# Applying a Concurrent Formal Framework to Process Modelling

J.A. Keane, J. Sa and B.C. Warboys,

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

Published in the proceedings of Formal Methods Europe'94, Barcelona, October 1994.

# Applying a Concurrent Formal Framework to Process Modelling*

J.A. Keane[1], J. Sa[2] and B.C. Warboys[3]

[1]Department of Computation, UMIST, Manchester, UK.
[2]Department of Computing, University of the West of England, Bristol, UK.
[3]Department of Computer Science, University of Manchester, UK.

**Abstract.** In this paper OBM, a method for developing concurrent systems in a formal framework is described. The method arose from an industrially collaborative project, and has been used, in various forms, to specify parts of operating systems, databases and retail systems.
OBM allows a system to be gradually refined by levelled descriptions. Varying granularities of activities can be specified, Most importantly, OBM has a formal basis to enable refinement to be verified. In this paper, this framework is applied to process modelling.

## 1 Motivation

The importance of process models and their evolution have been recognised. Many different approaches have been developed in this field, for example, ADELE [2], ProcessWise [4], SPELL [5] and PROCESS WEAVER [6]. However, very few process models have a formal basis to incorporate the design of concurrent activities or to allow the gradual evolution of the process description. A strong motivation for this work is a belief that both features are essential.

The extra complexity introduced by concurrency in an organisation means that the need for process models is even more essential. Of particular importance with the models of such organisations is their evolutionary nature, the expression of different levels, the composition of components within a level, the gradual refinement between levels, the expression of possible different granularities of process activity, and a formal basis to support the framework.

The work described in this paper provides mechanisms to capture the evolutionary description of organisational processes in a concurrent framework. This paper describes a framework, *Organisational Base Model (OBM)*, which provides a means to gradually identify the details, and to refine the detailed representation, of the organisational process.

The framework evolved from work on the industrially collaborative Esprit project, EDS, and has been used to specify "real" systems. Because of this close link to industry the work has been influenced by "user" requirements. In particular, it has been necessary to take on board comments from non-formalists and tailor the approach accordingly. Overall, the work has benefited from this influence.

* This work was was carried out whilst all authors were at the University of Manchester, and was supported by UK SERC grants GR/J11034 and GR/J48979.

OBM is based on Base Model (BM) [11]. BM and OBM are discussed in sections 2 and 3 respectively. In section 4 the temporal semantics of OBM constructs are given. An example refinement is discussed in section 5. Section 6 discusses various issues of relevance to process modelling and the OBM approach.

## 2 BM

BM was developed to specify concurrent systems, the components of which may interfere. Among the available formalisms that were suitable for describing such systems, temporal logic was used because it was capable of expressing liveness properties [10]. The particular temporal logic chosen offered a compositional approach [1]. Using this approach, a system is considered as a collection of components. Each component is specified as a temporal logic formula. The specification of the overall system is obtained by composing all the component specifications. It was found that the specification notation was rich enough to express the properties of the types of concurrent system of concern. However, the specifications were not easy to comprehend. This was of particular concern in an industrially collaborative project.

As a result, more and more high level constructs have been gradually introduced to denote the temporal logic formulae. The collection of constructs has evolved into a high level language with its semantics defined in temporal logic.

In the classification given in [16], BM is an object-based language. However, the motivation for defining BM was not to provide yet another object-oriented language, but rather to make it easier to specify systems using temporal logic.

In BM, a system is composed of components which may be executed concurrently. The model of a system is a collection of component specifications and how these components interact with each other. Each component provides operations and contains user defined variables only accessible to the operations. An operation may call other operations in other components. The behaviour of a component is specified by the pattern to which its operations, when executed, conform. Such patterns are called *operation patterns*.

Case studies, [12, 13], have shown that BM addresses many aspects of process modelling. Whilst BM is a suitable methodology for *representing* processes, it does not enable the development and refinement of a model. Nevertheless BM was considered a good basis on which to develop a more general framework. This more general framework, OBM, places the emphasis on the *development* and *refinement* of a concurrent process model.

## 3 OBM Constructs and Refinement

In OBM, a process is considered to be composed of components which can be executed concurrently. The model of a process is developed at several levels. At each level, the model contains a collection of component specifications. The difference from BM is that a component specification does not need to contain *every* detail. These abstract component specifications can be refined in the subsequent levels.

OBM provides *abstract objects* to model processes in an organisation at a level of abstraction, and *boxed objects* to compose a number of components and hide internal operations of the composition.

An *abstract object* models part of a process which may be refined later. In OBM, there are *compound operations* and *simple operations*. A compound operation may be refined into several operations which may themselves be compound operations. In the rest of the paper the term *operation* denotes a *simple operation* and the term *coperation* denotes both *compound operations* and *operations*.

An abstract object *provides* coperations and may *require* coperations provided by other components. The *interface* of an abstract object consists of a list of provided coperations, required coperations and an operation pattern defined in terms of the provided coperations. Operation patterns are defined using an *ordering expression*. The detailed definition of ordering expressions can be found in [11]. The *body* of an abstract object consists of a *call template* for each required coperation and a definition for each provided coperation which contains its type, i.e. active or passive, its *call pattern*, and its *pre* and *post-conditions*. Each call template is in the form:

| !*name* | |
|---|---|
| prop | *func/proc* |
| obj | *another* |
| apar | $v$ |

where !*name* is the name of the template, "prop" defines the type of the call, e.g. whether the caller receives a result, "obj" states the name of the component providing the required coperation and "apar" specifies the actual parameter.

Call patterns are defined using ordering expressions, and specify the order of the calls made to the required coperations. Call patterns are defined in terms of call template names. Pre and post-conditions are defined in terms of the parameter, the result and the values received from the calls made to the required coperations.

The behaviour of an abstract object is determined by the execution of its coperations. The order of these executions must conform to the operation pattern. The temporal semantics of abstract objects are given in section 4.1.

An abstract object may be *refined* by decomposing it into a number of sub-components. In order to justify the refinement, it is necessary to show that the composition of the sub-components is consistent with the abstract object.

The level i+1 refinement of a level i abstract object consists of:

1. *Component Refinement:* replacing the level i abstract object by a number of level i+1 components, and defining each level i+1 component.
2. *Coperation Refinement*: replacing the sub-patterns of the level i operation pattern by operation patterns defined in terms of level i+1 coperations. A sub-operation pattern may also be a single coperation. Given a level i+1 refinement, a level i+1 coperation is *internal* if it is not used in the *Coperation Refinement* part.

The consistency checking of refinements is described in section 4.6.

A *boxed object* is used for checking the consistency of refinements. It composes level i+1 components together to show that the composition is consistent with the level i abstract object. A boxed object contains a number of components and a list of internal coperations. The behaviour of a boxed object is the concurrent executions of all the contained components. The behaviour of the internal coperations is not visible. The temporal semantics of boxed objects are defined in section 4.5.

## 4 Temporal Semantics of OBM Constructs

OBM constructs are defined using the temporal logic described in [1]. The specification of a component specifies a set of sequences. In this model parallelism is modelled by *interleaving* events of different components, each specified sequence contains idle steps as well as the events of the component. Each idle step can be viewed as a slot for another component to perform an event. Idle steps are called *environment steps* and other steps are called *component steps*. In order to distinguish between component and environment steps, each component is associated with a *component proposition*, that has the value `True` if its component is performing an event, and the value `False` if its component is doing nothing visible externally. The *context* of a component contains the component proposition, component name, names of all provided coperations, names of all required coperations and names of all variables.

The intuitive meaning of each of the temporal operators used in the rest of this paper is as follows:

$\Box P$ specifies that $P$ is always true; $P\ \mathcal{C}\ Q$ specifies that $P$ is true and then $Q$ is true; $P\ \mathcal{C}^*\ Q$ specifies $P$ must be true at least once before $Q$ is true; $P\ \mathcal{U}\ Q$ specifies $P$ is true until $Q$ is true; $\bigcirc P$ specifies that $P$ is true at the next moment; *fin* specifies the last state of a sequence, i.e. $\neg \bigcirc\ true$.

### 4.1 Temporal Semantics of Abstract Objects

The semantics of an abstract object is determined by the semantics of its operation pattern with the constraint that at most one event may take place at any time. This constraint is to ensure that parallel activities are interleaved. Let:

- $Abs$ denote the name of an abstract object;
- $C$ denote the context in which the semantics of $Abs$ is defined;
- $pattern$ denote the operation pattern of $Abs$;
- $atmostone(C)$ denote the formula stating that at most one event may take place;
- $[\![Abs]\!]_C$ denote the semantics of $Abs$ in context $C$;

The semantics of $Abs$ is

$$[\![Abs]\!]_C \triangleq [\![pattern]\!]_C \ \wedge\ \Box atmostone(C)$$

$pattern$ is defined using an ordering expression. The semantics of a subset of the ordering expression is defined in section 4.2. $atmostone(C)$ is defined in [15].

### 4.2 Temporal Semantics of Ordering Expressions

A subset of the ordering expression is defined here. The full definition is in [15].

$$[\![\bowtie]\!]_C \triangleq fin$$

$$[\![P1; P2]\!]_C \triangleq [\![P1]\!]_C \; \mathcal{C} \; [\![P2]\!]_C$$

$$[\![P1 \mid\mid P2]\!]_C \triangleq ([\![P1]\!]_{C1} \; \mathcal{C} \; (idle(C1) \, \mathcal{U} \, fin)) \;\wedge\; ([\![P2]\!]_{C2} \; \mathcal{C} \; (idle(C2) \, \mathcal{U} \, fin))$$

$$[\![P^+]\!]_C \triangleq [\![P]\!]_C \; \mathcal{C}^* \; fin$$

where $P1; P2$ means $P1$ followed by $P2$; $P1 \mid\mid P2$ means $P1$ and $P2$ in parallel; $P^+$ means one or more occurrences; $idle(C1)$ means nothing in $C1$ is changed; and $\bowtie$ means the end.

### 4.3 Temporal Semantics of Operations

The semantics of an operation is determined by a sequence of events. The first event is to accept the call request. After this there may be a number of calls to operations in other components. The order of these calls must conform to the call pattern specified in this operation. The last event is to return. Let:

- $op$ denote an operation provided by the component $Abs$;
- $\pi_{Abs}$ denote the component proposition of $Abs$;
- $acc\text{-}op$ and $ret\text{-}op$ denote the propositions indicating the starting and finishing events;
- $s$ denote the variable of $Abs$;
- $p$ denote the parameter of $op$;
- $r$ denote the result of $op$;
- $callpattern$ denote the call pattern of $op$;
- $precond(p, s)$ denote the pre-condition of $op$;
- $postcond(p, s, s', r, res\text{-}O1\text{-}op1, res\text{-}O2\text{-}op2, ...)$ denote the post-condition of $op$; where $res\text{-}O1\text{-}op1$ and $res\text{-}O2\text{-}op2$ are the variables containing the results received from the calls made during the execution of $op$;

The temporal semantics is

$$[\![op]\!]_C \triangleq accepting(C, op, precond) \; \mathcal{C} \; [\![callpattern]\!]_C \; \mathcal{C} \; returning(C, op, postcond)$$

$$\begin{gathered} accepting(C, op, precond) \triangleq \\ idle(C) \, \mathcal{U} \\ (\pi_{Abs} \;\wedge\; acc\text{-}op \;\wedge \\ reset.history(C) \wedge \\ precond(p, s) \;\wedge\; fin) \end{gathered}$$

Briefly, the behaviour of *op* may be preceded by a number of idle steps, followed by the `accept` event. The pre-condition must be true at the time the `accept` event is taking place.

$$
\begin{gathered}
returning(C, op, postcond) \triangleq \\
idle(C)\, \mathcal{U} \\
(\pi_{Abs} \ \wedge \ ret\text{-}op \wedge \\
\exists v, hv1, hv2, \ldots \cdot \\
s = v \wedge \\
res\text{-}O1\text{-}op1 = hv1 \wedge \\
res\text{-}O2\text{-}op2 = hv2 \wedge \\
\ldots \wedge \\
\bigcirc(\exists v' \cdot s = v' \wedge \\
postcond(p, v, v', r, hv1, hv2, \ldots) \ \wedge \ fin))
\end{gathered}
$$

In a similar way to the `accept` event, the `return` event may be preceded by a number of idle steps. After the `return` event, the post-condition must be true. Note how the post-condition is embedded in the temporal logic formula. In this approach, the post-condition includes intermediate values such as $res\text{-}O1\text{-}op1$. This makes it possible to specify the interference behaviour.

*callpattern* is an ordering expression defined in terms of the names of the call templates. The semantics of the ordering expression is defined in section 4.2. The semantics of the call templates is defined in the next sub-section.

### 4.4 Temporal Semantics of Call Templates

There are two types of calls: functional and procedural. In this paper, only the semantics of the functional call templates is described.

The semantics of a functional call is that the component making the call may "perform" a few idle steps before sending the call request. Once the request is sent, the component idles until it receives the result returned from the called operation. Let:

- $\pi_{Abs}$ denote the component proposition of component *Abs*;
- $send\text{-}O1\text{-}op1$ and $rec\text{-}O1\text{-}op1$ denote the propositions of the `send` and `receive` events of this call;
- $apar\text{-}O1\text{-}op1$ denote the variable containing the parameters for this call; $apar\text{-}O1\text{-}op1$ is an array, each element of the array contains the parameter to each occurrence of the call;
- $fres\text{-}O1\text{-}op1$ denote the variable containing the results received from the calls to $op1$; similarly to $apar\text{-}O1\text{-}op1$, $fres\text{-}O1\text{-}op1$ is also an array;
- $occno\text{-}O1\text{-}op1$ denote the index variable recording the number of times that $op1$ is called by the operation; $occno\text{-}O1\text{-}op1$ is used to index into $apar\text{-}O1\text{-}op1$ and $fres\text{-}O1\text{-}op1$;

– the following template denote the call template of the call made by $Abs$.

Given the call template,

| $!op1$ | |
|---|---|
| prop | $func$ |
| obj | $O1$ |
| apar | $v$ |

The semantics is

$$[\![!op1]\!]_C \triangleq sending(C, O1, op1, v)\,\mathcal{C}\; receiving(C, O1, op1)$$

$$\begin{gathered} sending(C, O1, op1, v) \triangleq \\ (idle(C) \;\wedge\; \exists n \cdot occno\text{-}O1\text{-}op1 = n)\, \mathcal{U} \\ (\pi_{Abs} \;\wedge\; send\text{-}O1\text{-}op1 \wedge \\ apar\text{-}O1\text{-}op1(n) = v) \;\wedge\; fin) \end{gathered}$$

$$\begin{gathered} receiving(C, O1, op1) \triangleq \\ (idle(C) \;\wedge\; \exists n \cdot occno\text{-}O1\text{-}op1 = n)\, \mathcal{U} \\ (\pi_{Abs} \;\wedge\; rec\text{-}O1\text{-}op1 \wedge \\ \exists r \cdot fres\text{-}O1\text{-}op1(n) = r \wedge \\ occno\text{-}O1\text{-}op1 = n + 1 \;\wedge\; fin) \end{gathered}$$

The value of $r$ is bound to the actual value returned by $op1$ when the component $Abs$ is composed with the component $O1$.

### 4.5 Temporal Semantics of Boxed Objects

The behaviour of a boxed object is defined as the set of all possible execution sequences of the boxed object. Each step of an execution sequence of a boxed component, 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 a coperation of the other. In this case, the corresponding events from the two components must take place in parallel. In a boxed object, events of the internal coperations are not visible. Let:

- $Bobj$ denote the name of a boxed object containing two subcomponents $C1$ and $C2$;
- $W$, $W1$ and $W2$ denote the contexts of $Bobj$, $C1$ and $C2$ respectively;
- $Iops$ denote the internal coperations of $Bobj$;
- $interleave(W, W1, W2)$ denote the temporal formula stating that events from $W1$ and $W2$ are interleaved;

- $synchronise(W1, W2)$ denote the temporal formula stating that there exists an event in $W1$ which is synchronised with an event in $W2$;
- $hide(\{C1, C2\}, Iops)$ denote the temporal formula stating that all the events made by coperations in $Iops$ are hidden.

The semantics of $Bobj$ is

$$\begin{gathered} [\![Bobj]\!]_W \triangleq \\ ([\![C1]\!]_{W1}\ \mathcal{C}\ (idle(W1)\ \mathcal{U}\ fin)) \wedge \\ ([\![C2]\!]_{W2}\ \mathcal{C}\ (idle(W2)\ \mathcal{U}\ fin)) \wedge \\ \Box(interleave(W, W1, W2) \vee \\ synchronise(W1, W2)) \wedge \\ \Box hide(\{C1, C2\}, Iops) \end{gathered}$$

$C1$ and $C2$ are extended with idle steps so that they have the same length. *idle*, *interleave*, *synchronise* and *hide* have their intuitive meanings, and are formally defined in [15].

### 4.6 Consistency of Refinement

Given a level i abstract object $A$ and its level i+1 refinement, the consistency of the level i+1 refinement is validated by the following stages:

1. Identify the internal coperations. Let $Iops$ denote the set of these coperations.

2. Let $B$ denote the boxed object of the level i+1 refinement which contains: $r1$, ..., $rn$.

3. Replace the operation pattern of $A$ with level i+1 coperations.
   Let $A'$ denote the same specification as $A$ with the operation pattern replaced by the corresponding operation patterns defined in the *Coperation Refinement.*

4. Hide calls made to internal coperations.
   After replacing the coperations of $A$ with level i+1 coperations, $A'$ may then contain calls made to the internal coperations. Let $hidecall(Iops, \{r1, ..., rn\})$ denote the formula stating that the internal calls are hidden.

5. Extend the variables of $A'$ so that $A'$ and $B$ contain the same variables.
   Let *vars* denote the set of variables in $r1$,...,$rn$, but not in A'; $anyvalue(vars)$ denote the formula stating that variables in *vars* can have any value.

Let $Ca$ and $Cb$ denote the contexts of $A'$ and $B$ respectively; the level i+1 refinement is consistent with $A$ if:

$$[\![B]\!]_{Cb} \Rightarrow ([\![A']\!]_{Ca}\ \wedge\ \Box hidecalls(Iops, \{r1, ..., rn\})\ \wedge\ \Box anyvalue(vars))$$

Again due to lack of space, *hidecalls* and *anyvalue* are not defined here.

## 5 Example

Consider a university department that provides three coperations which may be performed repeatedly in parallel.

**Abstract Object** $MUCS_0$ **Interface**
**provides**: $teach_0$, $research_0$, $others_0$
**requires**: *none*
**operation pattern**: $(teach_0^+ \mid\mid research_0^+ \mid\mid others_0^+); \bowtie$
**Abstract Object** $MUCS_0$ **Body**
**call templates**: *none*
**coperation definitions**:
$teach_0$ **is** passive operation
$teach_0$'s **call pattern**: $\bowtie$

The pre-condition and the post-condition are trivial, and therefore they are not described here. Other coperations can be defined similarly.

### 5.1 Level 1 Refinement of $MUCS_0$

**Component Refinement**

1. Replacement: divide MUCS into sub-components, $IPG_1$, $CNC_1$, $LECTURES_1$, $ETC_1$.
2. Definitions of the Level 1 Components:

**Abstract Object** $IPG_1$ **Interface**
**provides**: $ipgteach_1$, $ipgresearch_1$, $ipgothers_1$
**requires**: $LECTURES_1$: $teachPost_1$
**operation pattern**: $(ipgteach_1^+ \mid\mid ipgresearch_1^+ \mid\mid ipgothers_1^+); \bowtie$

$IPG_1$ provides three coperations and requires one from $LECTURES_1$.

**Abstract Object** $IPG_1$ **Body**
**call templates**:

| $!teachPost_1$ | |
|---|---|
| prop | *func* |
| obj | $LECTURES_1$ |
| apar | $v$ |

**coperation definitions**:
$ipgteach_1$ **is** passive
$ipgteach_1$'s **call pattern**: $!teachPost_1; \bowtie$
$ipgteach_1$'s pre- and post-conditions:
*ipgteach(c)* $\triangleq$
**pre** *valid(c)*
**post** *projectmarks' = newmarks(projectmarks,res-teachPost*$_1$*)*

where *res-teachPost*$_1$ is the result returned from the call to $LECTURES_1$.

Other coperations can be defined similarly.

**Abstract Object** $LECTURES_1$ **Interface**
**provides**: $teachUnder_1, teachPost_1, teachExt_1$
**requires**: $LECTURES$: $teachPost_1$
**operation pattern**: $(teachUnder_1^+ \,||\, teachPost_1^+ \,||\, teachExt_1^+); \bowtie$

The body part of $LECTURES$ is trivial and hence not included here.

**Coperation Refinement**

Replace the sub-operation patterns of $MUCS_0$:

$$teach_0^+ \triangleq ipgteach_1^+ \,||\, cncteach_1^+ \,||\, etcteach_1^+$$

$$research_0^+ \triangleq ipgresearch_1^+ \,||\, cncresearch_1^+ \,||\, etcresearch_1^+$$

$$others_0^+ \triangleq ipgothers_1^+ \,||\, cncothers_1^+ \,||\, etcothers_1^+$$

where $cncteach_1$, $etcteach_1$, etc, are coperations provided by $CNC$ and $ETC$.

### 5.2 Consistency of the Level 1 Refinement

1. Let *LECTURESops* denote the set containing the coperation names provided by $LECTURES_1$ but not used in the *Coperation Refinement*.

2. Let *Boxlevel*1 denote the boxed object of the level 1 refinement.

   **Boxed Object** *BoxLevel1*
   **contains**: $IPG_1, CNC_1, LECTURES_1, ETC_1$
   **hidden coperations**: *LECTURESops*

3. Let $MUCS'$ denote the same specification as $MUCS_0$ with the coperations replaced by the operation patterns defined in the level 1 *Coperation Refinement*.

$$(ipgteach_1^+ \,||\, cncteach_1^+ \,||\, etcteach_1^+ \,||\, ipgresearch_1^+ \,||$$

$$cncresearch_1^+ \,||\, etcresearch_1^+ \,||\, ipgothers_1^+ \,||\, cncothers_1^+ \,||\, etcothers_1^+); \bowtie$$

4. Let $hidecalls(LECTURESops, \{IPG_1, CNC_1, LECTURES_1, ETC_1\})$ denote the formula for hiding the internal calls.

5. Let $anyvalue(vars)$ denote the formula for extending the variables of $MUCS'$.

The aim is to prove:

$$\llbracket Boxlevel1 \rrbracket_{Cb} \Rightarrow$$
$$(\llbracket MUCS' \rrbracket_{Cm} \wedge$$
$$\Box anyvalue(vars)) \wedge$$
$$\Box hidecalls(LECTURESops, \{IPG_1, CNC_1, LECTURES_1, ETC_1\})$$

Let $Ci$, $Cc$, $Cl$ and $Ce$ denote the contexts of $IPG$, $CNC$, $LECTURES$ and $ETC$ respectively; from section 4.5, then the boxed object $Boxlevel1$ can be expanded:

$$\llbracket Boxlevel1 \rrbracket_{Cb} \triangleq$$
$$(\llbracket IPG_1 \rrbracket_{Ci}\ \mathcal{C}\ (idle(Ci)\ \mathcal{U}\ fin)) \wedge$$
$$(\llbracket CNC_1 \rrbracket_{Cc}\ \mathcal{C}\ (idle(Cc)\ \mathcal{U}\ fin)) \wedge$$
$$(\llbracket LECTURES_1 \rrbracket_{Cl}\ \mathcal{C}\ (idle(Cl)\ \mathcal{U}\ fin)) \wedge$$
$$(\llbracket ETC_1 \rrbracket_{Ce}\ \mathcal{C}\ (idle(Ce)\ \mathcal{U}\ fin)) \wedge$$
$$\Box(interleaven(Cb, Ci, Cc, Cl, Ce) \vee$$
$$synchronisen(Ci, Cc, Cl, Ce)) \wedge$$
$$\Box hide(LECTURESops, \{IPG_1, CNC_1, LECTURES_1, ETC_1\})$$

where *interleaven* and *synchronisen* are the general forms for interleaving and synchronisation.

The above formula can be expanded (from section 4.1):

$$\llbracket Boxlevel1 \rrbracket_{Cb} \triangleq$$
$$((\llbracket IPG_1.pattern \rrbracket_{Ci}\ \wedge \Box atmostone(Ci))\ \mathcal{C}\ (idle(Ci)\ \mathcal{U}\ fin)) \wedge$$
$$((\llbracket CNC_1.pattern \rrbracket_{Cc}\ \wedge \Box atmostone(Cc))\ \mathcal{C}\ (idle(Cc)\ \mathcal{U}\ fin)) \wedge$$
$$((\llbracket LECTURES_1.pattern \rrbracket_{Cl}\ \wedge \Box atmostone(Cl))\ \mathcal{C}\ (idle(Cl)\ \mathcal{U}\ fin)) \wedge$$
$$((\llbracket ETC_1.pattern \rrbracket_{Ce}\ \wedge \Box atmostone(Ce))\ \mathcal{C}\ (idle(Ce)\ \mathcal{U}\ fin)) \wedge$$
$$\Box(interleaven(Cb, Ci, Cc, Cl, Ce) \vee$$
$$synchronisen(Ci, Cc, Cl, Ce)) \wedge$$
$$\Box hide(LECTURESops, \{IPG_1, CNC_1, LECTURES_1, ETC_1\})$$

From section 4.2 further expansion of the operation patterns gives:

$$\llbracket Boxlevel1 \rrbracket_{Cb} \triangleq$$
$$((((\llbracket ipgteach_1^+ \rrbracket_{Cit}\ \mathcal{C}\ (idle(Cit)\ \mathcal{U}\ fin)) \wedge$$
$$(\llbracket ipgresearch_1^+ \rrbracket_{Cir}\ \mathcal{C}\ (idle(Cir)\ \mathcal{U}\ fin)) \wedge$$
$$(\llbracket ipgothers_1^+ \rrbracket_{Cio}\ \mathcal{C}\ (idle(Cio)\ \mathcal{U}\ fin)))\ \mathcal{C}\ fin) \wedge$$
$$\Box atmostone(Ci))\ \mathcal{C}\ (idle(Ci)\ \mathcal{U}\ fin)) \wedge$$
$$(...) \wedge$$
$$\Box(interleaven(Cb, Ci, Cc, Cl, Ce) \vee$$
$$synchronisen(Ci, Cc, Cl, Ce)) \wedge$$
$$\Box hide(LECTURESops, \{IPG, CNC, LECTURES, ETC\})$$

The above formula states that:

- $Boxlevel1$ is an intersection of the semantics of all the level 1 coperations (in their iterated forms),
- each coperation may have a tail of idle steps,
- at any time only one event within each of $IPG$, $CNC$, $LECTURES$, and $ETC$ may take place,
- all events within the whole composition must be either interleaved or synchronised,
- all events of the internal coperations are hidden.

Now consider the right hand side of the proof:

$[\![MUCS']\!]$ expands to

$$[\![(ipgteach_1^+ \parallel cncteach_1^+ \parallel etcteach_1^+ \parallel ipgresearch_1^+ \parallel$$
$$cncresearch_1^+ \parallel etcteach_1^+ \parallel ipgothers_1^+ \parallel cncothers_1^+ \parallel etcothers_1^+); \bowtie]\!]_{Cm} \wedge$$
$$\Box atmostone(Cm) \wedge$$
$$\Box anyvalue(vars)) \wedge$$
$$\Box hidecalls(LECTURESops, \{IPG, CNC, LECTURES, ETC\}))$$

Expanding the semantics of the parallel operators, gives:

$$(([\![ipgteach_1^+]\!]_{Cit}\ \mathcal{C}\ (idle(Cit)\ \mathcal{U}\ fin)) \wedge$$
$$([\![cncteach_1^+]\!]_{Cct}\ \mathcal{C}\ (idle(Cct)\ \mathcal{U}\ fin)) \wedge$$
$$([\![etcteach_1^+]\!]_{Cet}\ \mathcal{C}\ (idle(Cet)\ \mathcal{U}\ fin)) \wedge$$
$$(...))\ \mathcal{C}\ fin) \wedge$$
$$\Box atmostone(Cm) \wedge$$
$$\Box hidecalls(LECTURESops, comps) \wedge$$
$$\Box anyvalue(vars)$$

The above formula states that:

- $MUCS'$ is defined as the intersection of all the level 1 coperations (in their iteration forms) except the internal ones, i.e. the coperations provided by $LECTURES$,
- each coperation may have a tail of idle steps,
- there is at most one event taking place in $MUCS'$,
- all calls made to the internal coperations are hidden,
- variables used in level 1 specification, but not in $MUCS'$, may have any value.

Comparing the definitions, it is not difficult to see that any behaviour defined by the left hand side of the proof satisfies the definition on the right hand side.

The abstract objects in level 1, e.g. $IPG$, $CNC$, can be further refined. However due to lack of space, they are not described here.

## 6 Discussion

In this section a number of issues are discussed, ranging from related and future work, through to the underlying background of the approaches taken in OBM.

### 6.1 Granularity

To be generally applicable a process model should not dictate any particular granularity of process or of process refinement. A process model is a specification level activity and does not indicate a particular implementation, thus it does not provide a model of computation.

OBM, therefore, makes no commitment to granularity of activity, and thus allows the designer to determine what is the most suitable grain. This is supported by the provision for gradual refinement and levelled description.

### 6.2 Specification of Interference

In [8], a method for specifying concurrent behaviour with possible interference is defined in terms of `rely` and `guarantee` conditions. The method is used to specify a system for shared variables. In OBM, communication and interaction between different components can only be achieved via operation calls. The only possible interference comes from receiving results from an operation call to another component. Therefore, to address interference, it is only necessary to consider results received from the operation calls.

### 6.3 Tool Support and Verification

The approach in this paper models the same part of a process at different levels. It also allows the gradual development of a model using the notion of refinement. An example was given and consistency shown by a hand-proof. Such proofs are error-prone, and so the *BM Stepper* [3] has been developed that provides a simulation facility to step through operation patterns. Further study will investigate the role of the BM Stepper in OBM.

### 6.4 Development of OBM

In OBM, abstract objects cannot contain user defined variables. Case studies suggest that this is insufficient to represent the behaviour of some processes. Future work will investigate how to reason about the consistency between the refinement and its abstract object if the abstract object contains user defined variables. A possible solution is to use the data reification technique of [7].

The integration of OBM with the temporal-logic based design methodology described in [9] is also under investigation.

### 6.5 Formal Process Modelling

The approach is based on a belief that ultimately a process model must have a formal basis underlying it. Therefore, OBM provides a process model that has a formal basis, with the semantics defined in a linear time temporal logic.

One of the special features in process modelling is that many of the activities in processes are human-oriented. When defining a model for such activities, it is important that the model is flexible enough to permit human creativity. Therefore, many people have advocated abstract, coarse-grained models for such activities. This appears to be so that the human developer does not find all the detail taken care of, and thus does not have all his/her creativity stifled. Unfortunately there seems to be a misconception that formal methods are only suitable for describing very detailed models. The consequence of this is that formal approaches are often considered inappropriate for process modelling.

On the contrary, this paper has attempted to illustrate that it is possible to model activities at different levels of abstraction using formal methods. The method described in this paper is based on temporal logic. By using language constructs to hide the details in the temporal logic the approach is made more practical.

A formal basis enables, where appropriate, a formal verification approach to be taken to validate the model. However, even in cases where verification of this nature is unlikely to occur, the formal basis enables many design issues to be more thoroughly considered.

## References


1. H.Barringer, R.Kuiper, and A.Pnueli, Now You May Compose Temporal Logic Specifications, *Proc. of 16th ACM Symp. on Theory of Computing*, 1984.
2. N. Belkhatir, J. Estublier and W.L. Melo, Software Process Model and Work Space Control in the Adele System, *Proc. of the Second International Conference on the Software Process*, Berlin, Germany, 1993.
3. R.F. Bruynoghe, R.M. Greenwood, I. Robertson, J. Sa and B.C. Warboys, PADM: Towards a Total Process Modelling System, in *Software Process Modelling and Technology*, A. Finkelstein, J. Kramer and B. Nuseibeh (Eds.), Research Studies Press, Wiley, 1994.
4. R.F. Bruynoghe, J.M. Parker, and J.S. Rowles, PSS: A System for Process Enactment, *Proc. of the First International Conference on the Software Process, Manufacturing Complex Systems*, IEEE Press, 1991.
5. R. Conradi, M.L. Jaccheri, C. Mazzi, M.N. Nguyen and A. Aarsten, Design, Use and Implementation of SPELL, a Language for Software Process Modeling and Evolution, *Proc. of the Second European Workshop on Software Process Technology – EWSPT'92*, Trondheim, Norway, 1992.
6. C. Fernström, PROCESS WEAVER: Adding Process Support to UNIX, *Proc. of the Second International Conference on the Software Process*, Berlin, Germany, 1993.
7. C.B. Jones, *Systematic Software Development Using VDM*, Prentice-Hall, 1990.
8. C.B. Jones, Tentative Steps Toward a Development Method for Interfering Programs, *ACM TOPLAS* 5(4), 1983.
9. J.A. Keane and W.Hussak, A Formal Approach to Parallel Resource Bindings, *Proc. 16th Int. Conf. on Software Engineering – ICSE-16*, pp. 15-22, IEEE Press, 1994.

10. L. Lamport, What Good is Temporal Logic?, *Information Processing 83*, R.E.A. Mason (Ed.), pp. 657-668, IFIP, 1983.
11. J. Sa and B. C. Warboys, Specifying Concurrent Object-based Systems using Combined Specification Notations, Technical Report UMCS-91-7-2, Department of Computer Science, University of Manchester, July 1991.
12. J. Sa and B. C. Warboys, Integrating a Formal Specification Method with PML: A Case Study, *Proc. of the 2nd European Workshop on Software Process Technology – EWSPT'92*, LNCS-635, Springer-Verlag, 1992.
13. J. Sa and B.C. Warboys, Modelling Processes using a Stepwise Refinement Technique, *Proc. of the 4th European Workshop on Software Process Technology – EWSPT'94*, LNCS-772, Springer-Verlag, 1994.
14. J.Sa and B.C. Warboys, A Formal Description of the ISPW-6 Software Process Example, Technical Report Series UMCS-93-6-1, Department of Computer Science, University of Manchester, June 1993.
15. J.Sa, OBM: A Tutorial, Project Report, Department of Computer Science, University of Manchester, April 1993.
16. P. Wegner, Dimensions of Object-Based Language Design, *Proc. of OOPSLA'87*, 1987.