# Mapping Enterprise Events to the CORBA Notification Service

Shelby Abraham, Keith Duddy, Michael Lawley, Zoran Milosevic,Kerry Raymond, Andrew Wood

CRC for Enterprise Distributed Systems Technology (DSTC)
University of Queensland, Brisbane, Queensland 4072, Australia

E-mail: {shelby, dud, lawley, zoran, kerry, woody}@dstc.edu.au

## Abstract

*To satisfy the need for a wide range of enterprise modelling choices, it is necessary to have a rich set of process-based and role-based modelling concepts. It is also necessary to have a sound and expressive business event model that includes a flexible way of associating business events with business processes and roles.*

*However, the ability to construct detailed and expressive enterprise models is not useful in a business context without some means of realising the model in technology.*

*This paper introduces the basic modelling concepts from the DSTC's UML Profile for Enterprise Distributed Object Computing before describing in detail the profile's business event model. The paper then describes how this event model can be mapped in a straightforward manner to implementations using the OMG's CORBA Notification Service.*

*This mapping can then be used as the basis for building automated tools that support generating implementations of systems from high level enterprise models.*

## 1. Introduction

As distributed object technologies mature and become more widely deployed, there is an increasing need for rich modelling languages to be able to describe the kinds of enterprise-wide applications that these technologies facilitate. In particular, it is no longer sufficient to provide just an information or computational specification of a system. Rather, for such enterprise systems, there is a recognised need to be able to describe such enterprise-level aspects of the system as the business processes, entities, roles, and events that are involved.

In addition, it is not sufficient to simply be able to describe such aspects of a system. It is important that there be a clear mapping from such enterprise models to distributed object technologies that will be used in the implementation of systems.

Our approach to providing support for the description of such systems is based on the introduction of modelling concepts that represent dynamic, structural and policy aspects of enterprises. The goals we have in producing such a modelling language are to provide:

- a small but powerful set of enterprise modelling concepts

- an expressive graphical notation

- a basis for automatic generation of component-based enterprise systems.

This paper focuses on how our business event modelling concepts can be used to specify aspects of a system implementation involving CORBA and CORBA services. In particular we discuss how the CORBA Notification Service can be used to implement the asynchronous communication between model elements.

This paper begins by introducing the key EDOC modelling concepts developed at the DSTC. This work has formed the basis of our initial submission to the Object Management Group (OMG) activity to standardise a UML Profile for Enterprise Distributed Object Computing [4]. Section 2 describes our notions of business process, business roles and business events. It outlines how business events are related to elements of the business process and business role models. Section 3 then describes in detail the business event model from the DSTC's UML Profile for EDOC submission. Section 4 introduces the CORBA Notification Service before Section 5 describes in detail a mapping from the business event model to the Notification Service. To provide some more practical illustration, Section 6 works through an example showing how the business events from a fragment of an enterprise model would be realised by the Notification Service. Finally, Section 7 ends the paper by expressing some conclusions from our work.

Throughout this paper we illustrate the concepts we wish to convey using as an example a system for producing the

proceedings for an academic conference. In this example, papers are submitted by Authors to a Programme Committee for reviewing. The Programme Committee allocates papers to reviewers who produce reviews of the papers. Based on these reviews, a number of papers are chosen for publication.

## 2. Key EDOC Modelling Concepts

This section introduces the key models we use to describe various aspects of Enterprise Distributed Object Computing (EDOC) systems. These models provide direct support for describing business processes, business roles, business entities, and business events. As such, they are well suited for forming the basis of extensions to the UML [2] so as to meet the EDOC requirements specified in the OMG UML Profile for EDOC RFP [3].

In order to provide a set of self-contained concepts suitable for practical enterprise modelling, we have integrated ideas from areas such as workflow systems, requirements engineering, and the ODP Enterprise Language Standard [1]. Our submission to the OMG and the work described in [11] focuses heavily on the expression of the modelling constructs in UML. We use techniques similar to those described in [15] to realise our models in UML.This paper is a companion to [11, 12] as it focuses on how Enterprise models can be mapped to technologies to support implementations of systems.

While this section introduces and positions our definitions of business processes and business roles, it should be kept in mind that this paper is about the specification and mapping of business events. Throughout this section we convey the concept of how business events may be attached to the various model elements of the business process and business role models so as to provide support for asynchronous messaging between different parts of the model. We also introduce our graphical notation for representing these concepts based on the notation used in [13].

### 2.1. Business Process Modelling

In our model, a business process is represented as a dependency graph of business tasks linked in a specific way to achieve some particular objective. A business process can be control-driven or data-driven, or both depending on the nature of the dependencies between tasks, and our model provides a rich semantics for expressions of these task dependencies. However, the smallest granularity of steps in a process, known as Application Tasks, are seen from the process viewpoint as discrete operations which produce results in a synchronous manner. The primary function of the Flows which enact the dependencies between these Tasks is to sequence them, or act as synchronisations for parallel

Tasks. The consequence of this is that our process model does not allow asynchronous results to be propagated from Tasks using Flows. This functionality is provided by the Event Model.

Our model makes provision for an association of business tasks with business roles to execute them.

Although our business process model uses concepts found in many workflow systems, nonetheless we view workflow as an IT solution to automating and managing business processes, mostly focusing on the execution semantics. Instead, in our approach, we have attempted to come up with a succinct business process model that encompasses different workflow execution semantics. In addition, we consider business processes in the context of other business determinants, such as business roles, business entities and business events resulting in an emphasis on business semantics over computational semantics. Our submission to the OMG [4] describes a number of ways of implementing these business process concepts using CORBA interfaces, one aspect of which includes the OMG's Workflow Management Facility specification [5].

A detailed description of our Business Process model including details of the semantics and notation that have been defined can be found in [4]. While the full details of this model are not suitable for this to paper, it is necessary to include a brief introduction to some of the business process model concepts which are referred to in our discussion of the Business Event Model that follows.

The basic building block of the business process model is the Task. A Task defines a self-contained unit of work in terms of its inputs, its function, and its outputs. Tasks can be divided into two subtypes: Simple Tasks, and Compound Tasks. A Simple Task refers to an activity that is carried out without further refinement at this level of abstraction, while Compound Tasks contain a set of statically-defined tasks that are co-ordinated to perform some larger scale activity.

A business process is described by a Compound Task and the behaviour of the business process is thus defined by the aggregate behaviour (the behaviour of all the contained Tasks - both Simple and Compound) of the Compound Task describing the business process. Examples of business processes are "issuing a Call for Papers for a conference" and "allocating papers to reviewers".

Tasks have input sets, output sets, and exceptions (which are subtypes of output set) as illustrated using our own notiation in Figure 1. An input set models the information required to commence execution of a Task as a set of name-value pairs known as inputs. An output set represents a possible outcome of the execution of a Task; it serves both as an indication that the Task has terminated and provides a set of outputs (name-value pairs) associated with that outcome. An exception indicates that the Task has terminated having failed to perform its function; it may have a set of
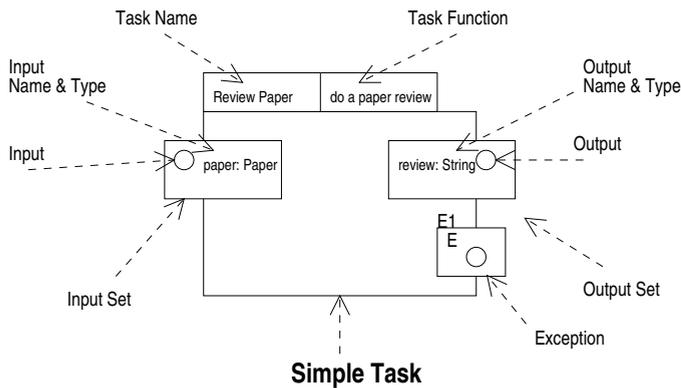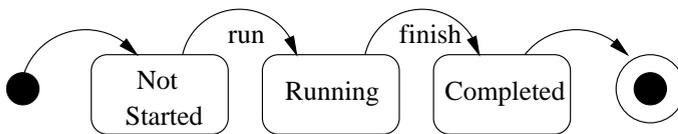
**Figure 1. Example of a Simple Task**



**Figure 2. Task State Machine**



**Figure 3. Example of a Compound Task**

outputs associated with that failure. There can be multiple input sets, output sets, and exceptions, modelling alternative circumstances in which the Task may start, complete or fail.

Figure 2 shows a state machine representation of the possible run-time states for a task. Changes in the state of execution of a task as indicated by State machine transitions can be triggers for the emission of business events.

As mentioned above, Compound Tasks contain Tasks. They also contain flows and control points as illustrated in Figure 3. Flows and control points are used to coordinate the execution of Tasks within a Compound Task. We define two types of flow: data flow which propagates data between inputs and output; and control flow which indicates a causal dependency.

In our business event model outlined in section 3, events can carry payload. One of the by-products this is the ability to receive an event and assign the contents of the event to or from an input or output. Within an enterprise model, the values of inputs/outputs are normally transmitted via data flows (subject to the hierarchical structure of Compound Tasks). Events enable the values of inputs and outputs to be transmitted or received from beyond the scope of the containing compound Task (or even the enterprise model itself). Such business event sources and sinks behave similarly to the sources and sinks of data flows for the purposes of propagation of data and determining whether an input/output set is satisfied.
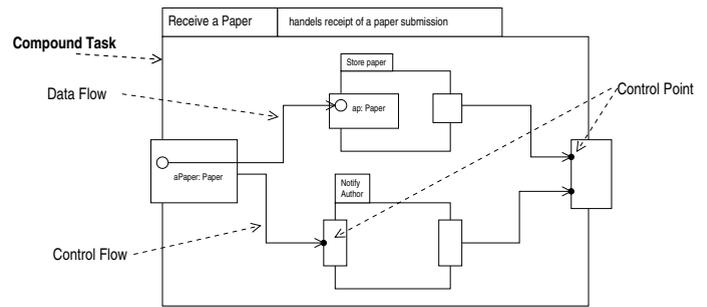
## 2.2. Business Roles and their Support

We believe that Business Roles should be described as fragments of behaviour of the enterprise - those that can then be fulfilled by specific business entities. The separation of the concepts of business entities and business roles enables the specification of the enterprise in terms of behaviour and not in terms of business entities. This modelling approach provides flexibility in assigning business entities to business roles; one business entity can fill more than one role and one role can be filled by different entities, as long as behaviour of such an entity is compatible with the behaviour of that business role. This allows flexibility in changing the assignments of business entities to business roles as new policy or resource requirements may demand.

Such treatment of business roles also provides a basis for flexible assignment of the performers of actions in a dependency graph of business tasks forming a business process. In fact, a business role can be regarded as a collection of actions that are involved in performing one or more business tasks and the grouping of these actions corresponds to the definition of business roles. Our notion of Role is similar to the OORAM concept of Role [14].

## 2.3. Business Events and Actions of Interest

It is necessary to nominate exactly what actions may be of interest for each kind of model element. All events are based on actions of interest, but not all actions of interest will be used as events. Note that it is theoretically possible to emit/receive all actions of interest as events, but the number of events becomes overwhelming in practice hence our requirement that event sources and sinks are explicitly identified in the enterprise model.

The commencement of execution of a Task is an action of interest as is its termination. Input sets and output sets have two actions of interest: becoming satisfied (once all its inputs and attached control points have been satisfied), and becoming enabled (when it is chosen by the Task for its commencement). The only action of interest for an input or

**Figure 4. UML Diagram of the Business Event Model**



**Figure 5. Example of the Event Notation**

output is to be assigned a value. The transmission of data or control along a flow is an action of interest.

For business roles and business entities, there are a number of actions of interest, including their creation and termination, the assignment or change of an attribute value, and the invocation or return of a method. The performing of a Task by a performer role and the use of artifact role in a Task are actions of interest. The assigning and de-assigning of business entities to fill business roles are also actions of interest.

## 3. Modelling Business Events

The business event model provides for the attaching of sources and sinks of asynchronous broadcast events to various EDOC Model Elements to allow them to expose their actions or state changes to other parts of the enterprise. Restrictions can be placed on how widely the events are broadcast. Figure 5 shows an annotated example of the notation we use to show the attachment of event sources and sinks to model elements - in this case Tasks.

The notion of an event itself is not defined precisely in our model; it is sufficient to model only the business event type. A business event type explicitly exposes an action that has a significant business semantics with respect to the enterprise being modelled or its environment.

Business event sources emit events, while business event sinks receive events. For example, withdrawing a submitted paper from a conference is an event; the business event source for this event is the role Author while the business event sink is the Review A Paper task.

Figure 4 shows a UML class diagram of our Business Event Model.

### 3.1. Business Event Type

A business event type is a declaration that provides the names and types of properties to be included in events that conform to this type. At runtime, events are instantiated as lists of named data values that may be broadcast using some notification mechanism for consumption by subscribers to this event type.

Business event types have a name and a domain (within which the name is interpreted). Business event type inherits from `UML::State Machine::Event`, which enables the specification of the names and types of the properties (as `UML::Parameters`) that conforming events will contain. This approach to event types is compatible with a number of well-known event transmission systems [6, 8, 9, 10].

Business event types are used in the context of an event emission in order to check that the expected properties of a generated event are present and have correctly typed values. In the context of an event consumption they simply provide the author of a notification rule with a set of names and types to use as terms in a subscription expression. It is expected that these types will be used by the mapping of the model to perform type checking, and that they will be stored in a repository for reference by the application.

### 3.2. Business Event Sources and Sinks

A business event source represents the external exposure of some enterprise action, while a business event sink represents the need for awareness of some enterprise action.

A business event source defines the business event type to be emitted, the conditions under which a business event of that type is emitted (the event exposure rule), how the values in the event are obtained from the source's state (the

event content mapping specification), and extent to which the event should be broadcast. Similarly, a business event sink defines the business event type to be received, the conditions under which such events should be consumed, how the event's values are assigned to the sink's state variables, and the extent from which events can be received.

Although events are primarily intended to support a decoupled communication paradigm, business event sources can be associated with specific business event sinks using the `transmit_to` association.

The extent to which the event is to be broadcast/received can be set to one of three built-in values:

- `global`, indicating that the event can be broadcast/received outside the scope of the enterprise being modelled

- `application`, indicating that this event can be broadcast/received by elements in this enterprise model

- `direct_only`, indicating that the event can only be broadcast to, or received from, explicitly-defined transmission paths (as defined in the `transmit_to` association).

User-defined extents can also be supported (typically implemented using event filtering).

### 3.3. Business Event Transceiver

A Business Event Transceiver represents the features of an event that are common to both the source and the sink of events. So, while the statements made above about sources and sinks is correct, in our meta-model we extract the common features from both sources and sinks into a common superclass called Business Event Transceiver as shown in figure 4.

### 3.4. Event Content Mapping Specification

The Event Content Mapping provides a way to describe how a Source can inject the payload into an event, or how a Sink can extract the payload from an event.

It supports the translation from values in the Source Model Element generating the event to the values represented in the Event Type, or alternatively how to map values from an Event Type to a Sink Model Element

In our model, the Event Content Mapping Specification is an aggregation of zero or more Event Content Mappings, where each Event Content Mapping represents the mapping for a single value.

Event Sources and Sinks contain Event Content Mapping Specifications through inheritance from the Business Event Transceiver properties as shown in figure 4.

## 4. The CORBA Notification Service

The CORBA Notification Service [8] is a publish/subscribe event notification service. Its primary concepts are suppliers, consumers, event notifications and channels. A supplier is a client of the service which publishes event notifications on a particular event channel. The channel propagates these notifications to all consumers which have subscriptions matching the notification's type and/or property values. An event channel can accept notifications from many suppliers, and deliver them to many consumers without these parties being aware of each others' identities.

The interfaces defined for suppliers and consumers are used as base interfaces for the channel. This approach allows the same operations to be used for a supplier to directly interact with a consumer as for a consumer to interact with a channel, and then for a channel to forward notifications to a consumer. The derived interfaces offered by the channel are called *proxy* interfaces.

There are two models for delivery of events from suppliers to channels, and from channels to consumers. These are known as push and pull models. In the push model the party sending the notification makes a call to the party recieving the notification, sending the notification as a parameter. In the pull model the party sending notifications gives a callback object reference to the party receiving notifications, and the receiver polls the sender by calling a pull method on that reference, receiving the notification as a return value.

The combination of consumer/supplier orientation and push/pull model gives four main kinds of proxy interfaces for channels to support:

- ProxyPushSupplier - for use by consumers using the push model

- ProxyPullSupplier - for use by consumers using the pull model

- ProxyPushConsumer - for use by suppliers using the push model

- ProxyPullConsumer - for use by suppliers using the pull model

Furthermore, there are three kinds of notifications defined by the Notification Service: typed, untyped and structured. The first is for defining variants of the service with domain-specific interfaces – we will not address this approach here. The second is for backward compatibility with the CORBA Event Service, and the third is the mechanism used by most Notification Service clients. The names of the interfaces that notification clients must support in order to interact with the service using structured events are of the form Structured[Push | Pull][Supplier | Consumer], and the
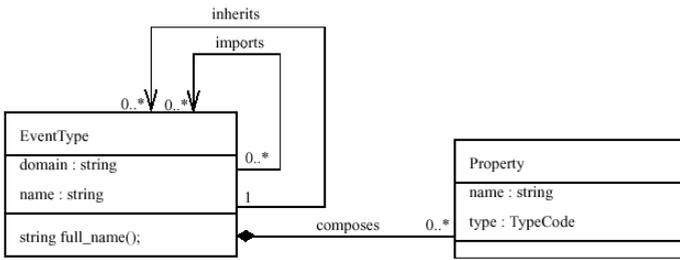
**Figure 6. Event Type Definition**

Proxies that they connect to on the notification channel are named: StructuredProxy[Push | Pull][Supplier | Consumer].

Here is an CORBA IDL example of such an interface definition:

```
interface StructuredPushConsumer :
        NotifyPublish {

  void
  push_structured_event(
    in StructuredEvent notification
  ) raises(Disconnected);

  void
  disconnect_structured_push_consumer();
};
```

When using structured notifications each notification has an event type, which has a domain, a name, and a set of property types consisting of a name and type. The Notification Service specification describes an Event Type Repository in terms of a Meta Object Facility [7] model. We show the part of the model, expressed in UML, which relates to event types in Figure 6.

The Notification Service defines a "front-door" interface called EventChannel, which has operations to return interfaces called SupplierAdmin and ConsumerAdmin. The Admin interfaces in turn allow the consumers and suppliers to call "obtain" operations which return proxy object references to them for connection to the channel and ultimately for pushing/pulling notifications.

Subscriptions to events may be made on the basis of an event type and/or a predicate over the values of a notification's properties. Subscriptions are contained inside Filter objects, which are attached to a proxy object. Filters may be attached to both proxy consumer objects, and proxy supplier objects. All Filters evaluate the notifications arriving at the proxy to which they are attached, and filter out notifications which do not match one or more of their subscriptions.

Filters attached to proxy consumers are typically used to filter out notifications which are not meant to be published by suppliers. This approach uses the evaluation engine in

the Notification Service to choose from a set of events sent to the channel, rather than programming a selection mechanism in the supplier itself. Filters attached to proxy suppliers by notification consumers evaluate notifications flowing through a channel against a set of consumer's subscriptions and deliver only the ones that match.

## 5. Mapping Business Events to the CORBA Notification Service

Models of business events in an enterprise application created using the business event model outlined above are at a high enough level of abstraction that several possible mappings to CORBA and CORBA services are possible. For example, the transmission of business events as asynchronous messages may be implemented using CORBA Messaging, the DII, CORBA Events or Notification services.

In this section we show only the mapping to the CORBA Notification Service.

### 5.1. Mapping Business Event Type

The Business event type described in section 3.1 is mapped to a Notification Service event type Repository entry.

For each business event type a Repository entry is created. The generalisation relationships between business event types can be preserved using the *inherits* relationship in the event type Repository. *inherits* is a single inheritance relationship, and so multiple inheritance of base business event types must be represented using one *inherits* relationship | to the parent Type considered to the the closest semantic match | with all other bases related through the *imports* relationship in order to ensure that the new EventType in the Repository still has the same set of inherited properties as its equivalent business event type.

The value of the business event type's domain_name attribute will become the value of the "domain" attribute of the new EventType, and the value of the business event type's type_name attribute will become the value of its "name" attribute.

All other attributes of the business event type are mapped to Properties contained by the new EventType. Each attribute name maps directly to a Property's name, as they are both strings, but the attribute's type may require some human choice to map the UML type to most convenient CORBA representation for the Property's "type" attribute.

### 5.2. Mapping Business Event Transceiver

The UML event model in figure 4 shows how our business event sources and sinks derive from a common abstract

base class: Business Event Transceiver. The Business Event Transceiver is mapped to an abstract CORBA IDL interface for use in the derived types.

```
module EDOC {
  struct PropMapping {
      from any;
      to any;
  };
  typedef sequence<PropMapping>
    PropMappings;

  interface BusinessEventTransceiver {
    attribute active boolean;
    readonly attribute rule string;
    readonly attribute
      type NotificationTypes::EventType;
    readonly attribute
      mapping PropMappings
    readonly attribute extent string;
  };
};
```

## 5.3. Mapping Business Event Source

Business Event Sources are objects that live in the same address space as the objects implementing the model elements to which the source is attached. The implementation of these model elements to which an event source is attached must expose their actions of interest (see 2.3) by calling appropriate methods on the object implementing the source.

### 5.3.1  Interface

Using the abstract BusinessEventTransceiver interface as a base class, the Business Event Source is mapped to an implementation of the following interface:

```
module EDOC {
 interface BusinessEventSource :
   BusinessEventTransceiver,
   CosNotifyComm::StructuredPushSupplier{
 void action (
   in string action_name,
   in any context);
 void transition(
     in string transition_name,
     in string from_state,
     in string to_state,
     in any context);
 };
};
```

The `action()` operation is invoked by the model element associated with this source whenever an action of interest occurs. When state transitions in a state machine representing the model element's behaviour are nominated as actions of interest the `transition()` operation must be invoked.

The StructuredPushSupplier base interface allows the object implementing the BusinessEventSource interface to connect to a Notification Service channel to supply notifications to it.

### 5.3.2  Implementation

The implementation of the object supporting the BusinessEventSource interface embodies the evaluation of the Business Rule attribute of the Business Event Source. The object must contain appropriate pointers and object references to the names in the rule representing values of the source Element. Whenever the `action()` or `transition()` operations are called the rule must be evaluated, and if the result is true then a StructuredEvent of the type mapped by the `type` attribute of the Business Event Source is created.

Appropriate local variable pointers and object references to the Model Elements of the mapping's contained `model_attrs` must also be available to the object. These variables must be used to obtain current values in the objects mapped by the `model_attrs`, and then copy them into the corresponding StucturedEvent property's `value` member.

Finally, the event should be pushed to the Notification Service. (See the Configuration sub-section 5.3.3 for details of connection to the Notification Service.) The push model is chosen so that the event is available to the Notification service, and hence to Business Event Sinks connected to it, at the earliest possible time. The pull model always involves a polling delay.

The evaluation of the Source's rule may be delegated to a Filter on the proxy consumer in the Notification Service to which it is connected. This requires that the Boolean-Expression language chosen in the model can be translated to the Notification Service Constraint Language, and that the attributes that the rule evaluates are present as mapped properties in the generated event. In this case, the invocation of the `action()` or `transition()` operations will always generate an event when the source is active, and the rule will evaluate as a filter which blocks the transmission of the event at the proxy if the rule is not satisfied. This trade-off means that more events may be generated by the source model element, but that less processing is done by the source as rule evaluation is delegated to the filter.

### 5.3.3  Configuration

The `extent` attribute of the Business Event Source is used to determine which Notification Channel is to be used to

broadcast events that are generated. The standard values for `extent` are mapped as follows:

- "global" – The application as a whole will need a bootstrap parameter containing an object reference for an event channel connecting this application to others that wish to see events that it publishes globally. Alternatively the default channel for such events should be that obtained by calling `resolve_initial_references()` on the ORB with "NotificationService" as its parameter. Then the default channel (number 0) should be chosen, and the default SupplierAdmin used to obtain a Proxy.

- "application" – All EDOC applications using Business Events should have some bootstrap code that creates a new channel for internal events. This channel's reference should be stored in a standard place in the Naming Service for access by all Business Event Sources and Sinks with application extent.

- "direct_only" – A new channel is created for each Business Event Source, and this channel's reference is obtained only by the Business Event Sinks that are associated with it by the `transmit_to` association in the model.

## 5.4. Mapping Business Event Sink

Business Event Sink is mapped to an implementation of the following interface:

```
module EDOC {
interface BusinessEventSink :
    BusinessEventTransceiver,
    CosNotifyComm::StructuredPushConsumer{};
};
```

Similarly to the Business Event Source implementation, the BusinessEventSink must have access to the objects implementing the `sink` Model Element.

The implementation may embed the Event Exposure Rule, given by the `rule` attribute in the model. In this case appropriate pointers and object references to the names in the rule representing values of the sink Element must be available. The rule is evaluated when the Notification Service calls the `push_structured_event()` method on the sink. When it evaluates to true, the `mapping` is applied. The code also must provide appropriate local variable pointers and object references to the Model Elements of the mapping's contained `model_attrs`.

The evaluation of the Sink's Rule may be delegated to a constraint expression string in the ConstraintExp used at the Filter on the Proxy to select events of the right type. This can be done if the BooleanExpression language chosen in the model can be translated to the Notification Service Constraint Language, and the variables that the Rule evaluates are all properties in the received event.

### 5.4.1 Configuration

The `extent` attribute of the Business Event Sink is mapped similarly to the mapping described for the Business Event Source.

The BusinessEventSink should use the ConsumerAdmin interface to obtain a ProxyStructuredPushSupplier from the channel, and pass its own object reference to the proxy's `connect_structured_push_conumer()` method. The Sink must then create a Filter and add to it a ConstraintExp that subscribes to events of the EventType it requires.

Whenever the active attribute is toggled from true to false, the Filter's subscription to the event type must be removed from the Filter using the `modify_constraints()` operation. When toggled from false to true, it should be re-added using either `add_contraints()`, or `modify_constraints()`.

## 6. Example

This section introduces by way of an example a fragment of an Enterprise specification which attaches business events to elements of business processes so as to model an asynchronous dataflow between business processes. It then shows how this model could be realised in code using the Notification service as target as has been described in section 5.

### 6.1. Overview

The fragment of the enterprise model used in this example defines business processes for handling the receipt of papers submitted to the conference, choosing the Business or Technical stream a given paper is most relevant to, and having papers reviewed by peers with relevant experience. As part of the review process, a reviewer can re-classify a paper from the currently selected stream to the other and cause the paper to be re-allocated to another more appropriate reviewer.

### 6.2. Detailed Description of Business Processes

Figure 7 shows the details of the `Receive/Process Paper` Business Process as a compound task. On executing the `Receive/Process Paper` task, the `Receive/Pre-sort Paper` sub-task can only commence execution when the Business Event Sink `recv_paper` receives an event of type `paper_event`.
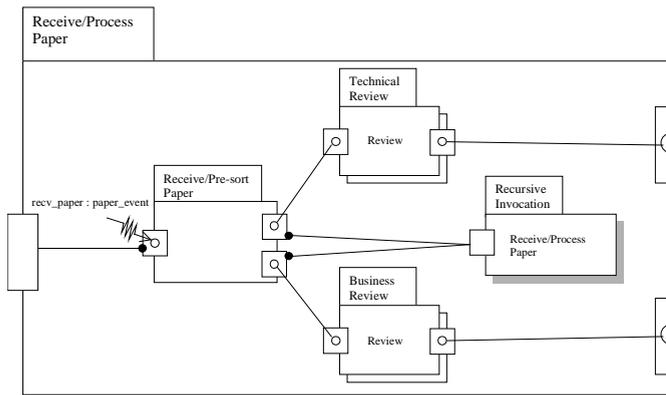
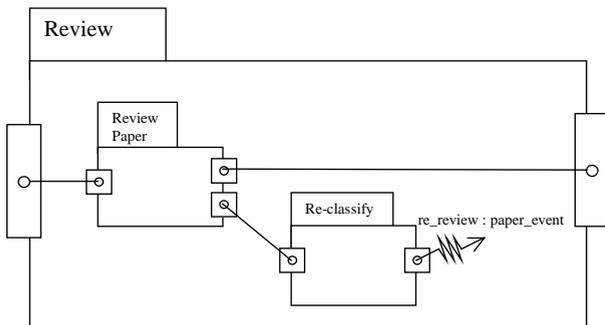**Figure 7. Receive/Process Paper Task**



**Figure 8. Review Task**

That is, the task will wait until it receives an asynchronous event from an external source. In a system implementing this business process, such a notification could come from, for example, a web-based paper submission system.

Completion of the `Receive/Pre-sort Paper` task enables either the `Technical Review` or a `Business Review` (but not both) to start. These tasks invoke the `Review` task. Completion of the `Receive/Pre-sort Paper` task also enables the `Recursive Invocation` task which causes a new invocation of the `Receive/Process Paper` task to be created which will itself wait for notification of papers to process and review.

Figure 8 shows the definition of the `Review` task. This task attempts to review the paper nominated by the data input. It either produces a review of the paper, or causes the paper to be re-classified and emits a `re-review` event of type `paper_event` to signal that the re-classified paper is ready to be processed and reviewed as is appropriate.

Both Figures 7 and 8 show data inputs and outputs to many of the tasks (show as the hollow circles in the input and output sets of the tasks). In our notation, each of these data inputs and outputs should be labeled with their name and type. However for clarity in this paper, these names and types have been omitted. In this example, all the data items
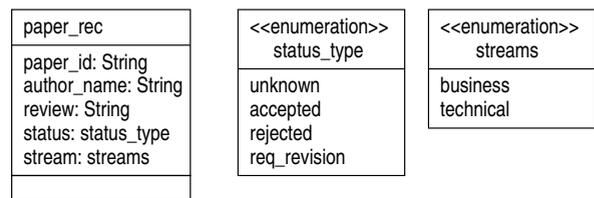


**Figure 9. UML definition of** `paper_rec`

are of type: `paper_rec` described by the UML diagram in Figure 9:

In Figure 9, `paper_id` is a unique id for the paper and can be used as a key for referencing the actual electronic version of the paper stored in the conference management system, `author_name` is the name of the primary author, `review` is the review of the paper, `status` tracks the status of the paper in the system, and `stream` denotes which stream of the conference this paper has been assigned to.

### 6.3. Event Mapping

In this section we detail how the business events produced and consumed in figures 8 and 7 can be mapped to the OMG Notification service.

#### 6.3.1 Business Event Type

The business event type `paper_event` has an entry in the Notification Service Event Type Repository of the form:

```
name: paper_event
domain: application

paper_id: String
author:   String
review:   String
status:   enum{unknown, accepted,
               rejected, req_revision}
stream:   enum{business, technical}
```

The domain for this event is defined here as 'application' meaning that the event has no meaning outside the scope of this application and will not be visible to applications external to this system.

#### 6.3.2 Business Event Source

The Business Event Source `paper re-classified: paper_event` shown in figure 8 is mapped to an object implementing the `BusinessEventSource` interface having the following properties:

```
active: TRUE;
rule: task.state=completed;
```

```
type: paper_event;
mapping:  source_mapping;
extent: application;

source_mapping:
  (paper_id, paper_id),
  (author_name, author),
  (review, review),
  ("req_revision", status)
  (stream, stream)
```

The `source_mapping` lists mappings from source object's representation of the data to the corresponding fields in the event type (shown here as ordered pairs). In this example, mostly these items have the same name, however `author_name` is mapped to the author field, and the string `"req_revision"` is mapped into the status field.

The `rule` is a guard ensuring that this event can never be emitted unless the task is in the 'completed' state - that is, the event can only be emitted when the `Re-classify` task has finished.

For this object, the `action` method will have an empty implementation and calls on this method should raise an exception. Instead, because the event will be emitted when state machine for the `Re-classify` task transitions to the `finished` state and the output set of the task is enabled, `transition` method will be called. This method must have an implementation that will push the event onto the notification channel by calling the `push_structured_event` method with appropriate parameters. The transition method will be called with the following parameters:

```
void transition (
   "finish", "running", "com-
pleted", context
  );
```

`finished`, `running` and `completed` are states in the state machine for the task as described in section 2.1.

The `context` parameter will be null in this instance as there is no additional context information that is required for this transition to cause an event to be emitted.

#### 6.3.3  Business Event Sink

The Business Event Sink `paper_to_review:  paper_event` shown in figure 7 is realised by an object implementing the `BuinessEventSink` interface with the following properties:

```
active: TRUE;
rule: task.state="Not Started";
type: paper_event;
mapping: sink_mapping ;
```

```
extent: application;

sink_mapping:
  (paper_id, paper_id),
  (author, author_name)
  (review, review),
  (status, status)
  (stream, stream)
```

## 7. Conclusion

The ability to define, publish and subscribe to events relating to items of interest in a business is significant element of any framework for enterprise level descriptions of systems [4][10]. However, it is not sufficient to simply describe such aspects of a system. It is important that there be a clear mapping from such enterprise models to distributed object technologies that will be used in the implementation of systems.

In this paper we have introduced some aspects of the DSTC initial submission to the OMG's RFP for a UML Profile for Enterprise Distributed Object Computing. While describing our process and role models, this paper has focused on the details of our business event model and how this model is related to elements of the business process and role models.

We have described the CORBA Notification Service as an introduction to showing how our business event model can be mapped to implementations using the Notification Service and we illustrate how this can be done by working through a practical example.

While we have demonstrated that our high-level model can be mapped to a low-level implementation, our future work in this area will be towards automating this mapping as much as is possible. Our goal is to allow analysts to produce high-level enterprise models and to facilitate as much as is possible the generation of code to support implementations of these models. We recognise that manual coding of the business logic will always be required, however we aim to generate skeleton code for the infrastructural middleware where possible. In this paper we described mappings to CORBA and CORBA Services since this is where our current expertise lies, however we recognise that other middleware technologies exist and are being developed and we fully intend to investigate similar mappings from our models to these technologies so that our modelling and mapping approach can be as widely applicable as possible.

## 8. Acknowledgements

Language within ISO [1]. This standardisation work provided the basis for our Business Entity Model. Our Business Process Model has been substantially based on results from a workflow project [13] carried out at the Department of Computer Science, University of Newcastle upon Tyne, UK, and sponsored in part by Nortel Corporation.

## 9. References

## References

[1] ISO, Open Distributed Processing - Enterprise Language, ISO/IEC JTC1/SC7/SC17 N0080, July 1999.

[2] Object Management Group, "Unified Modelling Language v1.3", OMG ad/99-06-08, June 1999.

[3] Object Management Group, "Request for Proposal: UML Profile for Enterprise Distributed Object Computing" OMG ad/99-03-10, March 1999.

[4] DSTC, "UML Profile for Enterprise Distributed Object Computing", OMG ad/99-10-07, October 1999.

[5] Object Management Group, Workflow Management Facility, OMG bom/98-06-07, July 1998.

[6] Object Management Group, Event Management, OMG formal/97-12-23, 1997.

[7] Object Management Group, Meta-Object Facility, OMG ad/97-08-14, September 1997.

[8] Object Management Group, Notification Service, OMG telecom/99-07-01, July 1999.

[9] B. Segall, D. Arnold, "Elvin has left the building: A publish/subscribe notification service with quenching" Proc. Australian Unix Users Group, Brisbane, Australia, September 1997.

[10] K. Reimer, "A Process-Driven Event-Based Business Object Model", Proc. 2nd International Enterprise Distributed Object Computing Workshop, pp 68-74, Novemeber 1999,

[11] A. Barros, K. Duddy, M. Lawley, Z. Milosevic, K. Raymond, A. Wood, "Processes, Roles, and Events: UML Concepts for an Enterprise Architecture", the Third International Conference on the UML, UML 2000 Conference (UML2000 UK), October, 2000.

[12] A. Barros, K. Duddy, M. Lawley, Z. Milosevic, K. Raymond, A. Woody, "MApping Enterprise Roles to CORBA Objects using Trader", to be published in 3rd IFIP/GI International Conference on Trends towards a Universal Service Market, USM 2000 Conference (USM2000 Germany), September, 2000.

[13] J.J. Halliday, S.K. Shrivastava, S.M. Wheater, Implementing Support for Work Activity Coordination within a Distributed Workflow System, Proc. 3rd International Enterprise Distributed Object Computing Conference, Sept 1999, pp 116-123.

[14] T. Reenskaug, P. Wold, and O. A. Lehne, Working with Objects - The OOram Software Engineering Method, Manning Publications, ISBN 1-884777-10-4, 1996

[15] B. Selic, J. Rumbaugh, Using UML for Modeling Complex Real-Time Systems, ObjecTime and Rational White Paper, http://www.objectime.com/otl/umlrt.html