Polymorphic Events in OOA/RD

Ref: CTN 57 v1.2

www.kc.com

Table of Contents

1. Introduction

The Executable UML formalism (xUML) supported by the iUML Toolset is a software development technique that has its roots in the OOA/RD method originally developed by Sally Shlaer and Steve Mellor in the late 1980s and early 1990s. In that form it was used successfully on a large number of software projects in various industry sectors.

In the late 1990s,with the advent of the Unified Modelling Language (UML), the ideas of OOA/RD were reused and extended to create the “Executable UML” technique.

This document (first published in 1999) clarified the model structure and execution semantics of polymorphic events in OOA/RD. These ideas have been fully carried forward into the Executable UML formalism.

We hope that such readers will find the document useful for understanding the background and development of the xUML formalism as well as providing, an in-depth justification for, and explanation of, the features of the technique.

Introduction to the 1999 edition

Polymorphic Events have been part of the OOA/RD method for many years. Kennedy Carter provided full support for such events in the first releases of the *Intelligent* OOA CASE tool. This support was consistent with the published examples of such events from Shlaer and Mellor.

Unfortunately, although the model construction aspects of the polymorphic events embedded in I-OOA were clear, the run time behaviour was insufficiently specified. This has lead to some variation in architectural behaviour. In addition, in the “OOA 96 Report” Shlaer and Mellor formally introduced the idea of polymorphic events. This was presented in a different way from the I-OOA approach.

This paper sets out the Kennedy Carter definition of both the model construction and run time behaviour of polymorphic events. Where appropriate this has been contrasted with the Project Technology approach.

Acknowledgements

This paper summarises the results of extensive thought by Kennedy Carter consultants and tool developers over the years. Their contribution is acknowledged. In addition we have benefited greatly from discussions with Steve Mellor.

2. Terminology

This paper uses a number of words and phrases to describe various aspects of polymorphic behaviour. We define their meaning here:

- "Super/Subtype Hierarchy" means tree descending from a supertype object down a sequence of super/subtype relationships. Where a given supertype has more than one super/subtype relationship descending from it, a particular hierarchy refers only to one of the relationships.

- "Subtype family" means a single super/subtype relationship and the subtype objects attached to it. A hierarchy is thus a sequence of one or more subtype families.

- "Down the hierarchy" means moving from the supertype to a subtype in a particular super/subtype relationship. (For example, a polymorphic event is propagated down a hierarchy).

- "Up the hierarchy" means moving from the subtype to the supertype in a particular super/subtype relationship.

3. Method Baselines

Since the publication of the first book on OOA by Shlaer and Mellor various clarifications and extensions have been issued. The intention of this section is to summarise these so that the discussion of polymorphic events may be seen in context. It is not intended that this in any way replaces the published definitions of these extensions and in case of doubt the original publications should be consulted.

A complication in defining clear versions of the method is that there have often been a continuous series of minor improvements that have been incorporated into, for example, training material or consulting work on real projects without there having been an "official" and published release. Nevertheless, it is useful and convenient to attempt to define such versions.

3.1 OOA 88

This method, described in [M2] is confined mainly to Information Modelling with a minimal treatment of dynamic modelling and “design by translation”.

3.2 OOA 91

The method as published in “Object Lifecycles: Modelling the World in States” [M3], represents the first virtually complete description of OOA as we might recognise today. It differs from OOA 88 in the following respects:

- Minor Improvements to the Information Modelling formalism and notation such as Numbered Relationships.
- Substantial definition of dynamic behaviour in terms of interacting state machines executing models represented by State Transition Diagrams and State Transition Tables. Included with this was a treatment of the rules of synchronisation and concurrency within an OOA model.
- Introduction of the idea of domain partitioning with an outline treatment of bridges.
- A discussion of the Software Architecture that emphasised the idea of translation as system construction approach.

3.3 OOA 92

While developing CASE tool support that *understood* the formalism Kennedy Carter extended the definition and notation some areas. Most of these issues were documented in the *Intelligent* OOA User Manual, although some technical notes were written.

The following issues were addressed:

Information Models

The following changes were made:

- The case of a general n-way relationship that had briefly been mentioned in OOA 88 but omitted from OOA 91 was withdrawn completely from our support. Our experience was that such relationships are very hard to understand and the real-world issues that they represent are better described by a number of simpler relationships.

- The notion of an "attribute domain", which was informally described in OOA 91 was tightened to include the idea of a data type with optional constraint.

State Models

In order to make the STT representation a complete description of the state model the following were added:

- A row representing the state of an instance before its creation. This pseudo-state enables the STT to show creation transitions. In addition, it allows the analyst to distinguish between the arrival of an event targeted at a non-existent instance being an error ("Cannot Happen") and being expected ("Ignore").

- The addition of the effect "Meaningless" for events arriving for instances in a state where the instance is deleted.

Other changes were:

- The notion of a polymorphic event[1] was formally introduced. This was done in a way that was, we believed, consistent with the polymorphic events (OL1 and OL2) in ODMS case study from Project Technology. In OOA 92, events are *directed* at the object with the key letter of the event. Instance events are sent in ASL using a handle on the object to which they are directed. Events which are directed at a supertype object are automatically *available* to all subtype state models. The analyst must then specify whether they are used (i.e. cause some transition) or are simply ignored. The Object Communication Model was enhanced to reflect the polymorphic transmission of events.

- The idea of “synchronous services” were introduced. Such services specify processing that is executed synchronously with respect to the invocation and can return data to the invoking state action. OOA 97 refined the ideas and introduces the formal association of such services with objects and with object instances.

[1] See “Polymorphic Events” [M18].

- Self directed events go to the head of the event queue for an instance. (Previously this was a recommendation only).

Process Models

- The Action Specification Language (ASL) [M14] was introduced as an alternative to Action Data Flow Diagrams (ADFDs) for specifying process models.

Domains and Bridges

- Introduction of an "OOA of Bridges" describing all the possible bridges mappings that can exist between OOA domains [M6].

- Interaction with other domains captured through events being sent to[2]/from terminators. Guidelines were provided for the level of abstraction for the terminator (i.e. that the terminator represents the "ultimate source or sink" of a structured analysis "essential model" rather than the domain that implements it).

3.4 OOA 96

The "OOA 96 Report" [M17] tidied up a number of a number of loose ends in the description of OOA 91 and introduced some additional concepts. Some of these concepts had previously been described in Project Technology training material, but not incorporated in an "official" description of the method.

In outline the areas addressed were:

Information Models

- Clarification of the ideas of mathematical and stochastic dependence of attributes. Introduction of the notation (M) for mathematically dependent attributes replacing the (D) notation used previously.

- Clarification of the idea of relationship loops and composed relationships.

- Clarification to the ideas of reflexive relationships and the introduction of a new special case of symmetric reflexive relationships.

State Models

[2] In OOA 97 we replaced the idea of an event being sent to a terminator with a "terminator service invocation" akin to a wormhole.

- Notational distinction between identifying event parameters and supplemental data.
- Events can no longer be sent to terminators.
- Self directed events go to head of the event queue for an instance.
- Formal introduction of polymorphic events through the idea of a "Polymorphic Event Table"
- Clarification to the definition of the operation of the Finite State Machine mechanism.
- Occurrence of "Cannot Happen" at run time defined as an analyst error.
- Introduction of the new concept of "multiple assigners"
- Clarifications to the rules surrounding object instance creation and deletion

Process Models

- Process models are not longer allowed to access the "Current State" attribute of an active object (except in the special case of synchronous creation).
- Introduction of the "proper attribution" rule for transient data on ADFDs[3]
- Introduction into ADFDs of the ideas of "base processes" working on (possibly ordered) sets of data, thus formally supporting the idea of iteration.
- Changes to the allowable properties of different process types on ADFDs.
- Replacement of the "Timer" mechanism with a simpler "Delayed Event" mechanism.
- Introduction of the term "wormhole" to refer to the invocation of services provided by other domains.

[3] In the discussions that KC & Project Technology had in December 1995, and subsequently on the eSMUG, Sally Shlaer indicated that the important issue is that all the data should have a defined *type*. This modified rule is consistent with the approach taken by ASL.

3.5 OOA 97

OOA 97 was an accumulation of a number of issues that built up in the course of our consultancy work. Initially it was published as a series of proposal documents [M7-12] soliciting a number of detailed comments from our clients. These issues were then gathered into a single OOA 97 document [M13].

- Refinement of the idea of synchronous services by formal association of services with domains, objects and instances of objects. In ASL (as of Level 2.5) this association was captured with a defined syntax for the service calls.

- Introduction of the additional FSM responses of "Hold" and "Shouldn't Happen". The first of these was to deal with a specific class of problem complexity[4], the second in response to the OOA 96 rules that classify "Cannot Happen" as an analysis error if it does happen.

- Introduction of exception handling mechanisms within the OOA formalism.

- Introduction of support from the formalism for both "Deferred" and "Dynamic" data types.

- Introduction of a comprehensive support for the definition of Bridges within OOA/ASL including the idea of "counterpart relationships".

In order to support these ideas, the definition of ASL was upgraded from ASL 2.4 to ASL 2.5 [M14].

3.6 OOA 96++

Since the OOA 96 report, PT issued further method documents [M15, 16, 20] on the subjects of Data Types, Bridges & Wormholes and Synchronous Services. For convenience we refer to these as OOA 96++.

- Introduction of formal notion of a Data Type with a base type, range and precision and units

- Introduction of idea of wormholes with client-server associations managed through transfer vectors

- Introduction of idea of a synchronous service provided by a domain.

[4] As will all aspects of any method, this feature can be horribly misused. Our judgement was however that, on balance, the benefits outweighed the drawbacks.

4. Instances in a Super/Subtype Hierarchy

Before we can discuss the behaviour of polymorphic events it is important to clarify the OOA/RD approach to thinking about objects and instances in super/subtype hierarchies.

At run time of an OOA model, instances of objects are created and deleted and instances of relationships between the object instances are similarly created and deleted. In addition, for those objects that are "active" (i.e. have a state model), there is a state "machine" per instance of the object.

In the case of a super/subtype pair, consider the example hierarchy shown in Figure 1:

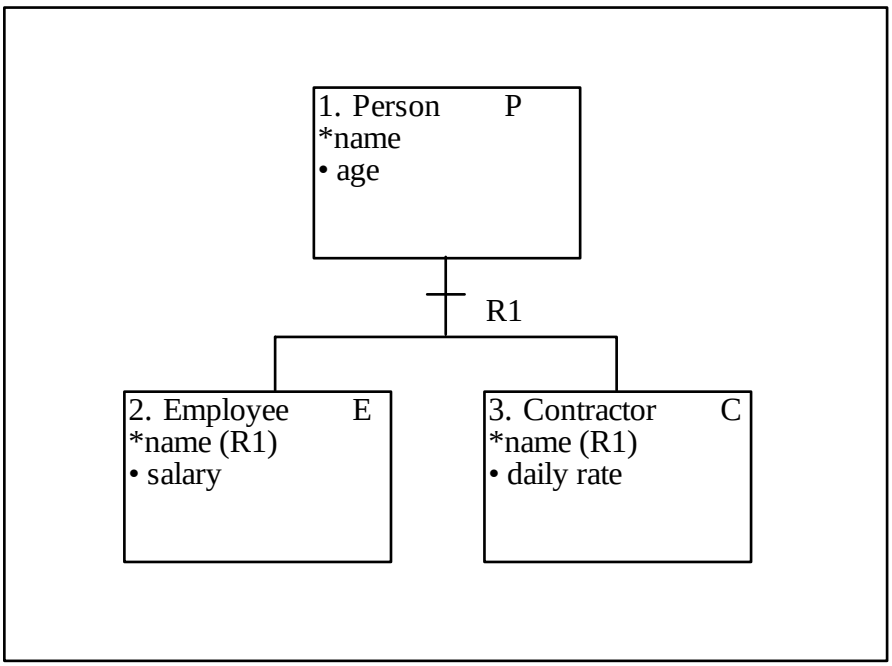

Figure 1 - A Super/Subtype Hierarchy

For a particular real-world employee for example, "Joe Bloggs" (of which there is only one instance), the OOA model captures **two** instances. There is a "Joe Bloggs" Person and a "Joe Bloggs" Employee. This is evident, for example, in ASL by the fact that an action segment must create instances of both these and explicitly link them:

```
new_person = create Person with name = "Joe Bloggs"
new_employee = create Employee with name = "Joe Bloggs"
link new_person R1 new_employee
```

At first sight this may seem a little cumbersome in that it would be simpler just to consider there to be one OOA instance, corresponding to the one real world instance and that the ASL would only have to show the creation of an instance of "Employee".

However consider now the following example:

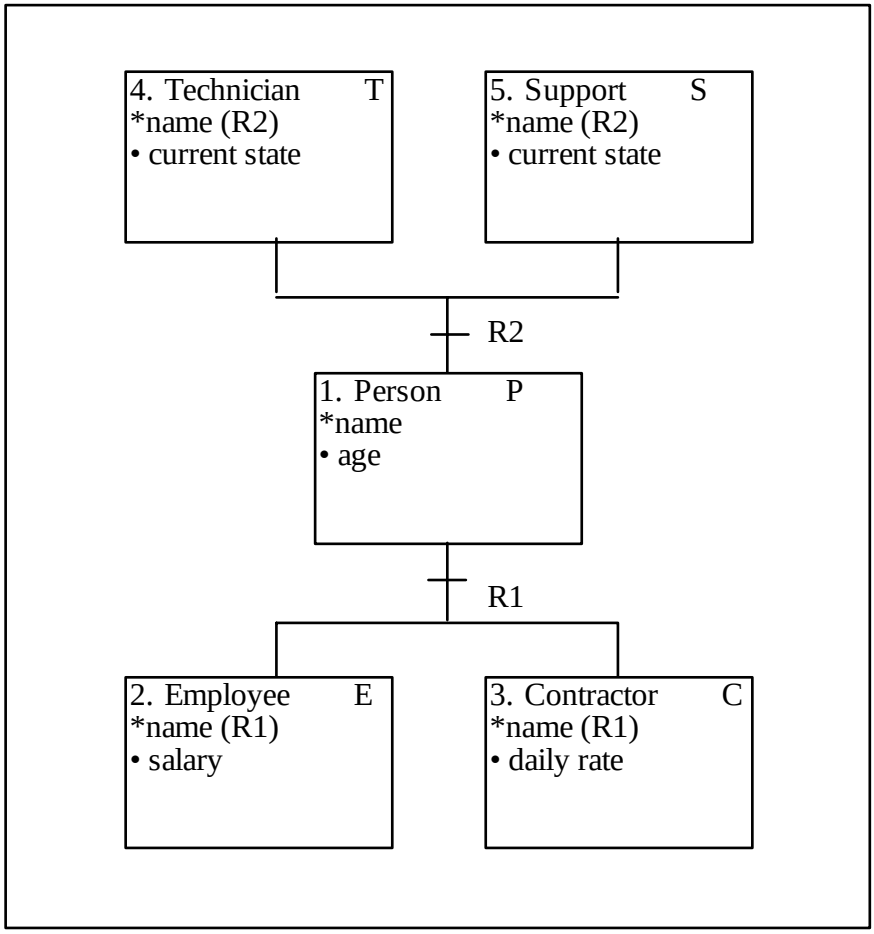

Figure 2 - Two Subtype Families

In this case, if we consider there to be only one OOA instance corresponding to the real-world instance, what happens when a new “Technician Employee” arrives ?

Do we create an Employee or a Technician or a Technician Employee ? What happens if a Support Employee changes jobs and becomes a Technician ? Does the Support Employee get deleted and a new Person get created ? If we were to model the problem that way this would not be a reasonable reflection of the real world. One person did not die and another get created. Rather, a single real world entity changed its role with respect to the problem domain.

Instead, taking the approach that each real world entity is represented by individual instances of all the appropriate objects in the hierarchy makes it much more straight forward to model this kind of behaviour. For example, if a Support Employee becomes a technician, we need only perform an action such as:

```
new_tech = create Technician with name = this.name
my_person = this -> R2
unlink this R2 my_person
link my_person R2 new_tech
delete this
```


Where “this” is the instance of “Support” that became a “Technician”.

This is the interpretation taken in OOA. Note that this approach regards each object in the hierarchy as being a regular OOA object that just happens to be connected to another via a relationship with particular semantics. This means that, potentially, each object (the supertype and the subtype) may have a state model.

5. Specification of Polymorphic Events

5.1 The Basic Requirement

The notion of a polymorphic event was formally introduced in OOA 92 (although at least one example of such an event had been previously published in training courses).

The concept of a polymorphic event is essentially a simple one. Analysts may use supertype/subtype hierarchies for the polymorphic transmission of events. This means that a 'sender' (typically an instance 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 sender of the event does not need to worry about the detail of which type (in the subtype sense) the destination object instance is.

Consider the following example:

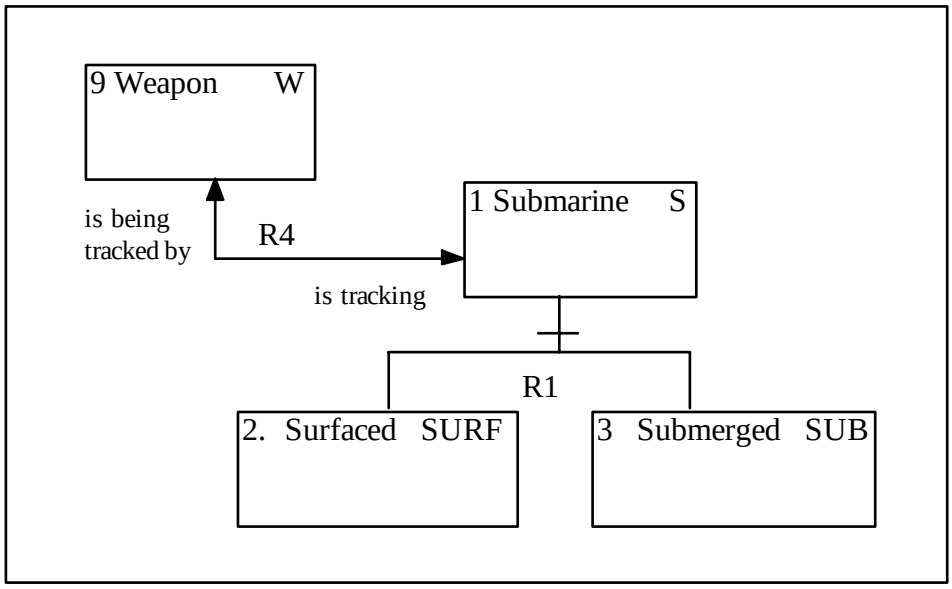

Figure 3 - Subtypes of Submarine

An action of "Weapon" can perform some activity such as:

```
tracked_sub = this -> R4
generate S1:weapon_assigned() to tracked_sub
```

With polymorphism, the event S1 will be received by the state model of either the Surfaced or the Submerged objects. The sender does not know, or care, which.

However, to be able to implement such polymorphism, we must fully define the rules for the processing of such events and in particular we must allow for:

- multiple layers of subtypes
- multiple subtype families
- state models in **both** the supertype and subtype objects
- subtype migration
- self-directed events

5.2 Specification of Polymorphic Events in OOA 92

In OOA events are “directed” at an object. This direction is formally captured through the event label. For example, the event if the previous section (“weapon_assigned”) has a label of “S1”. Since “S” is the key letter of the Submarine object we know that S1 is “directed” at the Submarine object.

With the exception of events causing a creation transition, events are not only directed at objects, but are targeted at specific instances of those objects. This is captured by the instance handle that is the subject of the “to” clause in an ASL generate statement.

In OOA 92 and ASL 2.5, the handle in the “to” clause must have the type of the object to which it is directed. Thus, in the example in the previous section “tracked_sub” must be an instance handle for a Submarine instance since S1 is directed at the Submarine object.

In OOA 92, events that are directed at supertype objects become automatically “available” to the subtype objects. The analyst need make no further declaration of this fact.

Thus, if S1 is declared as being an event directed at “Submarine” is automatically available to both the “Submerged” and “Surfaced” objects’ state models.

This means that if these subtypes have state models, then the event S1 is automatically a column in the STT and can be put on a transition in the STD:

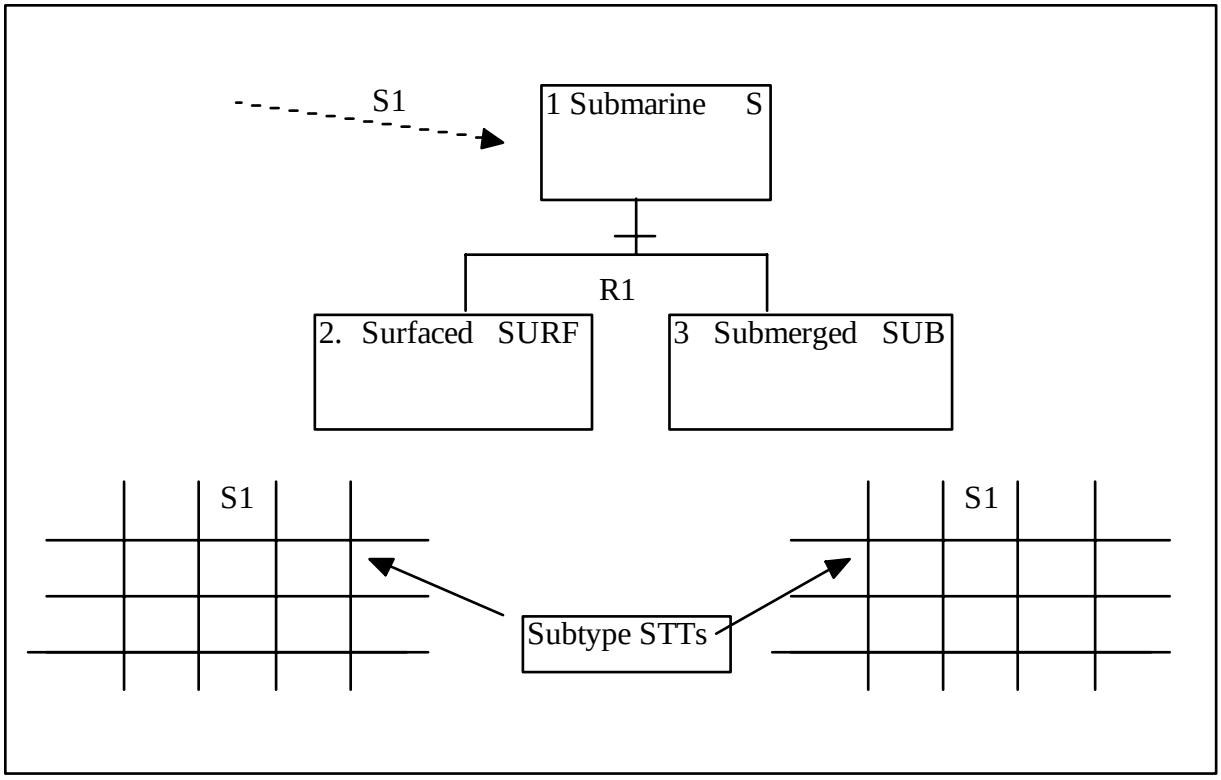

Figure 4 - Polymorphic Event Availability

Thus, at run time, when the event S1 is sent to an instance of Submarine, it is received by either the state machine for Surfaced or that for Submerged depending on which subtype the Submarine was at the time the event was received.

Notes:

- This polymorphism can only work for events targeted at instances since the architecture must know which subtype is involved. It cannot therefore operate for events causing a creation transition (since the instance does not yet exist) or for assigner events (since instances are not involved, and there can be no subtype assigner state models).

- This event availability is transitive. Thus, if “Submerged” were to have further subtypes, S1 would be automatically available to the state models of these objects also.

- Since both the supertype and the subtypes can have state models, any particular event can be received at more than one level in the hierarchy because the event becomes automatically available to all of the levels. This means at run time that any instance of an event may be responded to many times.

- If a the object at which the event is directed has multiple subtype hierarchies descending from it, then the event is available to all of these.

- If analysts do not wish an event to be received at a particular level in a hierarchy, they can simply declare the response to the event to be “Ignore” in every state.

- On the OCM the polymorphic event is shown going to the objects that receive it. Thus, in the above example, the event S1 is shown going from the sending object to both the Submerged and Surfaced objects (See Figure 5). If the Submarine object also had a state model, then the event would be shown going to that object as well.

- If the entire response of a given state model to the event is "Ignore" then the transmission of that event to the object/state model is not shown on the OCM.

- In a well formed model a non-ignore response must be defined somewhere for every possible configuration of subtypes in all of the hierarchies.

- I-OOA automatically maintains the appropriate presence of available events in all subtype state models. It also hides OCM transmissions to state models where the response is completely ignored.

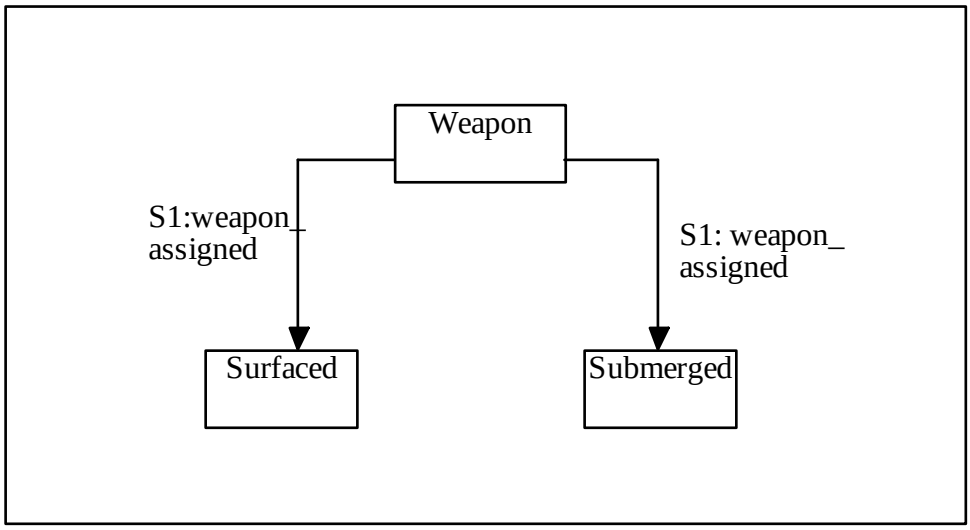

Figure 5 - Polymorphic Events on the OCM[5]

[5] Note: In OOA97, event transmissions on the OCM are shown with dotted lines.

5.3 Specification of Polymorphic Events in OOA 96

In OOA 96, an event directed to a supertype must be explicitly aliased to an event to be received by a subtype. This is achieved through the mechanism of a polymorphic event table (PET).

The PET shows the mapping between the senders view of an event (in our example S1) and the receiver's view. For example:

Sender	**Receiver**
S1	SURF1
S1	SUB7

Figure 6 - A Polymorphic Event Table

So that in our example:

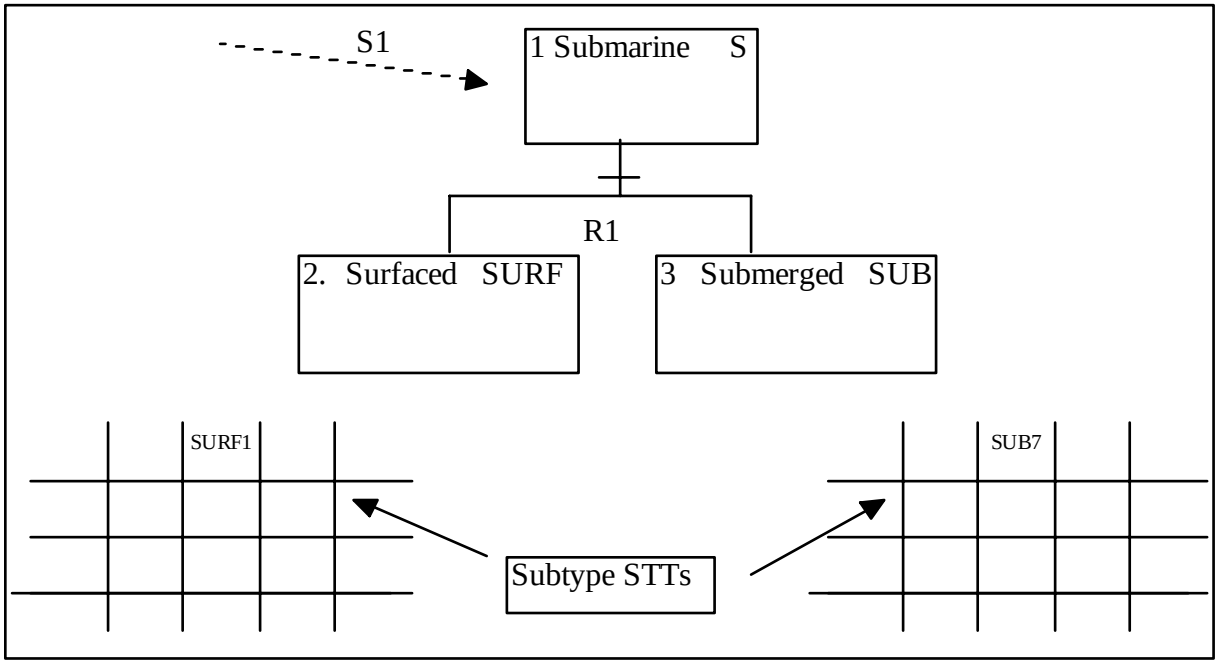

Figure 7 - Explicit Aliasing in OOA96

In addition, when sending the event, the fact that it is polymorphic is made explicit with an asterisk:


```
generate S1*:engaged() to tracked_sub
```

and the transmission on the OCM shows the mapping explicitly:

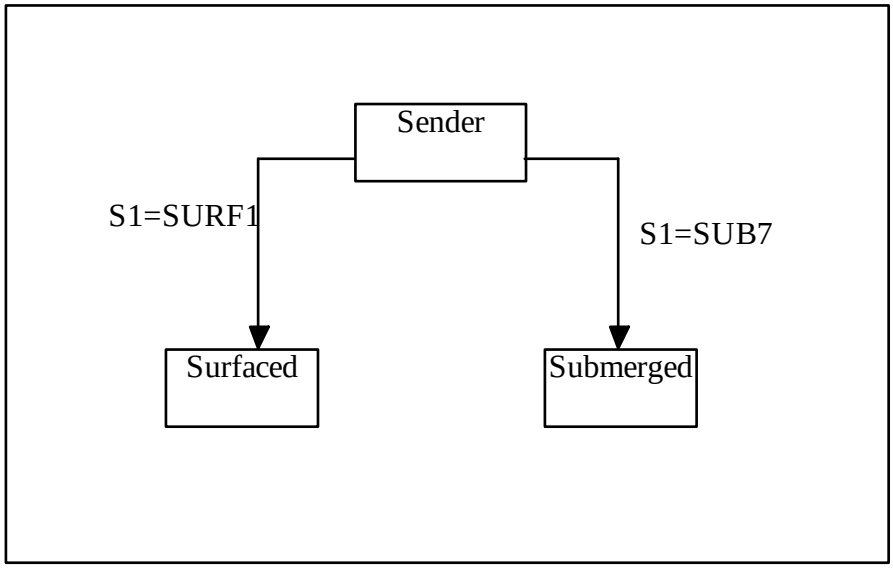

Figure 8 - Event Aliasing on an OOA96 OCM

Notes:

- The form of the PET allows for events to be aliased to more than one event in a given destination state model. This would be an ill-formed model.

- The form of the PET permits an event to be aliased both to a supertype event and to a subtype event. This would allow reception by both the supertype and the subtype in the manner of OOA 92. However, the event processing rules for this situation are not defined.

6. Polymorphic Event Processing Rules in OOA 92

6.1 Reminder of Standard Event Processing Rules

In general, event ordering cannot be guaranteed in OOA. The sender of an event cannot assume anything about when an event is processed, or the order in which events are processed except as specifically allowed for in certain circumstances. Thus:

Rule 1: *Except as explicitly allowed for by other rules, a model may make no assumptions about when or in what order events are processed.*

However, OOA 91 included a rule which regulated the order in which events were consumed by a state machine as follows:

Rule 2: *If a state machine generates multiple events to a **single** receiving instance, the events will be processed in the order they were generated.*

OOA 92 and OOA 96 introduced the idea of a self-directed event, and augmented the event ordering rules with:

Rule 3: *When a state machine sends an event to its own instance (a self directed event), that event will be processed before any other event for the instance*

6.2 Simple Dispatching of Polymorphic Events

In the simple case of reception by a single state machine, and where there is no subtype migration, there is little that needs to be said about the details of the reception of a polymorphic event. The sender sends the event which is, in due course, received and processed by the appropriate subtype. We need only clarify Rule 2 as follows:

Rule 2a: *If a state machine generates multiple events to a single receiving supertype instance, the events will be processed by the subtype in the order in which they were generated*

6.3 Reception by Multiple State Machines

As we discussed above, the OOA 92 formalism (and, indeed, OOA 96[6]) allows a specification that will result in a polymorphic event being consumed more than once. As a result we must extend the event ordering rules.

First:

[6] In OOA 96, an event would be mapped to two *different* events in different state models at different levels in the hierarchy. Thus, the original event is not consumed twice as such. However, the effect is the same and the ordering rules must be addressed.

Rule 4: Polymorphic event processing proceeds top-down from the instance to which the event was directed to all respective subtypes in all subtype families

This means that the event is first processed by the supertype state machine (if it exists) then by the appropriate subtype state machine, then by any further subtypes of that and so on.

Note that if the supertype has multiple subtype families, then processing proceeds down each hierarchy separately.

Justification for Rule 4: The ordering must be guaranteed to avoid excessively complicated state models. Bottom up cannot be used because there might be multiple subtype hierarchies, in which case we would end up with the event being consumed twice by the common supertype.

For similar reasons it is important for the analyst to know when the event is forwarded to the subtype:

Rule 5: The event is forwarded to the appropriate subtype after the completion of the action in the supertype that resulted from the consumption of the original event.

Note that:

- The forwarding occurs even if the event is ignored in the supertype
- The forwarding occurs even if there is no state model at the supertype level
- At any level of a subtype hierarchy, forwarding will be paused if the event is held by a supertype, and resumed again only after the supertype has consumed the event.
- In a hierarchy containing multiple subtype families, the event may be held simultaneously by multiple supertypes.
- A supertype will only forward the event to its subtypes once it has consumed the event. A held event will, therefore, not be forwarded until
- The forwarding occurs transitively down the hierarchy

For the avoidance of doubt we now further refine Rule 2:

*Rule 2b: If a state machine generates multiple events to a single receiving supertype object instance, then the events will be forwarded to, and received by **all** of the subtypes in all of the supertype/subtype hierarchies descending from that supertype instance in the order in which the events were generated.*

and introduce another

Rule 6: *When a supertype has multiple subtype hierarchies descending from it, nothing can be assumed about the order of processing of events in one hierarchy relative to another.*

6.4 Polymorphic Events and Self-Directed Events

Consider the following example:

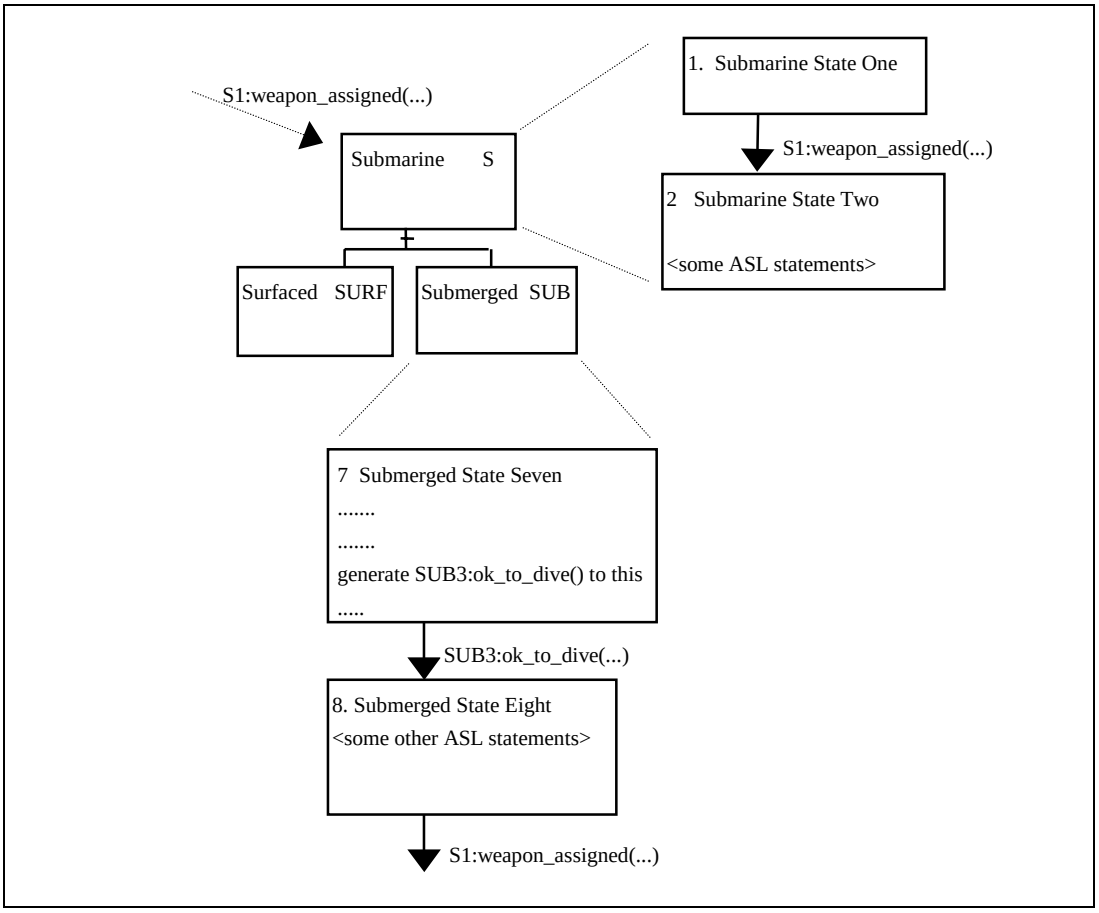

Figure 9 - State Machines Processing Polymorphic Events

Suppose that an instance of the Submarine object which has a corresponding instance of Submerged receives the polymorphic event S1 whilst in state 1.

When the event is consumed it causes a transition into state 2 and executes the action defined for that state. At the end of the action the polymorphic event is ‘forwarded’ to the corresponding Submerged instance, which is **currently** executing the action associated with state 7. Within this action a self directed event is unconditionally generated which the analyst intended should cause a subsequent transition into state 8.

In the above scenario, after completing the action associated with state 7 there will be two outstanding events for the Submerged instance, S1 (the polymorphic event forwarded by the supertype), and SUB3 (the self directed event).

The OOA 92 event rules define that if an instance sends an event to itself, that event will be accepted before any other events that are outstanding for that instance. Thus, in the above example, the self-directed event will be consumed first and the above state model for Submarine would be an accurate abstraction of the run-time behaviour - since the analyst never has to deal with the possibility that the event S1 is accepted in state 7 of the Submarine state model.

The question is should the event rules be modified such that polymorphic events take priority over the self-directed events ?

Let us consider the effect on the subtype state model if the polymorphic event is accepted before the self-directed event:

Assume the same scenario as before - i.e. that the event S1 is consumed by the Submarine state model which makes the transition from state 1 to state 2 and executes the action associated with that state. Upon completion of the action, the event S1 is polymorphically forwarded to the corresponding subtype instance - in this case the corresponding instance of the Submarine subtype - which is currently executing the action associated with state 7, which of course results in the self-directed event being placed on the state machine’s queue, except that this time it is placed behind the polymorphic event.

Now the analyst has to deal with the possibility of the two events being processed in **either** order (because the events S1 and SUB3 are from different sources), which results in the need for additional states and transitions:

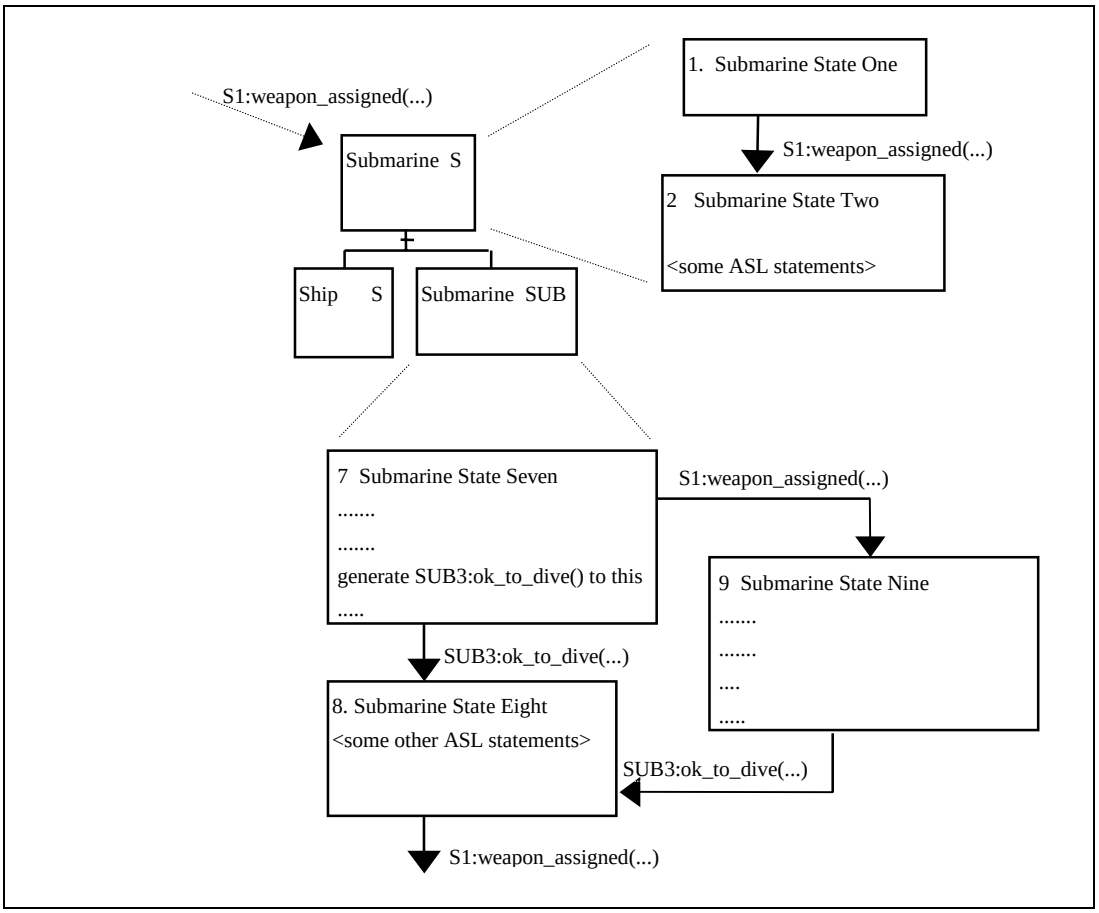

Figure 10 - Complexity due to Polymorphic Event Priority

The state model for the subtype has become unnecessarily complex even to deal with this fairly simple problem - the situation worsens when you take into account the possibility of a second polymorphic event needing to be processed whilst in state 9 - the number of permutations that the analyst potentially has to deal with makes the problem almost impossible to specify without heavily constraining the processing in other parts of the OOA model.

To avoid this, the self-directed event rule should be considered to always take precedence over polymorphic events.

Thus:

Rule 7: A subtype will always process its own self directed events before processing any polymorphically received events

6.5 Polymorphic Events in the Presence of Subtype Migration

Subtype migration could, with ill formed rules, cause the failure to deliver a polymorphic event. This is a different situation from the regular OOA problem of an event being sent to a deleted instance, since in the polymorphic case delivery failure

may occur even although the supertype instance at which the event was originally directed still exists.

Thus we have an additional rule:

Rule 8: *A forwarded polymorphic event must always be delivered to at least one subtype at each level as long as the supertype existed at the point at which the event was first received by the supertype at this level.*

To respect this rule in the presence of every possible combination of subtype migration and self deletion by the supertype action, we propose the following strategy:

The event is forwarded to a subtype instance determined by the following rule:

```
Define:
     “Initial Subtype” to be the subtype instance
     related to the supertype at the start of the
     supertype state action
     “Final Subtype” to be the subtype instance
     related to the supertype at the start of the
     supertype state action
If, at the end of the supertype action the supertype
still exists then:
     Forward to the “Final Subtype”
else
     If the “Initial Subtype” still exists then:
          Forward to the “Initial Subtype”
     endif
endif
```

We recognise that this will still not respect Rule 8 in the presence of the pathological patterns of the supertype action performing a synchronous subtype migration followed by synchronous supertype deletion.

7. Conclusions

We have presented a summary of polymorphism in OOA 92 and OOA 96. This has described the way in which the models are defined and the run time event processing rules.

These rules become complicated in the case where multiple reception of an event is permitted. This may lead to considerable complexity in the presence of subtype migration.

8. References

The following is a list of documents that are relevant to the OOA/RD method and to the I-OOA product suite. Many of the Kennedy Carter documents are updated frequently, so interested parties should visit the Kennedy Carter web site to get the latest versions, or contact Kennedy Carter directly.

The OOA/RD Method:

[M1] Ovum Evaluates Ltd.

Guide to Shlaer-Mellor

Ovum Ltd, 1 Mortimer Street, London, W1N 7RH, United Kingdom. Telephone (+44) 171 255 2670

[M2] Sally Shlaer and Stephen J. Mellor

Object Oriented Analysis: Modeling the World in Data

Prentice Hall, 1988

[M3] Sally Shlaer and Stephen J. Mellor

Object Oriented Analysis: Modelling the World in States

Prentice Hall, 1992

[M4] Leon Starr

How to Build Shlaer-Mellor Object Models

Prentice Hall, 1996

[M5] Christopher Raistrick and Colin Carter

OOA: A Formalism for Understanding Software Architectures

Kennedy Carter, 1994

[M6] Christopher Raistrick

A Practitioners Guide to the Use of Bridges in Shlaer-Mellor Recursive Development

Kennedy Carter, 1994

[M7] Ian Wilkie
A Proposed Policy for the Handling of Errors and Exceptions in OOA/RD
Kennedy Carter (KC/OOA/CTN 42)

[M8] Ian Wilkie
A Proposed Policy for the Implementation of Deferred Data Types using non-OOA Domains in OOA/RD
Kennedy Carter (KC/OOA/CTN 43)

[M9] Ian Wilkie
A Proposal for Formalisation of Synchronous Behaviour in OOA/RD
Kennedy Carter (KC/OOA/CTN 44)

[M10] Colin Carter and Ian Wilkie
A Proposed Policy for the Implementation of Dynamic Data Types in ASL
Kennedy Carter (KC/OOA/CTN 45)

[M11] Ian Wilkie
A Proposal for Modified Event Processing Rules in OOA/RD
Kennedy Carter (KC/OOA/CTN 46)

[M12] Ian Wilkie, Colin Carter and Paul Francis
Bridges in OOA/RD
Kennedy Carter (KC/OOA/CTN 47)

[M13] Ian Wilkie, Colin Carter and Paul Francis
OOA 97
Kennedy Carter (KC/OOA/CTN 53)

[M14] Ian Wilkie, Adrian King, Mike Clarke and Chas Weaver
The Action Specification Language (ASL) Reference Manual
Kennedy Carter (KC/OOA/CTN 06)

CASE Tools:

[C1] Butler Bloor Ltd.

Case and methods based Development Tools - An Evaluation and Comparison Report

Butler Bloor, Challenge House, Sherwood Drive, Bletchley, Milton Keynes, MK3 6DP, United Kingdom. Telephone (+44) 1908 373311.

[C2] Ovum Evaluates Ltd.

Intelligent OOA Evaluation

Ovum Ltd, 1 Mortimer Street, London, W1N 7RH, United Kingdom. Telephone (+44) 171 255 2670

[C3] Christopher Raistrick, Ian Wilkie and Mark Ellis

CASE Tool Guidelines for Evaluating a Shlaer-Mellor Object-Oriented Analysis and Recursive Design. CASE Tool

Kennedy Carter (KC/OOA/CTN 28)

Intelligent OOA Product Suite:

[P1] Ian Wilkie

Pre-Release Notes for Intelligent OOA Release x.x

Kennedy Carter (KC/OOA/CTN 17)

[P2] Ian Wilkie and Mark Ellis

I-OOA Analysis Tool Technical Overview

Kennedy Carter (KC/OOA/CTN 37)

[P3] Ian Wilkie

I-SIM Testing and Integration Tool Technical Overview

Kennedy Carter (KC/OOA/CTN 55)

[P4] Ian Wilkie, Adrian King, Mike Clarke and Chris Raistrick
The Intelligent OOA Strategy for Configurable Code Generation

Kennedy Carter (KC/OOA/CTN 27)

[P5] Colin Carter and Ian Wilkie
Software Architecture Requirements Technical Overview
Kennedy Carter (KC/OOA/CTN 36)

[P6] Ian Wilkie
OOA Architecture Test Suite Technical Overview
Kennedy Carter (KC/OOA/CTN 48)

[P7] Mark Ellis and Ian Wilkie
Architecture 1 (TA-1) Technical Overview
Kennedy Carter (KC/OOA/CTN 41)

[P8] Adrian King, Ian Wilkie and Colin Carter
Architecture 2 (TA-2) Technical Overview
Kennedy Carter (KC/OOA/CTN 40)

[P9] Ian Wilkie
Intelligent OOA Supported Platform Configurations
Kennedy Carter (KC/OOA/CTN 18)

