Intelligent OOA

Intelligent OOA Customer Technical Note

Polymorphic Events

Ref.	:	KC/OOA/CTN 09
Authors	:	Ian Wilkie
Date	:	24th January 1994
Version	:	1.3

Contents

1. Introduction

Intelligent OOA exhibits some rather complex behaviour in the area of polymorphic events, and we have found that this has caused some confusion amongst analysts. This note aims to make the policy behind the behaviour clear.

2. Polymorphic Events in OOA

In Shlaer Mellor OOA, analysts may use super/subtype hierarchies for the polymorphic transmission of events. This means that a state machine may transmit an event (with an appropriate instance identifier) apparently to a supertype object instance in the knowledge that the event will be received and acted upon by the related subtype instance. This feature provides a degree of behavioural encapsulation in that the transmitting object does not need to worry about the detail of which type (in the subtype sense) the destination object is.

For example, in the Optical Disk Management System (ODMS) case study, the Robot simply tells the supertype (Online Location): "Robot done at source". Each of the various subtypes (Slot, Drive and EE Port) respond in an appropriate way, but as far as the Robot is concerned it has simply taken a disk away from a location in the cabinet.

This is a powerful feature, but some aspects need to be made clear:

- Since in general, we do not know until run time which subtype will receive any given polymorphic event, as analysts we must specify the response for *all* the subtypes. It is not acceptable to leave any response as undefined.
- Supertypes can of course have multiple subtype hierarchies attached. Polymorphic events directed at a supertype must therefore be available to *all* the subtypes in all the hierarchies. The responses to these events must be defined. Note however, that in many of these subtypes, the response could be to ignore the event completely.
- Subtypes can themselves be supertypes of other subtypes. All events available to such intermediate types (including those that are available because of polymorphism) must become available to their subtypes. In principle there is not limit to the depth that such hierarchies can reach, and all responses must be defined.
- In such multiple level hierarchies there may of course be many objects that do not have state machines. For example in a simple hierarchy, the supertype could have no state model while the subtypes each have behaviour appropriate to their type. In those objects that have no state model, the availability of polymorphic events can be considered to have no effect.
- In a hierarchy where there is dynamic behaviour at more than one level (for example at both super and subtype level) there is in principle no reason why more than one state

machine should not receive and respond to the same event instance. For example both the supertype and its corresponding subtype might respond to such an event.

- No extra assumptions can be made about the order of reception or processing of polymorphic events at run time. In particular, one cannot assume that the supertype will process an event before the subtype.

3. *Intelligent* OOA Support for Polymorphic Events

Support for polymorphic events is provided through the following features:

- Events directed at a supertype object are automatically available to all of the subtypes in all of the super/subtype hierarchies descending from the object. The events are available at every level in the hierarchy. "Available" means that the events appear as columns in the State Transition Table for the object, and appear in the Event Selection pick list when attaching events to a transition on a State Transition Diagram.
- To make things clear, the tool distinguishes between those events that are received by an object directly, and those that are polymorphically available:
    - "Directed" events are those that have been explicitly directed to the object in question, and have the same Key Letter as the object.
    - "Available" events are those that will be transmitted to the object by virtue of polymorphic transmission. Such events will have the Key Letter of the (supertype) object that they were *Directed* at.

  Text frames and reports will distinguish between these two classes of events.
- When the analyst attempts to define a transmission of an event to a subtype object, he is given the choice of directing that event at the subtype or directing the event to a supertype of the object, and thus making the event polymorphically available to the subtype.
- If an object in a hierarchy does not have a state model, then any polymorphically available events have no effect on the object and are considered to be ignored. Further, in a state model at any particular level in the hierarchy a polymorphically available event may be explicitly ignored (by setting all the effects to ignore). However, when the model is checked the analyst will be warned if any given event is ignored at *every* level in some branch of the hierarchy.

4. The Object Communication Model (OCM)

The purpose of the OCM is to show the pattern of event transmission in a domain. If an object can transmit an event which can be received by another object then this will be shown as an arrow between the two objects on the OCM.

In *Intelligent* OOA, there are two ways that the transmission of an event can be defined by the analyst. First, the analyst may explicitly draw a transmission between two objects on the OCM, secondly the analyst may define that a particular event is generated in a state of an object. In the second case the tool will automatically create a transmission on the OCM between the object "owning" that state and the object that the event is directed at.

In the case of polymorphic events *Intelligent* OOA will automatically create a transmission, not only to the object that the event is *directed* at, but also to all those object that the event becomes polymorphically *available* to.

This has a rather striking consequence:

- Adding a transmission to a supertype will cause transmission to be created from the source object to *all* of the subtypes in all of the super/subtype hierarchies descending from the supertype object. This is an inevitable consequence of the notion of polymorphic events providing behavioural encapsulation. The transmitting object does not "know" which subtype will actually receive the event at run time. In fact the transmitting state model should not even care that the behaviour is modelled separately in the subtype objects. The OCM must therefore show the possibility of the event going to any of the subtype objects.

We have noted that some analysts prefer to think of polymorphic event transmission in terms of the event going to the supertype and then being *passed on* to the appropriate subtype. This can be a useful way to think about what is going on, and indeed, some architectures may be constructed this way (although the consequences of order of execution rules if the supertype has a state machine would need to be carefully thought through). However, we would maintain that whatever the mechanism by which the event becomes available to the subtype, the fact is that an object transmits the event and it is received by a subtype. The OCM should and does reflect this.

In practice of course there may be events that go to a supertype state model that are not used by the subtype state models or vise-versa. Under these circumstances the OCM will show many event transmissions that are simply not relevant. To combat this problem and to avoid very cluttered OCMs, if an event is completely ignored by a state model, *Intelligent* OOA will render the transmission to that object invisible. The transmission still exists in the database (together with all it diagrammatic positional information) and will re-appear if any response become unignored. To ignore an event completely, the analyst can either set all the effect of the event to “ignore”, or he can select the event name on the OCM transmission and select “Ignore Event” .

5. Event Types and Event Instances

We have noticed that where analysts become particularly surprised by *Intelligent* OOA’s insistence in inserting polymorphic transmissions is in the following case:

- A subtype sends an event to it’s *own* supertype: The logic of the transmitting action allows no other possibility. The event is then polymorphically transmitted to the instance of the subtype that sent it.

In this case, it seems odd that a transmission is shown from the subtype in question to all the other subtypes, since we know that this transmission will *never* occur at run time. We feel however that this is not a genuine and legitimate use of a polymorphic event since :

- The transmitting object *does* know, and care what subtype the receiving object is, and therefore there is no element of behavioural encapsulation.
- A simple notion of sending an event from an instance to itself has been made very complicated, and the OOA model will be obscured as a result.

The only advantages of using the polymorphic mechanisms in this way are:

- Both the supertype and the subtype are informed of an occurrence by one single event.
- A single event *name* is available for use by all the subtypes.

The first of these means the avoidance of one event and one event generation - hardly a major task for the analyst. The second advantage is more interesting. Imagine the situation where all the subtypes of “request” have different behaviour for the most part, but that some aspects are very similar. In particular every time a “request” ends it sends a “request completed” event to itself. One way of drawing attention to the fact that some behaviour is common between the subtypes is to ensure that the event has the same name in all the subtypes. By directing the event at the supertype, the analyst need only specify the event name once, and indeed need only change it in one place.

Clearly there is a need here to support some level of commonality between state machines. In this case we would have the notion of an *Event Type* (which has a name), where each subtype state model uses an instance of that event type. This is in fact a part of the more general issue of *Spliced* stapte machines, where not only events but also state and action specifications are shared between subtype state models.

Note that spliced state machines are not the same thing as having dynamic behaviour in both the super and subtypes. In the former case, there is only a single execution thread for a given super/subtype instance pair, whereas in the latter case there are parallel, independent threads of execution for the subtype instance *and* for the corresponding supertype instance. Indeed one could imagine a particular hierarchy having behaviour specified by a state model in the supertype *and* by spliced state models in the subtypes.

Intelligent OOA does not at present support the idea of spliced state machines.

6. Summary

- Polymorphic events are a means whereby a degree of behavioural encapsulation can be expressed in OOA.
- *Intelligent* OOA provides full and flexible support for polymorphic events.
- The polymorphic event mechanism can be abused to provide a degree of support for spliced state machines. It is possible that in the future, *Intelligent* OOA might detect and warn of such use.