# OBM: A Specification Method for Modelling Organisational Process

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

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

Published in the proceedings of the Workshop on Constraint Processing at CSAM'93 in St. Petersburg, 1993.

# 1

# OBM: A Specification Method for Modelling Organisational Process

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

ABSTRACT

Recently many process modelling approaches have been developed. However, in most of these approaches, it is often assumed that the processes in an organisation are clearly defined, and therefore the emphasis of process modelling is often put on the final representation of the organisational process. However, in reality, processes are often not clearly defined. It is important to provide mechanisms to capture the processes. In this paper, we present an approach, called OBM, which provides a means to gradually identify the details of the processes in an organisation and develops the detailled representation of the organisational process by refinement.

OBM is a method for developing models of organisational process. In OBM, a process model is developed at different levels. Each level may be composed of several components. Each component at level i may be refined in a subsequent level. The consistency between the level i component and its level i+1 refinement is checked.

OBM provides constructs which can be used to model abstract behaviour and detailled behaviour including an agent's behaviour and activities. For refinement purposes, OBM also provides composition and hiding facilities. The semantics of the OBM constructs are defined in a linear time temporal logic.

In this paper, we will use an example to explain how a process model may be gradually refined in OBM.

## 1.1 Introduction and Background

Recently many process modelling approaches have been developed, for instance, [17], [6], [3, 4], [5], [1], [12] and [7]. However, in many process modelling approaches, it is often assumed that the processes in an organisation are clearly defined, and therefore the emphasis of process modelling is often put on the final representation of the organisational process. However in reality, processes are often not clearly defined. It is important to provide mechanisms to capture such processes. In this paper, we present an approach, called Organisational Base Model (OBM), which provides a means to gradually identify the details of the processes in an organisation and develops the detailed representation of the organisational process by refinement.

OBM is based on Base Model (BM) [13], which was originally developed for specifying concurrent object-based systems that allow interference among objects. BM provides a framework so that specifications are written in a systematic way. A temporal logic semantics is given for each specification construct. This formal foundation makes it possible to reason about BM specifications mathematically.

At the 6th International Software Process Workshop, a list of process aspects and issues

[1] All authors are with:
Department of Computer Science,
University of Manchester,
Manchester M13 9PL,
England.

thought to be important when modelling the software process, were summarised. Our case studies [14, 15] have shown that BM is capable of addressing many of these aspects. These aspects include: control flow, sequencing and constraints, feedback to earlier steps, iterations, object/information management and flow through process, organisational responsibility for specific tasks, and parallelism. Whilst BM has been shown to be a suitable methodology for representing processes, the emphasis of OBM is on the support for the *development* of a process model.

## 1.2 Brief Overview of BM

In BM [13], a system is considered to be composed of components which may be executed in parallel. The model of a system, therefore, contains a collection of component specifications and how these components interact with each other. Each component provides a number of operations and contains some user defined 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 triggered by some conditions within the component, and a passive operation is invoked 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 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 invoked 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.

BM provides two component constructs and one composition construct:

**Agent object:** used to model a person's behaviour that is relevant to the modelled process.

**Step object:** used to model the details of the *activities*.

**Composed object:** used to compose the components of a system.

## 1.3 OBM constructs

In OBM, a system is also considered to be composed of components which can be executed in parallel. The final model of a system 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 all the details. These abstract component specifications can be refined in the subsequent levels.

In addition to the constructs provided by BM, OBM also provides the following constructs:

**Abstract object:** used to model abstract *processes* in an organisation.

**Boxed object:** used to compose a number of components and to hide internal operations of the composition.

The next two sections describe the abstract object and the boxed object.

## 1.4 Abstract Object

### Representation

In OBM, there are two types of operations: compound operations and simple operations. A compound operation may be refined into several operations which may themselves be compound operations. In the rest of this paper we use the term operation to denote the term simple operation and the term *coperation* to denote both the term compound operation and the term operation. An abstract object provides some coperations and it may also require some coperations provided by other components. The definition of an abstract object is divided into two parts: the interface part and the body part.

The interface part consists of:

- a list of provided coperations
- a list of required coperations
- 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 [13, 16]. Operation patterns are defined in terms of coperation names.

The body part consists of

- a *call template* for each required coperation
- a definition for each provided coperation which contains
  - the type of this coperation, i.e. active or passive
  - the *call pattern*
  - the pre- and the post-conditions of the coperation

Each call template is in the form

| !*name* | |
|---|---|
| prop | *func* |
| obj | *another* |
| apar | *v* |

where !*name* is the name of the template, "prop" defines the type of the call, e.g. a functional call sends a call request and receives a result, a procedural call sends a call request and does not expect any result, "obj" states the name of the component providing the required coperation and "apar" specifies the actual parameter.

Call patterns are also defined using ordering expressions. The call pattern of a coperation specifies the order of the calls made to the required coperations. Call patterns are defined in terms of call template names.

The pre- and the post-conditions of a compound operation are defined in terms of the parameter, the result and the values received from the calls made to the required coperations.

#### Semantics

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 1.7.1.

#### Usage

An abstract object is used to model part of a process at an abstract level which may be refined later. In [11] and [8], a process is defined as a set of partially ordered process steps, with sets of related artifacts, human and computerised resources, organisational structures and constraints, intended to reach a goal. In OBM, the behaviour specified by an abstract object may include the behaviours of both agents and activities.

## 1.5 Refinement

In OBM, a process model of an organisation is developed at different *levels*. A level may be composed of several components. Each component may be either an abstract object, an agent object or a step object. Abstract objects may be refined in subsequent levels. A process model is developed by gradually refining the abstract objects.

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

**Component Refinement**

- Replacing the level i abstract object by a number of level i+1 components.
- Defining each level i+1 component.

**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.

The consistency checking of refinements is described in section 1.8.

## 1.6 Boxed Object

As explained above, in the course of developing a model, 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.

#### Representation

A boxed object contains a number of components and a list of internal coperations.

#### Semantics

Given a level i+1 refinement of a level i component, a level i+1 coperation is *internal* if it is not used in the **Coperation Refinement** part.

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 1.7.2.

#### Usage

A boxed object is used to compose level i+1 components together to show that the composition is consistent with the level i abstract object.

## 1.7 Temporal Semantics of OBM Constructs

In this paper we define the semantics of OBM constructs using the temporal logic described in [2].

#### The Model

The specification of a component specifies a set of sequences. In our 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 the environment steps and the other steps are called the component steps.

#### Component Proposition

In order to distinguish between the component and the environment steps, each component is associated with a proposition, which we refer to as a component proposition. A component proposition has the value *True* if its component is making an event, and it has the value *False* if its component is doing nothing (visible externally). Detailed discussions on this type of interleaving model can be found in [2].

#### Context

A context of a component contains the component proposition, component name, names of all the provided coperation, names of all the required coperations and names of all the variables.

### 1.7.1 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;
- $\pi_{abs}$ denote the component proposition of $Abs$;
- $X$ denote the context in which the semantics of $Abs$ is defined;
- *pattern* denote the operation pattern of $Abs$;
- $atmostone(X)$ denote the formula stating that at most one event may take place;
- $[\![Abs]\!]_X$ denote the semantics of $Abs$ in context $X$;

The semantics of $Abs$ is

$$[\![Abs]\!]_X \triangleq [\![pattern]\!]_X \ \wedge \ \Box atmostone(X)$$

*pattern* is defined using an ordering expression. The semantics of a subset of the ordering expression is defined in section 1.7.3. $\Box P$ specifies that $P$ is always true. Due to lack of space, $atmostone(X)$ is not defined in this paper. For interested readers, it can be found in [16].

### 1.7.2 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, that consists 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 the case of synchronisation, 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;
- $\pi$, $\pi_1$ and $\pi_2$ denote the component propositions 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

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

In the above formula, $C1$ and $C2$ are extended with idle steps so that they have the same length. $P\ \mathcal{C}\ Q$ specifies that $P$ is true and then $Q$ is true. The definitions of *idle*, *interleave*, *synchronise* and *hide* are given in [16].

### 1.7.3 Temporal Semantics of Ordering Expression

In this section only a subset of the ordering expression is defined. The full definition can be found in [16].

$$[\![\bowtie]\!]_C \triangleq \neg \bigcirc true$$

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

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

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

Where $\bigcirc P$ specifies that $P$ is true at the next moment of the computation, *fin* indicates the end, i.e. $\neg \bigcirc true$.

## 1.8 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. Define the boxed object of the level i+1 refinement.

   Let $B$ denote this boxed object,
   Boxed Object $B$

   **contains**
   $r1, r2, ..., rn$

   **hidden coperations**
   *Iops*

3. Replace the operation pattern of $A$ with level i+1 coperations.

   Let $A'$ denote the same specification as $A$ except that the operation pattern is replaced by the corresponding operation patterns as defined in the **Coperation Refinement** part.

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 $hidecall(Iops, \{r1, r2, ..., rn\})$ 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 $r1$, $r2$, ..., $rn$, 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; we define that the level i+1 refinement is consistent with $A$ if

$$[\![B]\!]_{Cb} \Rightarrow$$
$$([\![A']\!]_{Ca} \wedge$$
$$\Box hidecalls(Iops, \{r1, r2, ..., rn\}) \wedge$$
$$\Box anyvalue(vars))$$

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

## 1.9 An Example

Consider the Manchester University Computer Science (MUCS) department as an example, it provides three coperations: teach, research and others. All three operations may be performed repeatedly in parallel.

### 1.9.1 MUCS

Abstract Object $MUCS_0$ The Interface Part:

**provides**

$$:teach_0, research_0, others_0$$

**requires**

$$:none$$

**operation pattern**

$$(teach_0^+ \;||\; research_0^+ \;||\; others_0^+); \bowtie$$

The subscript 0 indicates the level number. They may be omitted if the level number is obvious.

In the **operation pattern** part, $teach_0^+$ denotes one or more occurrences of $teach_0$; $teach_0^+ \parallel research_0^+$ denotes $teach_0^+$ and $research_0^+$ in parallel; $\bowtie$ denotes the end of computation.

Abstract Object $MUCS_0$ The Body Part:

**call templates**

*none*

**coperation definitions**

$teach_0$ **is** passive

$teach_0$**'s call pattern**

$\bowtie$

$teach_0$**'s pre- and post-conditions**

$teach_0() \triangleq$

pre $True$

post $True$

---

Other coperations can be defined similarly.

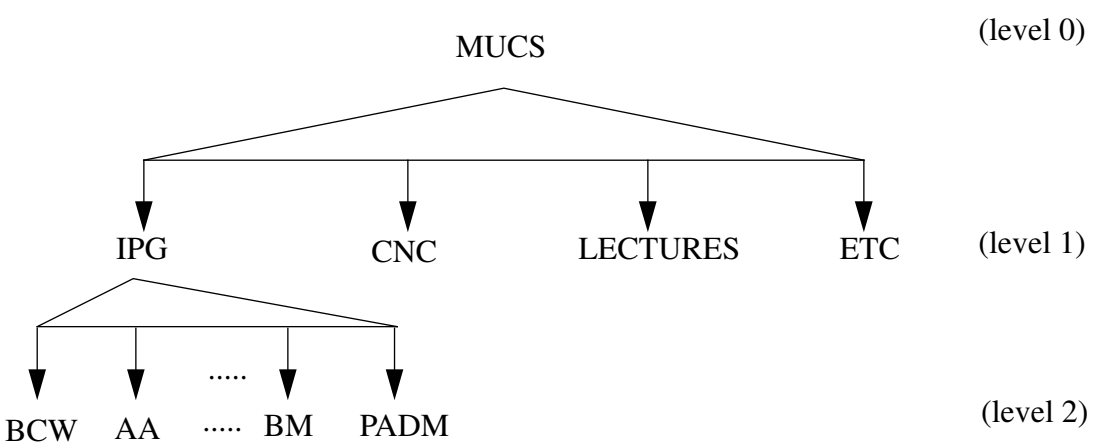


FIGURE 1.1. Levels of Component Refinement in MUCS

### 1.9.2 Level 1 Refinement of *MUCS*

**Component Refinement**

- Replacement
  MUCS is subdivided into a number of sub-components: IPG, CNC, LECTURES, ETC. The notation is as follows:

  $MUCS_0 :: IPG_1, CNC_1, LECTURES_1, ETC_1.$

- Definitions of the Level 1 Components

  Abstract Object $IPG_1$ The Interface Part:

  **provides**

  $:ipgteach_1, ipgresearch_1, ipgothers_1$

  **requires**

  $LECTURES$:$teachPost_1$

  **operation pattern**

  $(ipgteach_1^+ \,||\, ipgresearch_1^+ \,||\, ipgothers_1^+); \bowtie$

  *IPG* provides three coperations and requires one from *LECTURES*.

  Abstract Object $IPG_1$ The Body Part:

  **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**

  $ipgteach_1() \triangleq$

  pre $True$

  post $True$

---

Other coperations can be defined similarly.

Abstract Object $LECTURES_1$ The Interface Part:

**provides**

$:teachU_1, teachPost_1, teachPeve_1$

**requires**

$:none$

**operation pattern**

$$(teachU_1^+ \;||\; teachPost_1^+ \;||\; teachPeve_1^+);\bowtie$$

The body part of *LECTURES* is trivial and hence not included here.
Other components can be defined similarly.

**Coperation Refinement**

The sub-operation patterns of *MUCS* are replaced as follows:

$$teach_0^+ \triangleq ipgteach_1^+ \;||\; cncteach_1^+ \;||\; etcteach_1^+$$
$$research_0^+ \triangleq ipgresearch_1^+ \;||\; cncresearch_1^+ \;||\; etcresearch_1^+$$
$$others_0^+ \triangleq ipgothers_1^+ \;||\; cncothers_1^+ \;||\; etcothers_1^+$$

where $cncteach_1$, $cncresearch_1$, $etcteach_1$ etc. are coperations provided by *CNC* and *ETC*.

### 1.9.3 Consistency of the Level 1 Refinement

1. In the level 1 refinement, coperations provided by *LECTURES* are not used in the **Coperation Refinement** part. Therefore, by our definition, these are the internal coperations for this refinement. Let *LECTURESops* denote the set containing these coperation names.

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

   Boxed Object *Boxlevel*1

   **contains**
   $IPG, CNC, LECTURES, ETC$

   **hidden coperations**
   *LECTURESops*

3. Let *MUCS′* denote the same specification as *MUCS* except that the coperations are replaced according to the level 1 **Coperation refinement** part. The resulting operation pattern is

$$(ipgteach_1^+ \;||\; cncteach_1^+ \;||\; etcteach_1^+ \;||\; ipgresearch_1^+ \;||\; cncresearch_1^+ \;||\; etcresearch_1^+ \;||\; ipgothers_1^+ \;||\; cncothers_1^+ \;||\; etcothers_1^+); \bowtie$$

4. Let $hidecalls(LECTURESops, \{IPG, CNC, LECTURES, ETC\})$ 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

$$[\![Boxlevel1]\!]_{Cb} \Rightarrow ([\![MUCS']\!]_{Cm} \wedge \Box hidecalls(LECTURESops, \{IPG, CNC, LECTURES, ETC\}) \wedge \Box anyvalue(vars))$$

Let $Ci$, $Cc$, $Cl$ and $Ce$ denote the contexts of $IPG$, $CNC$, $LECTURES$ and $ETC$ respectively; according to section 1.7.2, the semantics of the boxed object $Boxlevel1$ can be expanded as follows:

$$\begin{aligned}&[\![Boxlevel1]\!]_{Cb} \triangleq \\ &([\![IPG]\!]_{Ci}\, \mathcal{C}\ (idle(Ci)\, \mathcal{U}\ fin)) \wedge \\ &([\![CNC]\!]_{Cc}\, \mathcal{C}\ (idle(Cc)\, \mathcal{U}\ fin)) \wedge \\ &([\![LECTURES]\!]_{Cl}\, \mathcal{C}\ (idle(Cl)\, \mathcal{U}\ fin)) \wedge \\ &([\![ETC]\!]_{Ce}\, \mathcal{C}\ (idle(Ce)\, \mathcal{U}\ fin)) \wedge \\ &\Box(interleaven(Cb, Ci, Cc, Cl, Ce) \vee synchronisen(Ci, Cc, Cl, Ce)) \wedge \\ &\Box hide(LECTURESops, \{IPG, CNC, LECTURES, ETC\})\end{aligned}$$

Where *interleaven* and *synchronisen* are the general forms for interleaving and synchronisation. Again their definitions can be found in [16]. If we expand the semantic definitions of the abstract objects $IPG$, $CNC$, $LECTURES$ and $ETC$ according to the definition given in section 1.7.1, the above formula can be expanded as follows:

$$\begin{aligned}&[\![Boxlevel1]\!]_{Cb} \triangleq \\ &(([\![IPG.pattern]\!]_{Ci} \wedge \Box atmostone(Ci))\, \mathcal{C}\ (idle(Ci)\, \mathcal{U}\ fin)) \wedge \\ &(([\![CNC.pattern]\!]_{Cc} \wedge \Box atmostone(Cc))\, \mathcal{C}\ (idle(Cc)\, \mathcal{U}\ fin)) \wedge \\ &(([\![LECTURES.pattern]\!]_{Cl} \wedge \Box atmostone(Cl))\, \mathcal{C}\ (idle(Cl)\, \mathcal{U}\ fin)) \wedge \\ &(([\![ETC.pattern]\!]_{Ce} \wedge \Box atmostone(Ce))\, \mathcal{C}\ (idle(Ce)\, \mathcal{U}\ fin)) \wedge \\ &\Box(interleaven(Cb, Ci, Cc, Cl, Ce) \vee synchronisen(Ci, Cc, Cl, Ce)) \wedge \\ &\Box hide(LECTURESops, \{IPG, CNC, LECTURES, ETC\})\end{aligned}$$

We then further expand the semantics of the operation patterns according to the definitions given in section 1.7.3 to give:

$$
\begin{array}{l}
[\![Boxlevel1]\!]_{Cb} \triangleq \\
(((((([\![ipgteach_1^+]\!]_{Cit}\ \mathcal{C}\ (idle(Cit)\ \mathcal{U}\ fin))\ \wedge \\
([\![ipgresearch_1^+]\!]_{Cir}\ \mathcal{C}\ (idle(Cir)\ \mathcal{U}\ fin))\ \wedge \\
([\![ipgothers_1^+]\!]_{Cio}\ \mathcal{C}\ (idle(Cio)\ \mathcal{U}\ fin)))\ \mathcal{C}\ fin)\ \wedge \\
\Box atmostone(Ci))\ \mathcal{C}\ (idle(Ci)\ \mathcal{U}\ fin))\ \wedge \\
((((([\![cncteach_1^+]\!]_{Cct}\ \mathcal{C}\ (idle(Cct)\ \mathcal{U}\ fin))\ \wedge \\
([\![cncresearch_1^+]\!]_{Ccr}\ \mathcal{C}\ (idle(Ccr)\ \mathcal{U}\ fin))\ \wedge \\
([\![cncothers_1^+]\!]_{Cco}\ \mathcal{C}\ (idle(Cco)\ \mathcal{U}\ fin)))\ \mathcal{C}\ fin)\ \wedge \\
\Box atmostone(Cc))\ \mathcal{C}\ (idle(Cc)\ \mathcal{U}\ fin))\ \wedge \\
((((([\![teachU_1^+]\!]_{Clu}\ \mathcal{C}\ (idle(Clu)\ \mathcal{U}\ fin))\ \wedge \\
([\![teachPost_1^+]\!]_{Clpo}\ \mathcal{C}\ (idle(Clpo)\ \mathcal{U}\ fin))\ \wedge \\
([\![teachPeve_1^+]\!]_{Clpe}\ \mathcal{C}\ (idle(Clpe)\ \mathcal{U}\ fin)))\ \mathcal{C}\ fin)\ \wedge \\
\Box atmostone(Cl))\ \mathcal{C}\ (idle(Cl)\ \mathcal{U}\ fin))\ \wedge \\
((((([\![etcteach_1^+]\!]_{Cet}\ \mathcal{C}\ (idle(Cet)\ \mathcal{U}\ fin))\ \wedge \\
([\![etcresearch_1^+]\!]_{Cer}\ \mathcal{C}\ (idle(Cer)\ \mathcal{U}\ fin))\ \wedge \\
([\![etcothers_1^+]\!]_{Ceo}\ \mathcal{C}\ (idle(Ceo)\ \mathcal{U}\ fin)))\ \mathcal{C}\ fin)\ \wedge \\
\Box atmostone(Ce))\ \mathcal{C}\ (idle(Ce)\ \mathcal{U}\ fin))\ \wedge \\
\Box(interleaven(Cb, Ci, Cc, Cl, Ce)\ \vee\ synchronisen(Ci, Cc, Cl, Ce))\ \wedge \\
\Box hide(LECTURESops, \{IPG, CNC, LECTURES, ETC\})
\end{array}
$$

The above formula defines that:

- *Boxlevel*1 is an intersection of the semantics of all the level 1 coperations (in their iteration forms),
- each coperation may have a tail of idle steps,
- at any time only one event within $IPG$ may take place,
- at any time only one event within $CNC$ may take place,
- at any time only one event within $LECTURES$ may take place,
- at any time only one event within $ETC$ 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, if we expand $[\![MUCS']\!]$, we get the following formula,

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

Again expanding the semantics of the parallel operators, we get,

$$
\begin{array}{l}
((([\![ipgteach_1^+]\!]_{Cit}\ \mathcal{C}\ (idle(Cit)\ \mathcal{U}\ fin))\ \wedge \\
([\![cncteach_1^+]\!]_{Cct}\ \mathcal{C}\ (idle(Cct)\ \mathcal{U}\ fin))\ \wedge \\
([\![etcteach_1^+]\!]_{Cet}\ \mathcal{C}\ (idle(Cet)\ \mathcal{U}\ fin))\ \wedge \\
([\![ipgresearch_1^+]\!]_{Cir}\ \mathcal{C}\ (idle(Cir)\ \mathcal{U}\ fin))\ \wedge \\
([\![cncresearch_1^+]\!]_{Ccr}\ \mathcal{C}\ (idle(Ccr)\ \mathcal{U}\ fin))\ \wedge \\
([\![etcresearch_1^+]\!]_{Cer}\ \mathcal{C}\ (idle(Cer)\ \mathcal{U}\ fin))\ \wedge \\
([\![ipgothers_1^+]\!]_{Cio}\ \mathcal{C}\ (idle(Cio)\ \mathcal{U}\ fin))\ \wedge \\
([\![cncothers_1^+]\!]_{Cco}\ \mathcal{C}\ (idle(Cco)\ \mathcal{U}\ fin))\ \wedge \\
([\![etcothers_1^+]\!]_{Ceo}\ \mathcal{C}\ (idle(Ceo)\ \mathcal{U}\ fin)))\ \mathcal{C}\ fin)\ \wedge \\
\Box atmostone(Cm)\ \wedge \\
\Box hidecalls(LECTURESops, comps)\ \wedge \\
\Box anyvalue(vars))
\end{array}
$$

The above formula defines that

- $MUCS'$ is defined as an 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 on both the left and the right hand sides, 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.

## 1.10 Discussions and Future Work

The work on OBM is still an on-going research. As mentioned in the introduction, our earlier work was mainly concerned with *modelling* processes, however we often found it was difficult to capture all the details of an organisational process in a specification because different people view the processes at different levels of abstractions. This first led us to think about representing different parts of an organisational process at different levels of abstractions. However this did not solve our problem because the same part of a process may be viewed at different levels of abstractions. The work presented in this paper solves this problem by modelling the same part of a process at different levels. In addition to being able to represent different views, we also found it very useful to capture the process and gradually develop a model of it using the notion of refinement.

In the current version of OBM, an abstract object does not contain any user defined variables. Some of our case studies have shown that this is not sufficient to represent the

behaviours of some processes. One aspect of our future work is to investigate how to reason about the consistency between the refinement and its abstract object if the abstract object contains some user defined variables. A possible solution is to use the data reification technique as described in [10].

In this paper, an example of a hand proof was shown. It is not difficult to see that mistakes can be made easily in the proof itself due to the complexity. At the moment we have developed a BM Stepper using the object-oriented proto-typing language *Enact* [9]. The BM Stepper provides a simulation facility by stepping through the operation patterns of the components. Further study will be carried out to investigate the role of the BM Stepper in OBM.

*Acknowledgements:* The authors would like to thank Dr. R.A. Snowdon for comments on the earlier work and M. Greenwood for discussions on BM and the implementation of the BM Stepper.

This work was funded by the CNC group of the Department of Computer Science, University of Manchester.

## 1.11 References

[1] S. Bandinelli, A.Fuggetta, C. Ghezzi and S. Grigolli, "Process Enactment in SPADE", Proc. of the 2nd European Workshop, EWSPT'92, Trondheim, Norway, September 1992, LNCS 635, pages 67-83.

[2] H.Barringer, R.Kuiper and A.Pnueli, Now you may compose temporal logic specifications, Proc. of the 16th ACM symp. on Theory of Computing, 1984.

[3] N. Belkhatir, J. Estublier and W.L. Melo, "ADELE2:a Support to Large Software Development Process", Proc. of the 1st International Conference on the Software Process, October 1991.

[4] N. Belkhatir, J. Estublier and W.L. Melo, "Software Process Model and Work Space Control in the Adele System", Proc. of the 2nd International Conference on the Software Process, Berlin, Germany, February 1993.

[5] R. Conradi, et al., "Design, Use and Implementation of SPELL, a Language for Software Process Modeling and Evolution", Proc. of the 2nd European Workshop, EWSPT'92, Trondheim, Norway, September 1992, LNCS 635, pages 167-177.

[6] C. Fernström, "PROCESS WEAVER: Adding Process Support to UNIX", Proc. of the 2nd International Conference on the Software Process, Berlin, Germany, February 1993.

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

[8] P.H. Feiler, WS Humphrey: "Software Process Development And Enactment: Concepts And Definitions", Proc. of the 2nd International Conference on Software Process, 1993.

[9] P. Henderson, "Object-Oriented Specification and Design with C++". International Series in Software Engineering, McGraw-Hill, 1993.

[10] C.B. Jones, "Systematic Software Development Using VDM", Prentice-Hall International Series in Computer Science, C.A.R. Hoare Series Editor, 1986.

[11] J. Lonchamp, "A Structured Conceptual and Terminological Framework for Software Process Engineering", Proc. of the 2nd International Conference on Software Process, 1993.

[12] B. Peuschel, W. Schäfer and S. Wolf, "A Knowledge-based Software Development Environment Supporting Cooperative Work", International Journal of Software Engineering and Knowledge Engineering (SeKe), Vol. 2, No. 1, World Scientific Publishing Company, March 1992.

[13] J. Sa, B. C. Warboys, "Specifying Concurrent Object-based Systems using Combined Specification Notations", Technical Report Series UMCS-91-7-2, Department of Computer Science, University of Manchester, July 1991.

[14] J. Sa, B. C. Warboys, "Integrating a Formal Specification Method with PML: A Case Study", Proc. of the 2nd European Workshop, EWSPT'92, Trondheim, Norway, September 1992, LNCS 635, pages 106-122.

[15] J.Sa, 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.

[16] J.Sa, "OBM: A Tutorial", Project Report, Department of Computer Science, University of Manchester, April 1993.

[17] 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.