OBM: A Tutorial

1 Introduction and Background

Recently many process modelling approaches have been developed, for instance, [5], [3], [4], [1], [9] and [6]. 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 always 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 document, 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 based on Base Model (BM) [10] which was originally developed for specifying concurrent object-based systems that allow interferences 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, which were thought to be important to accommodate when modelling software process, were summarised. Our case studies [11, 12] have shown that BM is capable of addressing many of these aspects. Among them the major aspects are: 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 proved to be a suitable methodology for representing processes, the emphasis of OBM is on the *development* of a process model.

2 Brief Overview of BM

In BM [10], 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 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 follow: 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 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 OBM constructs

OBM provides the following component constructs:

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

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

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

For refinement purpose, OBM also provides the following two constructs in addition to the above three component constructs.

Composed object used to compose a number of components.

Boxed object used to compose a number of components and to hide some operations of the composition.

4 Abstract Object

Representation

An abstract object provides some interfaces and it may also require some interfaces provided by other components. An interface is either an operation or a compound operation. We use the term *coperation* to denote either an operation or a compound operation. A compound operation is one which may be refined into several coperations. A coperation may be either active or passive.

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 *ordering expression.* (See appendix F for the notations of ordering expression. Operation patterns are defined in terms of coperation names.)

The body part consists of

- a *call template* for each required coperation (see appendix I for call template)
- definition for each provided coperation
  - type of this coperation, i.e. active or passive
  - the *call pattern*
  - the pre- and the post-conditions of the coperation

Each call template specifies which component provides this required operation, what type of call this is, e.g. function or procedure, what the parameter is.

Call patterns are also defined using ordering expression. 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 an active compound operation is defined in terms of the relationship between the values received from the calls to the required coperations. The pre- and the post-conditions of a compound passive operation is defined in terms of the parameter and the result as well as the values received from the calls 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. In an abstract object, coperations may be executed in parallel with each other. The temporal semantics of an abstract object is given in appendix C.

Usage

An abstract object is used to model part of a process at an abstract level which may be refined later. In [8] and [7], 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.

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. Here we assume that MUCS is a self-contained component. It does not require any coperation.

Abstract Object $MUCS_0$ The Interface Part:

provides

$:teach_0, research_0, others_0$

requires

$:none$

operation pattern

$$(teach_0^+ \mid\mid research_0^+ \mid\mid others_0^+); \circ$$

The subscript 0 indicates the level number. The concept of levels will be explained in the refinement section. The subscript may be omitted if the level number is obvious.

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

Abstract Object $MUCS_0$ The Body Part:

call templates

$none$

coperation definitions

$teach_0$ **is** passive

$teach_0$**'s call pattern**

$\circ$

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

$teach_0() \quad \triangleq$

pre $True$

post $True$

$research_0$ **is** passive

$research_0$**'s call pattern**

$\circ$

$research_0$**'s pre- and post-conditions**

$research_0() \quad \triangleq$

pre $True$

post $True$

$others_0$ **is** passive

$others_0$**'s call pattern**

$\circ$

$others_0$**'s pre- and post-conditions**

$$others_0() \quad \triangleq$$

pre $True$

post $True$

There is no call template because $MUCS$ does not require any coperation. All three coperations are passive.

5 Agent Object (age)

Representation

An agent object contains some encapsulated user defined variables and provides a number of operations. Operations may be either active or passive.

The interface definition of an agent object consists of

- a list of provided operations
- a list of required coperations
- an operation pattern defined in terms of the provided operations.

Note that an agent object may require compound operations as well as operations. This allows a model to be partially refined and therefore different components may be modelled at different levels of abstraction. The operation pattern of an agent object may not use the parallel operator.

The body part consists of

- variable declarations
- a call template for each required coperation
- definition for each provided operation
  - type of the operation, i.e. active or passive
  - the call pattern defined in terms of the names of the call templates
  - the pre- and the post-condition

Semantics

The behaviour of an agent object is a sequential execution of its operations. The order of the executions must conform to the operation pattern. The temporal semantics of an agent object is defined in appendix D.

Usage

An agent object is used to model the behaviour of a person relevant to the process being modelled.

An Example

Consider the agent BCW who provides three operations: *bcwteach*, *bcwresearch* and *bcwfindfound*; requires three operations: *teachPost*, *designbm* and *designpadm*.

Agent Object *BCW* The Interface Part:

provides

$$:bcwteach, bcwresearch, bcwfindfound$$

requires

$$\begin{array}{ll} LECTURES_1 & :teachPost_1 \\ BM_2 & :designbm_2 \\ PADM_2 & :designpadm_2 \end{array}$$

operation pattern

$$(bcwteach \oplus bcwresearch \oplus bcwfindfound)^+ ; \circ$$

The **requires** part lists the names of all the components and their coperations required by this agent object. In the **operation pattern** part, $bcwteach \oplus bcwresearch$ denotes either *bcwteach* or *bcwresearch*.

Agent Object *BCW* The Body Part:

variables $hrspent$: $Hourspent$

call templates

$!teachPost_1$	
prop	$func$
obj	$LECTURE_1$
apar	$hours$

$!designbm_2$	
prop	$func$
obj	BM_2
apar	$hours$

$!designpadm_2$	
prop	$func$
obj	$PADBM_2$
apar	$hours$

operation definitions

bcwteach **is** passive

bcwteach's **call pattern**

$$!teachPost_1 ; \circ$$

bcwteach's **pre- and post-conditions**

$$bcwteach(hours) \quad \triangleq$$

$$\begin{array}{l} \mathsf{pre}\ True \\ \mathsf{post}\ \exists tt, rr, oo \cdot hrspent = mk\text{-}Hourspent(tt, rr, oo) \ \wedge \\ \qquad hrspent' = mk((tt + hours), rr, oo) \end{array}$$

---

bcwresearch **is** passive

bcwresearch's **call pattern**

$$(!designbm_2 \oplus !designpadm_2); \circ$$

bcwresearch's **pre- and post-conditions**

$$bcwresearch(hours) \quad \triangleq$$

$$\begin{array}{l} \textsf{pre } True \\ \textsf{post } \exists tt, rr, oo \cdot hrspent = mk\text{-}Hourspent(tt, rr, oo) \ \wedge \\ \quad\quad hrspent' = mk(tt, (rr + hours), oo) \end{array}$$

———————————————-

In the **variables** part *hrspent* denotes a variable recording the hours spent on each type of activity. *Hourspent* denotes the type of the variable. Hourspent consists of hours spent on teaching, research and other activities.

$$Hourspent :: t : Integer, r : Integer, o : Integer$$

In each of the above call templates, the first line, e.g. $!teachPost_1$, denotes the name of the call template, the template name is always the name of the required operation preceded by the ! mark. The second line indicates the type of the call, e.g. function or procedure. The third line identifies the component being called. The last line specifies the actual parameters of the call.

Note that the level numbers in the call templates of BCW are different. As mentioned before, in OBM different components may be modelled at different levels of abstraction. In this example the component LECTURES is modelled at level 1 and the other two components, i.e. BM and PADM are modelled at level 2. See the refinement section 7 for more details.

The post-condition of *bcwteach* states that after each execution of *bcwteach*, the hour spent on teaching is added up. The other operations are similar.

6 Step Object (step)

Representation

The definition of a step is similar to an agent.

6.1 Semantics

Again it is similar to an agent object. See appendix E for the temporal semantic definition.

Usage

A step object is used to model the details of activities. In [8] and [7] an activity is defined as an elementary process step with no visible substructure.

An Example

Consider a step object BM which models the activities related to the work on BM.

<u>Step Object</u> *BM* <u>The Interface Part:</u>

provides

$: designbm, casestudy, designtoolreq$

requires

$: none$

operation pattern

$$(designmethod \oplus casestudy \oplus designtoolreq)^{+}; \circ$$

Step Object *BM* The Body Part:

variables $methodfile$: $File$, $casefile$: $File$, $toolfile$: $File$

call templates

none

operation definitions

designbm **is** passive

designbm's **call pattern**

$$\circ$$

designbm's **pre- and post-conditions**

$$designbm(changes) \quad \triangleq$$

pre $True$
post $methodfile' = modified(methodfile, changes)$

---

where $modified(f, c)$ is a function which returns a file.
The other operations can be defined similarly.

7 Refinement

In OBM, a process model of an organisation is represented in different *levels*. Each level is a refinement of the previous level and an abstraction of the subsequent level. A process model is developed by gradually refining the levels until it reaches a level where the details of the activities are clearly defined.

At the top level of the development, the details of the processes are often unclear to the modeller, therefore an abstract object is usually used to model the abstract process. For instance, MUCS is modelled as an abstract object at the top level. At the next level MUCS may be decomposed into several sub-components, e.g. IPG, CNC, etc.. Each of the sub-components may be further decomposed in the subsequent levels of refinement.

Level i+1 Refinement of a Level i Component

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

Component Refinement

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

Coperation Refinement

- Replacing the level i coperations by an operation pattern defined in terms of level i+1 coperations. This replacement can be based on either the individual level i coperations or the sub operation patterns of the level i component.

An Example

7.1 Level 1 Refinement of Level 0 Component MUCS

Figure 1 illustrates the different levels of component refinement in MUCS.

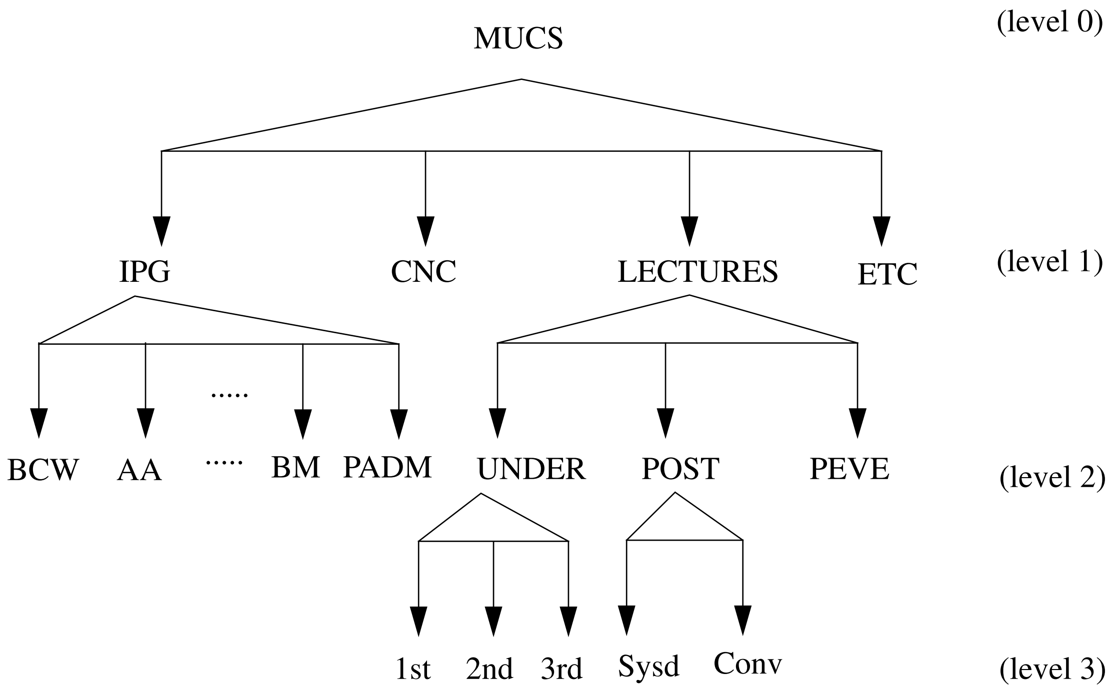

Figure 1: Levels of Component Refinement in MUCS

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 New (Level 1) Components

  Abstract Object IPG_1 The Interface Part:

  provides

  $$:ipgteach_1, ipgresearch_1, ipgothers_1$$

  requires

  $$LECTURES:teachPost_1$$

  operation pattern

  $$(ipgteach_1^+ \mid\mid ipgresearch_1^+ \mid\mid ipgothers_1^+); \circ$$

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**

$!teachPost_1; \circ$

$ipgteach_1$**'s pre- and post-conditions**

$ipgteach_1() \quad \triangleq$

pre $True$

post $True$

———————————-

$ipgresearch_1$ **is** passive

$ipgresearch_1$**'s call pattern**

$\circ$

$ipgresearch_1$**'s pre- and post-conditions**

$ipgresearch_1() \quad \triangleq$

pre $True$

post $True$

———————————-

Abstract Object $LECTURES_1$ The Interface Part:

provides

$:teachU_1, teachPost_1, teachPeve_1$

requires

$:none$

operation pattern

$$(teachU_1^+ \mid\mid teachPost_1^+ \mid\mid teachPeve_1^+); \circ$$

Abstract Object $LECTURES_1$ The Body Part:

call templates

$none$

coperation definitions

$teachU_1$ **is** passive

$teachU_1$**'s call pattern**

$\circ$

$teachU_1$**'s pre- and post-conditions**

$teachU_1() \quad \triangleq$

pre $True$
post $True$

———————————————-

$teachPost_1$ **is** passive

$teachPost_1$**'s call pattern**

$\circ$

$teachPost_1$**'s pre- and post-conditions**

$teachPost_1() \quad \triangleq$

pre $True$
post $True$

———————————————-

Other components can be defined similarly.

Coperation Refinement

- Replacements

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

Note that these coperation refinements are based on sub-operation patterns of MUCS.

7.2 Level 2 Refinement of Level 1 Component IPG

Component Refinement

- Replacement
  IPG is divided into a number of agent objects and step objects as follows:

$$IPG :: BCW, AA, PK, IR, PW, JSA, BM, PADM, TOOL.$$

- Definition of the New (Level 2) Components
  See sections 5 and 6 for the definitions of BCW and BM.

Coperation Refinement

- Replacements

$$ipgteach^{+} \triangleq bcwteach^{+} \mid\mid \ldots$$
$$ipgresearch^{+} \triangleq bcwresearch^{+} \mid\mid \ldots$$

7.3 Level 2 Refinement of Level 1 Component LECTURES

Component Refinement

- Replacement

$$LECTURES :: UNDER, POST, PEVE.$$

- Definitions of the New (Level 2) Components
Abstract Object *UNDER* The Interface Part:
provides

$$: teach1st, teach2nd, teach3rd$$

requires

$$: none$$

operation pattern

$$(teach1st^{+} \mid\mid teach2nd^{+} \mid\mid teach3rd^{+}); \circ$$

Abstract Object *POST* The Interface Part:
provides

$$: teachConv, teachSysd$$

requires

$$: none$$

operation pattern

$$(teachConv^{+} \mid\mid teachSysd^{+}); \circ$$

The body parts of the two abstract objects are trivial.

Coperation Refinement

- Replacements

$$teachU \triangleq teach1st \mid\mid teach2nd \mid\mid teach3rd$$
$$teachPost \triangleq teachConv \mid\mid teachSysd$$

Note that these coperation refinements are based on the individual coperations of LECTURES.

8 Propagations of the Refinement

Given a description of an organisational process, in which component A refers to component B by means of call templates. If component B is refined, the effect of this refinement should be propagated to component A. In this section we explain how the effect of refinement is propagated in OBM.

In the MUCS example described in the previous section, the level 2 component BCW_2 refers to the level 1 coperation $teachPost_1$ provided by the component $LECTURES_1$ via the call template $!teachPost_1$.

In the level 2 refinement of $LECTURES_1$, it is decomposed into three components $UNDER_2$, $POST_2$ and $PEVE_2$. The coperation $teachPost_1$ is replaced by $teachConv_2 \mid\mid teachSysd_2$. This information should be provided to the modeller of BCW_2 so that the modeller can decide how to change BCW_2 accordingly.

In OBM, whenever a coperation is changed or refined, the modellers of all the components which refer to this coperation are informed. No automatic change is performed. OBM only provides means to inform the feedbacks to the modellers.

In the MUCS example, the modeller of BCW_2 will revise its definition by redefining the call template and the call pattern of the operation $bcwteach$ as follows:

Agent Object BCW The Interface Part:

provides

$$:bcwteach, bcwresearch, bcwfindfound$$

requires

$$POST_2 : teachSysd_2$$
$$BM_2 : designbm_2$$
$$PADM_2 : designpadm_2$$

operation pattern

$$(bcwteach \ \oplus \ bcwresearch \ \oplus \ bcwfindfound)^+ ; \circ$$

Agent Object BCW The Body Part:

variables $hrspent$: $Hourspent$

call templates

$!teachSysd_2$	
prop	$func$
obj	$POST_2$
apar	$hours$

$!designbm_2$	
prop	$func$
obj	BM_2
apar	$hours$

$!designpadm_2$	
prop	$func$
obj	$PADBM_2$
apar	$hours$

operation definitions not changed

9 Composed Object

The previous section explained that an abstract object may be decomposed into a number of subcomponents that can be executed in parallel. This section explains how to compose a number of components together.

Representation

A composed object contains a number of components.

Semantics

The behaviour of a composed object is the concurrent executions of all the contained components. The temporal semantics is defined in appendix G.

Usage

A composed object is used to compose a number of components together.

An Example

Let $Leve1$ denote the composed object which contains IPG, CNC, $LECTURES$ and ETC.

Composed Object $Level1$

contains

$IPG, CNC, LECTURES, ETC$

10 Boxed Object

Boxed object is similar to composed object except that the internal coperations are hidden. Given a composition of a number of components, a coperation is said to be internal if it is only required by those components.

Representation

It contains a number of components and a list of coperations to be hidden.

Semantics

Similar to a composed object except that the internal coperations are not visible. The temporal semantics of a boxed object is defined in appendix K.

Usage

In the course of developing a model, an abstract object may be refined by decomposing it into a number of subcomponents. In order to justify the refinement, it is necessary to show that the composition of the subcomponents is consistent with the abstract object.

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.

A boxed object is used to compose level i+1 components together to show that the composition is consistent with the level i component. The difference between a composed object and a boxed object is that a boxed object hides the level i+1 internal coperations. So a boxed object is only used in relation to refinement whilst a composed object may be used to compose a number of components in any circumstance.

An Example

In $Level1$, the coperations provided by all the contained components are visible. However coperations provided by $LECTURES$ are only required by IPG and CNC, they are not required by any component outside $Leve1$. Let $Iops$ denote the list of coperations provided by the level 1 components, but not used to refine the $MUCS$ coperations in the **coperation refinement** part. The following representation hides these coperations from the composition.

Boxed Object $Boxlevel1$

contains

$IPG, CNC, LECTURES, ETC$

hidden coperations

$Iops$

11 Consistency of Refinement

Let

- A denote a level i component;
- r1, r2,..., rn denote some specifications of the level i+1 components;
- B denote the boxed object containing r1, r2, ..., rn;
- A' denote the same specification as A except that the level i coperations are replaced by their corresponding coperation patterns as defined in the **Coperation Refinement** part;

Since A' contains level i+1 coperations that may contain calls made to the internal coperations, it is necessary to hide these call, let

- $Iops$ denote the set of internal coperation names in level i+1 refinement;
- $comps$ denote the set of names of the level i+1 components;
- $hidecalls(Iops, comps)$ denote the predicate stating that the internal calls are hidden;

Since B contains more variables than A', the semantics of A' must be extended to include the definitions over these variables, let

- $vars$ denote the set of variables in $r1$, $r2$, ..., rn, but not in A';
- $anyvalue(vars)$ denote the predicate stating that variables in $vars$ can have any value;
- $[\![A']\!]_{Ca}$ and $[\![B]\!]_{Cb}$ denote the temporal semantics of A' and B; (Ca and Cb are the corresponding contexts, see appendix B.2 for details.)

In OBM, we define that r1, r2, ..., rn is a valid refinement of the level i component A if

$$\begin{array}{l} [\![B]\!]_{Cb} \Rightarrow \\ ([\![A']\!]_{Ca} \;\wedge \\ \Box hidecalls(Iops, comps) \;\wedge \\ \Box anyvalue(vars)) \end{array}$$

where

$$\begin{array}{l} hidecalls(Iops, comps) \triangleq \\ \forall o \in Iops \cdot \forall c \in comps \cdot \\ (send\text{-}c\text{-}o \Rightarrow True) \;\wedge \\ (rec\text{-}c\text{-}o \Rightarrow True) \end{array}$$

$$\begin{array}{l} anyvalue(vars) \triangleq \\ \forall x \in vars \cdot \exists v \cdot x = v) \end{array}$$

An Example

In this example we will justify that the level 1 refinement of $MUCS$ is a valid refinement. Let

- $MUCS'$ denote the same specification as $MUCS$ except that the coperations are replaced according to the level 1 coperation refinement; So

<u>Abstract Object</u> $MUCS'$ <u>The Interface Part:</u>

provides

:...

requires

:$none$

operation pattern

$$
\begin{array}{l}
(ipgteach_1^+ \mid\mid cncteach_1^+ \mid\mid etcteach_1^+ \mid\mid \\
ipgresearch_1^+ \mid\mid cncresearch_1^+ \mid\mid etcresearch_1^+ \mid\mid \\
ipgothers_1^+ \mid\mid cncothers_1^+ \mid\mid etcothers_1^+);\circ
\end{array}
$$

- $Boxlevel1$ denote the boxed object of the level 1 refinement;
- $LECTURESops$ denote the internal coperations of the level 1 refinement;
- $comps$ denote the set containing the names of IPG, CNC, $LECTURES$ and ETC;
- $vars$ denote the set of variables in the level 1 refinement, but not in $MUCS'$;
- Cb and Cm denote the contexts of $Boxlevel1$ and $MUCS'$;

The aim is to prove

$$
\begin{array}{l}
[\![Boxlevel1]\!]_{Cb} \Rightarrow \\
([\![MUCS']\!]_{Cm} \ \wedge \\
\Box hidecalls(LECTURESops, comps) \ \wedge \\
\Box anyvalue(vars))
\end{array}
$$

The following presents a rigorous reasoning.

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

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

If we expand the semantic definitions of the abstract objects IPG, CNC, $LECTURES$ and ETC according to the definition given in appendix C, the above formula can be expanded as follows:

$$
\begin{array}{l}
[\![Boxlevel1]\!]_{Cb} \triangleq \\
(([\![((ipgteach_1^+ \mid\mid ipgresearch_1^+ \mid\mid ipgothers_1^+);\circ)]\!]_{Ci} \ \wedge \ \Box atmostone(Ci)) \ \mathcal{C} \\
(idle(Ci) \ \mathcal{U} \ fin)) \ \wedge \\
(([\![CNC.pattern]\!]_{Cc} \ \wedge \ \Box atmostone(Cc)) \ \mathcal{C} \\
(idle(Cc) \ \mathcal{U} \ fin)) \ \wedge \\
(([\![((teachU_1^+ \mid\mid teachPost_1^+ \mid\mid teachPeve_1^+);\circ)]\!]_{Cl} \ \wedge \ \Box atmostone(Cl)) \ \mathcal{C} \\
(idle(Cc) \ \mathcal{U} \ fin)) \ \wedge \\
(([\![ETC.pattern]\!]_{Ce} \ \wedge \ \Box atmostone(Ce)) \ \mathcal{C} \\
(idle(Ce) \ \mathcal{U} \ fin)) \ \wedge \\
\Box(interleaven(Cb, Ci, Cc, Cl, Ce) \ \vee \ synchronisen(Ci, Cc, Cl, Ce)) \ \wedge \\
\Box hide(LECTURESops, comps)
\end{array}
$$

We then further expand the semantics of the operation patterns according to the definitions given in appendix F, we get

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

The above formula defines that

- $Boxlevel1$ 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 communications on internal coperations are hidden

Now consider the right hand side of the proof, if we expand $[\![MUCS']\!]$, we get the following formula,

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

Again expanding the semantics of the operation pattern, 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 palce 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.

12 Information Flow in OBM

In the MUCS example above information flow is not modelled. In this section we describe how information flow is modelled using this example.

Consider the two components *IPG* and *LECTURES* in level 1, IPG will need to supply information regarding the courses to LECTURES and LECTURES will supply examination results back to IPG and perhaps the faculty. (Faculty is not modelled in this report.)

In OBM we use explicit operations to model information flow. Therefore IPG provides an active operation that supplies the course details to LECTURES and LECTURES provides a passive operation that receives the course details. Similarly, LECTURES should provide an active supply operation to deliver the examination results.

It may seem that such an information flow may be modelled simply by using the parameter passing mechanism, i.e. the course detail is treated as the parameter of $teachU$, therefore information regarding the course detail is flew from IPG to LECTURES when $teachU$ is called by IPG. Reasons for not modelling it this way are explained below:

- Firstly, it is not the case that every time IPG calls $teachU$, the course detail should be supplied. In fact for each course, the detail should only be supplied to LECTURES once during each academic year, but $teachU$ must be performed many times (hopefully).
- Secondly, the course detail may not be supplied by the one who is going to call $teachU$, for instance, some course details may be supplied by the Electronic Engineering Department or the Psychology Department.
- Thirdly, the results of an activity are not necessarily going back to the original caller, in this example, the examination results should go to the faculty as well as IPG so that the degrees can be awarded.
- Fourthly, this approach gives a clear separation between information flow and activities.

In OBM, all three component constructs provide the following three kinds of coperations:

supply: supplies artifacts to other components, e.g. send course details. Supply operations are usually active operations.

receive: receives artifacts from other components, e.g. receive examination results. Receive operations are usually passive operations.

actions: performs activities, e.g. teaching undergraduates. Action operations may be either active or passive.

We can now re-define the interface part of IPG with the supply and receive operations incorporated as follows:

<u>Abstract Object</u> *IPG* <u>The Interface Part:</u>

provides

$supply$:$detailtoLECTURES$
$receive$:$resultfrLECTURES$
$action$:$ipgteach$, $ipgresearch$, $ipgothers$

requires

$LECTURES$:$teachPost$

operation pattern

$$(oneyearteach^{+} \parallel ipgresearch^{+} \parallel ipgothers^{+}); \circ$$

where

$$oneyearteach \triangleq (detailtoLECTURES; ipgteach^{+}; resultfrLECTURES)$$

A Concepts used in Modelling

Active Coperation An active coperation of a component is enabled by the conditions within the component.

Activity an activity is an elementary process step with no visible substructure.

Call Pattern A call pattern of a coperation defines the order of the calls made by the coperation to the required coperations.

Call Template A call template defines the call made to a required coperation.

Component A component is either an abstract object, an agent object or a step object.

Compound Operation A compound operation is an operation that may be refined in a subsequent levels of refinement.

Coperation A coperation is either a compound operation or an operation.

Operation Pattern An operation pattern of a component defines the order in which the execution of its coperations should conform.

Ordering Expression Ordering expression is a notation used to define both operation patterns and call patterns.

Passive Coperation The execution of a passive operation is only triggered by a caller.

Process A process is 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.

B Concepts Used in the Semantic Definitions

B.1 The Model

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

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. In order to distinguish between the component and the environment steps, each component is associated with a proposition, which we refer 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].

B.2 Concepts and Definitions

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).

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, including both user defined and specification variables.

event There are the following types of events:

- *start*: start the execution of an active coperation
- *end*: finish the execution of an active coperation
- *acc*: accept a call request and start the execution of a passive coperation
- *ret*: return the result of to the caller and finish the execution of a passive coperation
- *send*: send a request to a component to call one of its coperations.
- *receive*: receive the result return from the called coperation

event proposition An event proposition has the value *True* if the event is taking place, *false* otherwise.

history variable A history variable is a type of specification variable. It is used to record values received from calls made to other coperations.

specification variables One of the aims of OBM is to allow specifications to be written in a systematic way. In order to do so, each component has a set of variables used for the purposes of the specification and the subsequent temporal logic definition. These variables are called the specification variables in OBM. The followings are some examples of specification variables:

- *fpar-obj-op* is used to denote the parameter of the coperation *op* provided by the component *obj*.
- *ares-obj-op* is used to denote the result of *op*.
- *fres-*$O1$*-*$op1$ is used to contain the results returned from the calls made to $op1$ (which is provided by $O1$);

C Semantics of Abstract Object

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. Let

- Abs denote the name of an abstract object;
- π_{abs} denote the component proposition of Abs;
- X denote the context in which the semantics of Abs is defined;
- $[\![Abs]\!]_X$ denote the semantics of A in context X;

Given the abstract object
Abstract Object Abs The Interface Part:

provides

$supply : cop, opls1$
$receive : opls2$
$action : opls3$

requires

$O1 : op1$
$O2 : op2$
$\ldots : \ldots$

operation pattern

$pattern$

The semantics of Abs is

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

$pattern$ is defined using ordering expression. The semantics of ordering expression is define in appendix F. $\Box atmostone(X)$ states that at most one event may take place at any time. This constraint is to ensure that parallel activities are interleaved.The definition of $atmostone(X)$ is defined in appendix L.

D Semantics of Agent Object

The semantics of an agent object is determined by the semantics of its operation pattern. Since the $||$ operator is not allowed in the operation pattern of an agent object, there can only be at most one event taking place at any time. Let

- Age denote the name of an agent object;
- π_{age} denote the component proposition of Age;
- Y denote the context in which the semantics of Age is defined;

Given the agent object
Agent Object Age The Interface Part:

provides

$supply$:aop, $opls1$
$receive$:pop, $opls2$
$action$:$opls3$

requires

$o1$:$op1$
$o1$:$op2$
... :...
oi :opi

operation pattern

$pattern$

The semantics of Age is

$$[\![Age]\!]_Y \triangleq [\![pattern]\!]_Y$$

E Semantics of Step Object

The semantics of a step object is similar to agent object. Let

- St denote the name of an agent object;
- π_{st} denote the component proposition of St;
- Z denote the context in which the semantics of St is defined;

Given the step object
Step Object St The Interface Part:

provides

$$\begin{array}{l} supply : opls1 \\ receive : opls2 \\ action : opls3 \end{array}$$

requires

$$\begin{array}{l} Com1 : rls1 \\ Com2 : rls2 \\ \ldots \quad : \ldots \end{array}$$

operation pattern

$$pattern$$

The semantics of St is

$$[\![St]\!]_Z \triangleq [\![pattern]\!]_Z$$

F Ordering Expressions

F.1 Syntax

$\circ$**:** end of computation

op**:** an operation name

$call$**:** a call template name

$P1; P2$**:** $P1$ followed by $P2$

$P1 \oplus P2$**:** $P1$ or $P2$

$P1 \otimes P2$**:** $P1$ followed by $P2$ or $P2$ followed by $P1$

$P1 \mid\mid P2$**:** $P1$ in parallel with $P2$

P^+**:** one or more occurrences of P

P^***:** zero or more occurrences of P

$\forall x \cdot P(x)$**:** $P(x)$ occurs for all x

$\exists x \cdot P(x)$**:** $P(x)$ occurs for some x

F.2 semantics of Ordering Expression

Let $[\![P]\!]_C$ denote the temporal semantics of P in the context C.

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

$$[\![op]\!]_C \triangleq see\ the\ semantics\ of\ a\ coperation$$

$$[\![call]\!]_C \triangleq see\ the\ semantics\ of\ a\ call\ template$$

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

$$\begin{array}{l} [\![P1 \mid\mid P2]\!]_C \triangleq \\ ([\![P1]\!]_{C1}\ \mathcal{C}\ (idle(C1)\ \mathcal{U}\ fin)) \ \wedge \\ ([\![P2]\!]_{C2}\ \mathcal{C}\ (idle(C2)\ \mathcal{U}\ fin)) \end{array}$$

where $C1$ and $C2$ are the contexts for $P1$ and $P2$. Ci contains π_i, Pi, the names of all the operations used in Pi and the names of all the variables used in Pi. In the above definition, Pi is extended with idle steps so that sequences of the extended $P1$ and $P2$ are of the same length.

$$[\![P1 \ \oplus \ P2]\!]_C \triangleq [\![P1]\!]_C \ \wedge \ \neg[\![P2]\!]_C \ \vee \ \neg[\![P1]\!]_C \ \wedge \ [\![P2]\!]_C$$

$$[\![P^+]\!]_C \triangleq [\![P1]\!]_C \ \mathcal{C}^* \ fin$$

$$[\![\forall x \cdot P(x)]\!]_C \triangleq \forall x \cdot [\![P(x)]\!]_C$$

$$[\![\exists x \cdot P(x)]\!]_C \triangleq \exists x \cdot [\![P(x)]\!]_C$$

The following two operators can be defined in terms of the above.

$$P^* \triangleq P^+ \ \oplus \ fin$$

$$\begin{array}{l} P1 \ \otimes \ P2 \ \otimes \ ... \ \otimes \ Pi \triangleq \\ P1; (P2 \ \otimes \ P3 \ \otimes \ ... \ \otimes \ Pi) \ \oplus \\ P2; (P1 \ \otimes \ P3 \ \otimes \ ... \ \otimes \ Pi) \ \oplus \\ ... \\ Pi; (P1 \ \otimes \ P2 \ \otimes \ ... \ \otimes \ P(i\text{-}1)) \end{array}$$

G Semantics of Compound Operation

There are two type of compound operations: active and passive.

G.1 Active Compound Operation

The semantics of an active compound operation is that the component providing this compound operation may "perform" a few idle steps and then start the execution of this operation. During the execution, the operation may call other coperations. Idle steps may be "perform" at any time. When the operation starts the execution, the pre-condition must be true and the post-condition must be true when the operation finishes. Let

- *cop* denote a compound operation;
- *Abs* denote the name of the abstract object providing *cop*;
- π_{abs} denote the component proposition of *Abs*;
- X denote the context in which the semantics of *Abs* is defined;
- *start-abs-cop* and *end-abs-cop* denote the event propositions of the starting and finishing events of *cop* respectively;
- *fres-O*1*-op*1, *fres-O*2*-op*2, ..., *fres-Oi-opi* denote the history variables used to record the values received from the calls made to *op*1, *op*2, ..., *opi* of *O*1, *O*2, ..., *Oi*;

Given the compound operation *cop*

cop **is** active

cop'**s call pattern**

call.pattern

cop's **pre- and post-conditions**

$$cop() \triangleq$$

pre $precond$
post $postcond(\mathit{fres}\text{-}O1\text{-}op1, \mathit{fres}\text{-}O2\text{-}op2, ..., \mathit{fres}\text{-}Oi\text{-}opi)$

———————————————-

The semantics of cop is

$$[\![cop]\!]_X \triangleq starting(X, cop, precond)\ \mathcal{C}\ [\![call.pattern]\!]_X\ \mathcal{C}\ ending(X, cop, postcond)$$

The semantics of call pattern is given in appendix F.

$$\begin{array}{l} starting(X, cop, precond) \triangleq idle(X)\ \mathcal{U} \\ (\pi_{abs}\ \wedge\ start\text{-}abs\text{-}cop\ \wedge \\ reset.history(X)\ \wedge \\ precond\ \wedge\ fin) \end{array}$$

The definition of $idle(X)$ is given in appendix L. $reset.history(X)$ is a predicate which states that all the history variable for this operation are initialised. Note that, both the component proposition and the event proposition are needed to specify an event. The component proposition only indicates who is performing an event, but does not indicate which event. An event proposition only indicates which event is taking place, but does not indicate who is performing the event.

$$\begin{array}{l} ending(X, cop, postcond) \triangleq \\ idle(X)\ \mathcal{U} \\ (\pi_{abs}\ \wedge\ end\text{-}abs\text{-}cop\ \wedge \\ \exists hv1, hv2, ..., hvi \cdot \\ \mathit{fres}\text{-}O1\text{-}op1 = hv1\ \wedge \\ \mathit{fres}\text{-}O2\text{-}op2 = hv2\ \wedge \\ ...\ \wedge \\ \mathit{fres}\text{-}oi\text{-}opi = hvi\ \wedge \\ \bigcirc(postcond(hv1, hv2, ..., hvi)\ \wedge\ fin)) \end{array}$$

Note that we cannot put the history variables in $postcond$ because they will be interpreted in the next ($\bigcirc$) state. The values of the history variables are defined in the call templates, see appendix I.

G.2 Passive Compound Operation

The semantics of a passive compound operation is similar to an active compound operation. The differences are: firstly the event propositions $start\text{-}abs\text{-}cop$ and $end\text{-}abs\text{-}cop$ are replaced by $acc\text{-}abs\text{-}cop$ and $ret\text{-}abs\text{-}cop$ respectively; secondly a passive compound operation may take parameter and return result, so the pre-condition and the post-condition are defined in terms of the parameter and the result as well as the history variables. In addition to the names used in defining the semantics of an active compound operation, let

- $acc\text{-}abs\text{-}cop$ and $ret\text{-}abs\text{-}cop$ denote the event propositions of the accepting and returning events of cop respectively;
- $\mathit{fpar}\text{-}abs\text{-}cop$ be the specification variable containing the parameter of cop;
- $ares\text{-}abs\text{-}cop$ be the specification variable containing the result of cop;

cop **is** passive

cop's **call pattern**

$$call.pattern$$

cop's **pre- and post-conditions**

$$cop() \triangleq$$

pre $precond(fpar\text{-}abs\text{-}cop)$
post $postcond(fpar\text{-}abs\text{-}cop, fres\text{-}O1\text{-}op1, fres\text{-}O2\text{-}op2, ..., fres\text{-}Oi\text{-}opi,\ ares\text{-}abs\text{-}cop)$

——————————————-

The semantics is

$$[\![cop]\!]_X \triangleq accepting(X, cop, precond)\ \mathcal{C}\ [\![call.pattern]\!]_X\ \mathcal{C}\ returning(X, cop, postcond)$$

$$\begin{array}{l} accepting(X, cop, precond) \triangleq \\ idle(X)\ \mathcal{U} \\ (\pi_{abs}\ \wedge\ acc\text{-}abs\text{-}cop\ \wedge \\ reset.history(X)\ \wedge \\ precond(fpar\text{-}abs\text{-}cop)\ \wedge\ fin) \end{array}$$

$$\begin{array}{l} returning(X, cop, postcond) \triangleq \\ idle(X)\ \mathcal{U} \\ (\pi\ \wedge\ ret\text{-}abs\text{-}cop\ \wedge \\ \exists pv, hv1, hv2, ..., hvi \cdot fpar\text{-}abs\text{-}cop = pv\ \wedge \\ fres\text{-}O1\text{-}op1 = hv1\ \wedge \\ fres\text{-}O2\text{-}op2 = hv2\ \wedge \\ ...\ \wedge \\ fres\text{-}Oi\text{-}opi = hvi\ \wedge \\ \bigcirc(postcond(pv, hv1, hv2, ..., hvi, ares\text{-}age\text{-}pop)\ \wedge\ fin)) \end{array}$$

H Semantics of Operation

There are also two types of operations: active and passive. The difference between a compound operation and an operation is that the definition of an operation include pre- and post-conditions.

H.1 Semantics of Active Operation

The semantics of an active operation is similar to an active compound operation. The difference is that an active operation is defined in term of the user defined variables as well as the parameters, the result and the specification variables. Let

- *aop* denote an active operation provided by the agent object *Age* as defined in appendix D,
- Y denote the context in which the semantics of *Age* is defined;
- *start-age-aop* and *end-age-aop* denote the event propositions of the starting and finishing events of *aop* respectively;
- s denote the user defined variable of *Age*; execution of *aop*;
- *fres-O*1*-op*1, *fres-O*2*-op*2, ..., *fres-Oi-opi* denote the history variables used to record the values received from the calls made to *op*1, *op*2, ..., *opi* of *O*1, *O*2, ..., *Oi*;

Given the active operation *aop*

aop **is** active

aop'**s call pattern**

$$call.pattern$$

aop'**s pre- and post-conditions**

$$aop() \triangleq$$

pre $precond(s)$
post $postcond(s, s', \mathit{fres}\text{-}O1\text{-}op1, \mathit{fres}\text{-}O2\text{-}op2, ..., \mathit{fres}\text{-}Oi\text{-}opi)$

——————————————-

s and s' contain the values before and after the execution of *aop* respectively. The semantics of *aop* is

$$[\![aop]\!]_Y \triangleq staring(Y, aop, precond)\ \mathcal{C}\ [\![call.pattern]\!]_Y\ \mathcal{C}\ ending(Y, aop, precond)$$

$$\begin{array}{l} staring(Y, aop, precond) \triangleq \\ idle(Y)\ \mathcal{U} \\ (\pi \ \wedge\ start\text{-}age\text{-}aop \ \wedge\ reset.history(Y) \ \wedge\ precond(s) \ \wedge\ \mathit{fin}) \end{array}$$

$$\begin{array}{l} ending(Y, aop, precond) \triangleq \\ idle(Y)\ \mathcal{U} \\ (\pi \ \wedge\ end\text{-}age\text{-}aop \ \wedge \\ \exists v, hv1, hv2, ..., hvi \cdot \\ s = v \ \wedge \\ \mathit{fres}\text{-}O1\text{-}op1 = hv1 \ \wedge \\ \mathit{fres}\text{-}O2\text{-}op2 = hv2 \ \wedge \\ ... \ \wedge \\ \mathit{fres}\text{-}Oi\text{-}opi = hvi \ \wedge \\ \bigcirc(\exists v' \cdot s = v' \ \wedge\ postcond(v, v', hv1, hv2, ..., hvi) \ \wedge\ \mathit{fin})) \end{array}$$

H.2 Semantics of Passive Operation

The semantics of a passive operation is similar to a passive compound operation. The difference is that a passive operation may be defined in terms of the user defined variables as well as the parameter, the result and the specification variable. Let

- *pop* denote an operation provided by the agent object *Age* as defined in appendix D,
- Y denote the context in which the semantics of *Age* is defined;
- *acc-age-pop* and *ret-age-pop* denote the event propositions of the accepting and returning events of *pop* respectively;
- s denote the user defined variable of *Age*;
- *fpar-age-pop* be the specification variable containing the parameter of *pop*;
- *ares-age-pop* be the specification variable containing the result of *pop*;

Given the operation *pop*

pop **is** passive

pop's **call pattern**

$$call.pattern$$

pop's **pre- and post-conditions**

$$pop() \triangleq$$

pre $precond(s, fpar\text{-}age\text{-}pop)$
post $postcond(s, s', fpar\text{-}age\text{-}pop, fres\text{-}O1\text{-}op1, fres\text{-}O2\text{-}op2, ..., fres\text{-}Oi\text{-}opi, ares\text{-}age\text{-}pop)$

———————————————-

The semantics is

$$[\![pop]\!]_Y \triangleq accepting(Y, pop, precond)\ \mathcal{C}\ [\![call.pattern]\!]_Y\ \mathcal{C}\ returning(Y, pop, postcond)$$

$$\begin{array}{l} accepting(Y, pop, precond) \triangleq \\ idle(Y)\ \mathcal{U} \\ (\pi_{age}\ \wedge\ acc\text{-}age\text{-}pop\ \wedge \\ reset.history(Y)\ \wedge \\ precond(s, fpar\text{-}age\text{-}pop)\ \wedge\ fin) \end{array}$$

$$\begin{array}{l} returning(Y, pop, postcond) \triangleq \\ idle(Y)\ \mathcal{U} \\ (\pi_{age}\ \wedge\ ret\text{-}age\text{-}pop\ \wedge \\ \exists v, pv, hv1, hv2, ..., hvi \cdot\ \wedge \\ s = v\ \wedge \\ fpar\text{-}age\text{-}pop = pv\ \wedge \\ fres\text{-}O1\text{-}op1 = hv1\ \wedge \\ fres\text{-}O2\text{-}op2 = hv2\ \wedge \\ ...\ \wedge \\ fres\text{-}Oi\text{-}opi = hvi\ \wedge \\ \bigcirc(\exists v' \cdot s = v'\ \wedge\ postcond(v, v', hv1, hv2, ..., hvi, ares\text{-}age\text{-}pop)\ \wedge\ fin) \end{array}$$

I Call Template

There are two types of calls: function and procedure.

I.1 Semantics of Function Call

The semantics of a function 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 coperation. Let

- the following template denote the call template of the call made by Age as define in appendix D;
- Y denote the context in which Age is defined;
- $send\text{-}O1\text{-}op1$ and $rec\text{-}O1\text{-}op1$ denote the event propositions of the send and receive events of this call;
- $apar\text{-}O1\text{-}op1$ denote the specification 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 specification 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 specification index variable recording the number of times that $op1$ is called by the coperation; $occno\text{-}O1\text{-}op1$ is used to index into $apar\text{-}O1\text{-}op1$ and $fres\text{-}O1\text{-}op1$.

Given the call template,

$!op1$	
prop	$func$
obj	$o1$
apar	v

The semantics is

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

$$\begin{array}{l} sending(Y, O1, op1, v) \triangleq \\ (idle(Y) \ \wedge\ \exists n \cdot .occno\text{-}O1\text{-}op1 = n)\, \mathcal{U} \\ (\pi_{age} \ \wedge\ send\text{-}O1\text{-}op1 \ \wedge \\ apar\text{-}O1\text{-}op1(n) = v) \ \wedge\ fin) \end{array}$$

$$\begin{array}{l} receiving(Y, O1, op1) \triangleq \\ (idle(Y) \ \wedge\ \exists n \cdot occno\text{-}O1\text{-}op1 = n)\, \mathcal{U} \\ (\pi_{age} \ \wedge\ rec\text{-}O1\text{-}op1 \ \wedge \\ \exists r \cdot fres\text{-}O1\text{-}op1(n) = r \ \wedge \\ occno\text{-}O1\text{-}op1 = n + 1 \ \wedge \\ fin) \end{array}$$

The value of r is bound to the actual value returned by $op1$ when the component Age is composed with the component $O1$. See the definition of $synchronise$ in appendix L.

I.2 Semantics of Procedure Call

In a procedure call, the component making the call may also idle for a few steps before it makes the calls, and then it may idle for some more steps. It does not receive any result.

To be completed.

J Semantics of Composed Object

The behaviour of a composed object is defined as the set of all the possible execution sequences of the composed object. Each step of an execution sequence of a 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. Let

- $Cobj$ denote the name of a composed object containing two subcomponents $C1$ and $C2$;
- W, $W1$ and $W2$ denote the contexts of $Cobj$, $C1$ and $C2$ respectively;
- π, π_1 and π_2 denote the component propositions of $Cobj$, $C1$ and $C2$ respectively;

where

$W1$ contains $\{\pi_1, C1, pops1, rops1, vars1\}$, $pops1$ is the set names of the coperations provided by $C1$, $rops1$ is the set of names of coperations required by $C1$, $vars1$ is the set of names of the specification variables for $C1$ and the user define variables if $C1$ is not an abstract object.

$W2$ contains $\{\pi_2, C2, pops2, rops2, vars2\}$, $pops2$ is the set names of the coperations provided by $C2$, $rops2$ is the set of names of coperations required by $C2$, $vars2$ is the set of names of the specification variables for $C2$ and the user define variables if $C2$ is not an abstract object.

W contains $\{\pi, Cobj, pops1 \cup pops2, rops1 \cup rops2, vars1 \cup cars2\}$.

The semantics of $Cobj$ is

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

$C1$ and/or $C2$ are extended with idle steps so that they have the same length. The definition of $interleave(W, W1, W2)$ and $synchronise(W1, W2)$ are given in appendix L.

The semantics of a composed object containing more than two components can be based on the semantics given above easily.

K Semantics of Boxed Object

The semantics of a boxed object containing several components is defined in the same way as a composed object by interleaving the events of the contained components. The difference is that in a composed object, all the events are visible, whilst in a boxed object 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;
- π, π_1 and π_2 denote the component propositions of $Bobj$, $C1$ and $C2$ respectively;
- $Iops$ denote the internal coperations of $Bobj$;

where $W1$, $W2$ and W have the same contents as above.

Given the boxed object

<u>Boxed Object</u> $Bobj$

contains

$C1, C2$

hidden coperations

$Iops$

The semantics of $Bobj$ is

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

where Cs is the names of the contained components.

$$
\begin{array}{l}
hide(Cs, Iops) \triangleq \\
\forall c \in Cs \cdot \forall o \in Iops \cdot \\
(send\text{-}c\text{-}o\ \wedge\ acc\text{-}c\text{-}o \Rightarrow True\ \wedge \\
(rec\text{-}c\text{-}o\ \wedge\ ret\text{-}c\text{-}o \Rightarrow True
\end{array}
$$

L Other Definitions

$$
\begin{array}{l}
atmostone(C) \triangleq \\
\forall r, t \in OPs \cdot \\
\exists e1, e2, o1, o2 \cdot \\
((e1\text{-}r\text{-}o1\ \wedge\ e2\text{-}t\text{-}o2) \Rightarrow (e1 = e2\ \wedge\ r = t\ \wedge\ o1 = o1))
\end{array}
$$

where OPs is the set of coperation names provided and required in C.

$$
\begin{array}{l}
interleave(W, W1, W2) \triangleq \\
\pi = (\pi_1\ \vee\ \pi_2)\ \wedge\ \neg(\pi_1\ \wedge\ \pi_2)\ \wedge \\
atmostone(W)
\end{array}
$$

where π, π_1 and π_2 are the component propositions in W, $W1$ and $W2$ respectively.

Interleaving among N components can be easily defined based on the same idea.

$$
\begin{array}{l}
synchronise(W1, W2) \triangleq \pi_1 \ \wedge \ \pi_2 \ \wedge \\
\forall o \in SOPs \cdot \\
(send\text{-}c\text{-}o \ \wedge \ acc\text{-}c\text{-}o \ \wedge \\
\exists n \cdot occno\text{-}c\text{-}o = n \ \wedge \\
\exists pv \cdot apar\text{-}c\text{-}o(n) = pv \ \wedge \ fpar\text{-}c\text{-}o = pv) \\
\vee \\
rec\text{-}c\text{-}o \ \wedge \ ret\text{-}c\text{-}o \ \wedge \\
\exists n \cdot occno\text{-}c\text{-}o = n \ \wedge \\
\exists rv \cdot ares\text{-}c\text{-}o = rv \ \wedge \ fres\text{-}c\text{-}o(n) = rv)
\end{array}
$$

where π_1 and π_2 are components propositions in $W1$ and $W2$; c is either the component name in $W1$ or $W2$; $SOPs$ is the set of coperation names that must be synchronised between the two components; let $OPs1$ and $OPs2$ denote the sets of coperation names in $W1$ and $W2$, OPs is the intersection of $OPs1$ and $OPs2$.

Synchronisation among N components can be easily defined based on the same idea.

$$
\begin{array}{l}
idle(X) \triangleq \\
\neg \pi \ \wedge \\
\forall e \cdot \\
\forall o \in OPs \cdot \neg e\text{-}r\text{-}o \ \wedge \\
\forall var \in VARs \cdot \bigcirc var = var \ \wedge
\end{array}
$$

where π is the component proposition in X; r is the component name in X; OPs is the sets of coperation names in X; $VARs$ is the set of user defined variable names and specification variable names in X.

References

[1] 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.

[2] H.Barringer, R.Kuiper and A.Pnueli, Now you may compose temporal logic specifications, Proc, 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", Proceedings of the First International Conference on the Software Process, October, 1991. !!!change to second!!!

[4] 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.

[5] C. Fernström, "PROCESS WEAVER: Adding Process Support to UNIX", To appear in Proceedings of the Second International Conference on the Software Process, Berlin, Germany, February, 1993.

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

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

[8] J. Lonchamp, "A Structured Conceptual and Terminological Framework for Software Process Engineering", Proceedings of the second International Conference on Software Process 1993.

[9] 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.

[10] 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.

[11] 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.

[12] J.Sa, B.C. Warboys, "A Formal Description of the ISPW-6 Software Process Example", Working paper, Department of Computer Science, University of Manchester, 1993.