTP*	*destoMUTP*
Notification of completion of the step to *MP*	*endMDtoMP*

- Let *doingMD* denote the name of the step operation.
- The operation pattern for *MD* is

$reqfrSAT; doingMD; destoRD; destoMC; destoMUTP;$
$(endMDtoMP; fin$
$\oplus$
$(revfrRD; doingMD; destoRD; destoMC; destoMUTP)^+;$
$endMDtoMP; fin)$

This step object first of all is allowed to do the design. Upon the completion, the design is sent to *RD*, *MC* and *MUTP*. If the design is approved unconditionally, a notification of the completion of the step is sent to *MP*. However changes may be recommended by *RD*, in which case, this step object will then need to receive the reviewed design from *RD* and repeat the process.

- define each operation

The operations can be defined similarly as those in *SAT*.

Review Design—*RD*

- identify the receive operations

Description	Operation Name
requirement change from *SAT*	*reqfrSAT*
(modified) design from *MD*	*desfrMD*

- identify the supply operations

Description	Operation Name
design review feedback to *MD*	*revtoMD*
outcome info to *MP*	*infotoMP*
Notification of completion of the step to *MP*	*endRDtoMP*

- Let *doingRD* denote the name of the step operation.
- The operation pattern for *RD* is

$reqfrSAT; desfrMD; doingRD;$
$(endRDtoMP; fin$
$\oplus$
$(revtoMD; infotoMP; desfrMD; doingMD)^+; endRDtoMP; fin)$

This operation pattern states that after receiving the design from *MD* (*desfrMD*), the design is reviewed (*doingRD*). If the design is approved, a notification of completion is sent to *MP* (*endRDtoMP*), otherwise the review is sent back to *MD* (*revtoMD*) and the review outcome is also sent to *MP* (*infotoMP*). This step object will then expect to receive the modified design from *MD* (*desfrMD*) and repeat the review process.

7.5 Define the Whole ISPW-6 Example as a Composed Object

Here only a few synchronisation templates are defined. The rest can be defined similarly.

The *synchronisation template* between *CCB* and *SAT* is:

ccb-sat	
caller	*CCB*
callee	*SAT*
ops	reqfrCCB authtoCCB

This *synchronisation template* states that *CCB* and *SAT* are synchronised through the calls on the two operations: *reqfrCCB* and *authfrCCB*.

The *synchronisation template* between *PM* and *SAT* is:

pm-sat	
caller	*PM*
callee	*SAT*
ops	doingSAT

Similarly *PM* and *SAT* are synchronised on the calls to the operation: *doingSAT*.

In BM, all single objects within a composed object may be executed in parallel with each other. The parallelism is modelled in an interleaved fashion unless there is a synchronisation between two objects, in which case the corresponding events, e.g. calling an operation and accepting a call, must take place together. Therefore in the ISPW-6 example, all the agent and step objects may be executed in parallel in an interleaved fashion unless synchronisation is necessary.

8 Other Works

Since the software process example was first defined at the sixth International Software Process Workshop, much work has been done in providing a solution by different people using different approaches. Existing solutions include: AP5 [11], APPL/A [24], EPOS [7], GRAPPLE [13], MASP/DL, Merlin [20], PROCESS WEAVER [5], SEI's approach [15], SLANG [1] and Adele [4].

In some of the above mentioned approaches, agents are modelled as variables, parameters or attributes which can be bound to different steps from time to time. For instance, in EPOS, an agent is modelled as a role attribute, and in PROCESS WEAVER, an agent is treated as an input parameter to a step. In BM agents are modelled as objects. The reasons are explained in subsection 5.6.

Another major difference between this approach and many others is how information flow is modelled. In most approaches, inputs and outputs of steps are treated as parameters and results. In our approach, there is a clear distinction between operations regarding information flow and operations regarding actions. Information flow is explicitly modelled with the supply and receive operations. We believe this gives better control to where the information is going and when the information may be transferred. In the AP5 solution for the ISPW-7 process example, the authors noted that since inputs and outputs of a procedure are typically handled through parameters and return values in their Lisp based language, a problem exists if the results of an activity are not needed by the caller, but by some other activities. In order to get around this problem, they have implemented some subclass of the general class *step* to handle the inputs and outputs. Although not aimed at modelling information flow explicitly, it has a similar effect to the ideas of our supply and receive operations.

9 Possible Solutions to Some Issues in the ISPW-7 Extensions

Although the framework presented in this report was set out to solve the ISPW-6 example, a number of issues defined in the ISPW-7 extensions to the example can be easily modelled within this framework. In this section proposals for solving some of these extended issues are outlined.

9.1 Resource Management

Given a set of durations for each step in the process and a resource database about personnel commitments, the ISPW-7 extension requires to generate a schedule in accordance with the commitment and the duration constraints.

Since agents are modelled as objects rather than as parameters to steps, resource information can be naturally incorporated in the model as follows. In addition to the existing state variables, the agent object representing the process manager should also contain variables recording the duration information for all the step objects and the commitment information for all the agent objects. Tasks are assigned to agents according to this information. Each step object should also contain a variable which records the time left for this step. Each agent object should also contain variables recording commitment information for each step. To ensure that the process satisfies the constraints, each step operation is guarded. For instance, for a step operation in an agent object, the guard should check that the agent still has time left to do this particular step.

9.2 Coordination and Communication

This extension involves coverting the code modification step of the ISPW-6 example from a single person activity to an activity involving parallel modifications and merging of effort.

Since BM is designed for concurrent systems, parallel activities can be easily modelled. Each time a new change is required, the process manager will initiate a set of new objects to do the change and coding. In addition to the existing supply operations, receive operations and step operations, it is also necessary to have some operations dealing with coordination and communication so that information may be passed between different sets of objects.

10 Conclusion

At the 6th International Software Process Workshop, a list of process aspects and issues, which were thought to be important to accommodate when modelling software process, were summarised. The example in this report has illustrated that BM is capable of addressing many of these aspects. Among them the major aspects are: multiple levels of abstraction—i.e. composed object and single object; control flow, sequencing and constraints—e.g. the operation patterns; feedback to earlier steps, iterations—e.g. should review design cause modify design to be repeated, the appropriate supply operations of the review design object will call the corresponding receive operations of the modify design object to pass the feedback, and modify design may be repeated once all its inputs are ready; object/information management and flow through process—i.e. through the use of the supply and receive operations; organisational responsibility for specific tasks—i.e. agent objects are responsible to perform the steps; parallelism—i.e. all objects may be executed in parallel.

Among those aspects that are not addressed, some are not difficult to incorporate, e.g. communication means, others require some further studies. Our future work will concentrate mainly on the issue of process change.

The other aspect of this work that we are currently developing is a simulation facility which allows us to step through the operation patterns of the objects. A prototype version [10] of this stepping through facility has been implemented using Enact [12]. Early experiments with it have confirmed its significance in the construction of a specification in the sense that the modellers are able to gain much more confidence than simply reading the specifications.

An enactment model of the ISPW-6 example has also been implemented in a process modelling language [19]. Rigorous checking of the enactment model against the BM specification has been performed using the stepping through facility [10].

A BM Specification of the ISPW-6 Software Process Example

A.1 Identify the Agent Objects

The following table lists all the agent objects in the example.

Agent Name	**Agent Object Name**
Configuration Control Board	CCB
Project Manager	PM
Design Engineers	$DE_{[i:DEset]}$
QA Engineers	$QAE_{[i:QAEset]}$
Software Engineers	$SWE_{[i:SWEset]}$

The notation $A_{[i:set]}$ denotes an array of $A_{[i]}$s ranging over set. $DEset$, $QAEset$ and $SWEset$ contain the groups of design enginers, QA engineers and software engineers respectively.

A.2 Identify the Step Objects

The following table lists all the step objects.

Step Name	**Step Object Name**
Schedule and Assign Tasks	SAT
Modify Design	MD
Review Design	RD
Modify Code	MC
Modify Test Plans	MTP
Modify Unit Test Package	$MUTP$
Test Unit	TU
Monitor Progress	MP

A.3 Description for Each Agent Object

Configuration Control Board—CCB

- identify the supply operations

  The following table lists the supply operations.

Description	**Operation Name**
requirement change to step object SAT	$reqtoSAT$
authorization to step object SAT	$authtoSAT$
Notification of cancellation to PM	$cantoPM$
cancellation to MP	$cantoMP$

- identify the receive operations

  None

- identify the step interface operations

  None

- define the operation pattern

  Not described in the IPSW-6 problem

- define each operation

Project Manager—PM

- identify the supply operations

  None

- identify receive operations

Description	**Operation Name**
cancellation from CCB	$canfrCCB$

- identify the step interface operations

Step Name	**Step Interface Operation Name**
SAT	$doSAT$
MP	$doMP$

- define the operation pattern for PM

  The operation pattern for PM is

$doSAT; (doSAT \oplus doMP)^+; (canfrCCB; fin \oplus fin)$

- define each operation

  call template of $doMP$:

calldoingMP	
prop	*function*
obj	*MP*
op	*doingMP*
apar	*someparameters*

call pattern of *doMP*:

$calldoingMP; fin$

Let *scheduleinfo* and *progressinfo* denote two variables containing scheduling information and progress information respectively, the pre- and the post-conditions of *doMP* are as follows:

$doMP() \quad \triangleq$

pre $\neg complete(progressinfo)$

post $progressinfo' = update(progressinfo, res\text{-}MP\text{-}doingMP) \ \wedge$
$scheduleinfo' = scheduleinfo$

res-MP-doingMP is a variable which records the values received from the call *calldoingMP*.

Design Engineer—$DE_{[e]}$

- identify the supply operations

  None

- identify the receive operations

Description	**Operation Name**
notification from SAT	*notefrSAT*

- identify the step operations

Step Name	**Step Operation Name**
MD	*doMD*
RD	*doRD*
MC	*doMC*
MTU	*doTU*

- define the oepration pattern for $DE_{[e]}$

$notefrSAT; ((doMD; doRD)^{+}; (doMC; doTU)^{+})^{+}; fin$

QA Engineer—$QAE_{[e]}$

- identify the supply operations

  None

- identify the receive operations

Description	Operation Name
notification from SAT	*notefrSAT*

- identify the step operations

Step Name	Step Operation Name
RD	*doRD*
MTP	*doMTP*
MUTP	*doMUTP*
TU	*doTU*

- define the operation pattern

$notefrSAT; (doRD^+; doMTP; doMUTP; doTU)^+; fin$

- define each operation

A.4 Description for each Step Object

Schedule and Assign Tasks—SAT

- identify the receive operations

Description	Operation Name
requirement change from CCB	*reqfrCCB*
authorization from CCB	*authfrCCB*

  These receive operations will be called by the corresponding supply operations of CCB, i.e. *reqtoSAT* and *authtoSAT*.

- identify the supply operations

Description	**Operation Name**
notification of assigned tasks to all affected personnel	$notetoPM$
	$notetoDE_{[i:DEset]}$
	$notetoQAE_{[i:QAEset]}$
	$notetoSWE_{[i:SWEset]}$
requirement change to all other steps	$reqtoMD$
	$reqtoRD$
	$reqtoMC$
	$reqtoMTP$
	$reqtoMUTP$
	$reqtoTU$
	$reqtoMP$
Notification of completion of the step to *MP*	$endSATtoMP$

These supply operations will call the corresponding receive operations of the relevant agent or step objects.

- name the step operation

  Let $doingSAT$ denote the name of the step operation.

- define the operation pattern for the step object

  The operation pattern for SAT is

$reqfrCCB; authfrCCB;$
$(doingSAT;$
$(info2DesGrp \oplus$
$info2QAGrp \oplus$
$info2CodeGrp \oplus$
$info2TestGrp \oplus$
$info2MonitorGrp);$
$endSATtoMP)^+; fin$

where $info2DesGrp$ denotes

$reqtoMD; notetoDE_{[d]}$

$info2QAGrp$ denotes

$reqtoRD; notetoDE_{[d]}; notetoQAE_{[q]}; notetoSWE_{[s1]}; notetoSWE_{[s2]}$

$info2CodeGrp$ denotes

$reqtoMC; notetoDE_{[d]}$

$info2TestGrp$ denotes

$reqtoMTP; reqtoMUTP; reqtoTU; notetoDE_{[d]}; notetoQAE_{[q]}$

$info2MonitorGrp$ denotes $reqtoMP; notetoPM$.

where d, q, $s1$ and $s2$ are variables containing the names of the most recently assigned design engineer, QA engineer and software engineers respectively.

The operation pattern of SAT states that after receiving the requirement and authorization from CCB, the SAT object is allowed to repeatedly schedule and assign tasks and supply the relevant information to the concerned personnels and step objects.

Modify Design—*MD*

- identify the receive operations

Description	Operation Name
requirement change from *SAT*	*reqfrSAT*
design review from *RD*	*revfrRD*

- identify the supply operations

Description	Operation Name
(modified) design to *RD*	*destoRD*
(modified) design to *MC*	*destoMC*
(modified) design to *MUTP*	*destoMUTP*
Notification of completion of the step to *MP*	*endMDtoMP*

- Let *doingMD* denote the name of the step operation.
- The operation pattern for *MD* is

$reqfrSAT;$
$doingMD; destoRD; destoMC; destoMUTP;$
$(endMDtoMP; fin$
$\oplus$
$(revfrRD; doingMD; destoRD; destoMC; destoMUTP)^+; endMDtoMP; fin)$

This step object first of all is allowed to do the design. Upon the completion, the design is sent to *RD*, *MC* and *MUTP*. If the design is approved unconditionally, a notification of the completion of the step is sent to *MP*. However changes may be recommended by *RD*, in which case, this step object will then need to receive the reviewed design from *RD* and repeat the process.

Review Design—*RD*

- identify the receive operations

Description	Operation Name
requirement change from *SAT*	*reqfrSAT*
(modified) design from *MD*	*desfrMD*

- identify the supply operations

Description	Operation Name
design review feedback to *MD*	*revtoMD*
outcome info to *MP*	*infotoMP*
Notification of completion of the step to *MP*	*endRDtoMP*

- Let *doingRD* denote the name of the step operation.

- The operation pattern for *RD* is

$reqfrSAT$;
$desfrMD$; $doingRD$;
($endRDtoMP$; fin
$\oplus$
($revtoMD$; $infotoMP$; $desfrMD$; $doingRD$)$^+$; $endRDtoMP$); fin

Modify Code—*MC*

- identify the receive operations

Description	Operation Name
requirement change from *SAT*	*reqfrSAT*
design from *MD*	*desfrMD*
feedback from *TU*	*codebackfrTU*

- identify the supply operations

Description	Operation Name
(modified) source code to *TU*	*srctoMUTP*
object code to *TU*	*codetoTU*
Notification of completion of the step to *MP*	*endMCtoMP*

- Let *doingMC* denote the name of the step operation.

- The operation pattern for MC is

$reqfrSAT$;
$desfrMD; (doingMC; srctoMUTP; codetoTU; codebackfrTU)^+$;
$endMCtoMP; fin$

Modify Test Plan—MTP

- identify the receive operations

Description	Operation Name
requirement change from SAT	$reqfrSAT$

- identify the supply operations

Description	Operation Name
(modified) test plan to $MUTP$	$tplantoMUTP$
Notification of completion of the step to MP	$endMTPtoMP$

- Let $doingMTP$ denote the name of the step operation.

- The operation pattern for MTP is

$reqfrSAT; (doingMTP; tplantoMUTP)^+; endMTPtoMP; fin$

Modify Unit Test Package—$MUTP$

- identify the receive operations

Description	Operation Name
requirement change from SAT	$reqfrSAT$
test plan from MTP	$tplanfrMTP$
modified desing from MD	$desfrMD$
source code from MC	$srcfrMC$
feedback from TU	$packbackfrTU$

- identify the supply operations

Description	Operation Name
unit test package to TU	$packtoTU$
Notification of completion of the step to MP	$endMUTPtoMP$

- Let *doingMUTP* denote the name of the step operation.
- The operation pattern for *MUTP* is

$reqfrSAT$;
$(desfrMD; srcfrMC; tplanfrMTP; doingMUTP;$
$packtoTU; packbackfrTU)^+$;
$endMUTPtoMP; fin$

Test Unit—*TU*

- identify the receive operations

Description	Operation Name
requirement change from *SAT*	*reqfrSAT*
object code from *MC*	*codefrMC*
unit test package from *MUTP*	*packfrMUTP*

- identify the supply operations

Description	Operation Name
feedback of code to *MC*	*codebacktoMC*
feedback of test package to *MUTP*	*packbacktoMUTP*
notification of successful testing	*succtoMP*
Notification of completion of the step to *MP*	*endTUtoMP*

- Let *doingTU* denote the name of the step operation.
- The operation pattern for *TU* is

$reqfrSAT$;
$(codefrMC; packfrMUTP; doingTU; codebacktoMC; packbacktoMUTP)^+$;
$succtoMP; endTUtoMP; fin$

Monitor Progress—*PM*

- identify the receive operations

Description	**Operation Name**
requirement change from *SAT*	*reqfrSAT*
notifications of completions from all steps	*notefrSAT*
	notefrMD
	notefrRD
	notefrMC
	notefrMTP
	notefrMUTP
	notefrTU
	succfrTU
	infofrRD
cancellation from *CCB*	*canfrCCB*

- identify the supply operations

Description	**Operation Name**
recommendation to CCB	*rectoCCB*

- Let *doingMP* denote the name of the step operation.
- The operation pattern for *MP* is

$reqfrSAT$;
$(notefrSAT \oplus notefrMD \oplus notefrRD \oplus notefrMC \oplus notefrMTP \oplus notefrMUTP \oplus notefrTU \oplus infofrRD \oplus doingMP \oplus rectoCCB)^+$;
$(succfrTU \oplus canfrCCB); fin$

Acknowledgement

The authors would like to thank Reidar Conradi, Jean-Claude Derniame and Maryse Bourdon for providing essential research information. Technical comments from Bob Snowdon have been especially valuable.

References

[1] S. Bandinelli, A.Fuggetta and C. Ghezzi, "SLANG Solution for the ISPW-7 Software Process Example", October, 1991.

[2] S. Bandinelli, A.Fuggetta, C. Ghezzi and S. Grigolli, "Process Enactment in SPADE", Proceedings of the 2nd European Workshop, EWSPT'92, Trondheim, Norway, September 1992, LNCS 635.

[3] N. Belkhatir, J. Estublier and w.L. Melo, “ADELE2:a Support to Large Software Development Process”, Proceedings of the First International Conference on the Software Process, October, 1991.

[4] N. Belkhatir, J. Estublier and w.L. Melo, “Software Process Modeling in Adele: The ISPW-7 Example”, October, 1991.

[5] M. Bourdon, “PROCESS WEAVER: Process Modelling Experience Report”, Technical Report Series TR 92014, Cap Gemini Innovation, 1992.

[6] R. Conradi, et al., “Design, Use and Implementation of SPELL, a Language for Software Process Modeling and Evolution”, Proceeding of the second European Workshop, EWSPT’92, Trondheim, Norway, September 1992.

[7] R. Conradi, et al., “The EPOS Approach to the ISPW-7 Software Process Modeling Example”, October, 1991.

[8] B. Curtis, M.I. Kellner and J. Over, “Process Modelling”, Communication of ACM, Volume 35, Number 9, September, 1992.

[9] C. Fernström, “PROCESS WEAVER: Adding Process Support to UNIX”, Proceedings of the Second International Conference on the Software Process, Berlin, Germany, February, 1993.

[10] M. Greenwood, “BM stepper”, Department of Electronics and Computer Science, The University of Southampton, May 1993.

[11] N. Goldman and K. Narayanaswamy, “Solution to ISPW-7 Process Example”, October, 1991.

[12] P. Henderson, “Enact:a functional, object-oriented modelling language”, Reference Manual, Department of Electronics and Computer Science, University of Southampton, July,1992.

[13] K.E. Huff “ISPW-7 Process Example in GRAPPLE”, October, 1991.

[14] M. Kellner, “Representation Formalisms for Software Process Modeling”, Proceeding of the 4th International Software Process Workshop: Representing and Enacting the Software Process, ACM Software Engineering Notes 14(4), June, 1989.

[15] M.I. Kellner, “Solution to the ISPW-7 Software Process Example”, October, 1991.

[16] M. Kellner, P.Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo and H.D.Rombach, “ISPW-6 Software Process Example”, Proceedings of the First International Conference on Software Process. IEEE Computer Society Press, Washington, DC, 1991.

[17] P.H. Feiler, WS Humphrey: "Software Process Development And Enactment: Concepts And Definitions", Proceedings of the second International Conference on Software Process 1993.

[18] J. Lonchamp, "A Structured Conceptual and Terminological Framework for Software Process Engineering", Proceedings of the Second International Conference on the Software Process, Berlin, Germany, February, 1993.

[19] I. Robertson and M. Greenwood, "The Implementation of the ISPW-6 Example in PWI", Department of Computer Science, University of Manchester, May, 1993.

[20] B. Peuschel and W. Schaffer, "ISPW-7 Exerise Solution", October, 1991.

[21] J. Sa, B. C. Warboys, "Specifying Concurrent Object-based Systems using Combined Specification Notations", Technical Report Series UMCS-91-9-2, Department of Computer Science, University of Manchester, July, 1991.

[22] J. Sa, "BM Specification of the Surrey Project PSS Demonstration Model", Draft Note, Department of Computer Science, University of Manchester, February, 1992.

[23] J. Sa, B. C. Warboys, "Integrating a Formal Specification Method with PML: A Case Study", Lecture Notes in Computer Science 635, page 106-123, Springer-Verlag, September, 1992.

[24] S.M. Sutton, Jr., "APPL/A Solution for the ISPW7 Coordination and Communication Problem", October, 1991.

[25] B.C. Warboys, "The ISPE2.5 Project: Process Modelling as the basis for a Support Environment", Proceedings of the first International Conference on Software Development, Environments and Factories, Berlin, 1989.