IntelliCorp, Inc.

Response to the OMG Analysis and Design Task Force

UML RTF 2.0 Request for Information

Suggested Enhancements for UML

Prepared by: Conrad Bock
IntelliCorp, Inc.

OMG Document ad/99-12-02

Response Date: 17 December 1999

Contributors: Guus Ramackers
Oracle Corporation

Contact: Conrad Bock
IntelliCorp, Inc
1975 El Camino Real West, Suite 201
Mountain View, CA 94040-2216
(650) 965-5720
bock@intellicorp.com

Table of Contents

1. INTRODUCTION **3**

2. RFI QUESTIONS **3**

2.1 NEED FOR MAJOR UML REVISION 3
2.2 ROADMAP RECOMMENDATIONS 3

3. STATIC ELEMENTS **4**

3.1 RELATIONSHIP 4
3.1.1 Contextualized association in composites 4
3.1.2 Association specialization 8
3.2 EXTENSION MECHANISM 9
3.2.1 First-class extension mechanism 9

4. BEHAVIOR ELEMENTS **10**

4.1 COMMON BEHAVIOR 10
4.1.1 Publish/subscribe 10
4.2 ACTIVITY GRAPH 11
4.2.1 Activity graphs independent of state machines 11
4.2.2 Goal/result modeling 11
4.2.3 Event consumption 13
4.2.4 Separation of control and object flow 13
4.2.5 Branch to path that does not synchronize 15
4.2.6 Events starting a process 16
4.2.7 Starting thread without fork 17
4.2.8 Transformation of values from output parameters to input 18
4.2.9 Notation for object flow between input/output parameters 19
4.2.10 Concise notation for showing the type of object used in a call state 20
4.3 STATE MACHINE 20
4.3.1 Interruptible actions 20
4.3.2 Explicit completion events 20
4.3.3 Boolean event combination 21
4.3.4 Parallel event dispatch 22
4.3.5 Models of execution-time behavior 23
4.3.6 Parameterized state machines 24

5. APPENDIX: UML 1.4 ISSUES **25**

6. REFERENCES **28**

1. Introduction

This document is submitted to the OMG's Analysis and Design Task Force (ADTF) in response to the Request for Information (RFI) entitled "UML 2.0 RFI." It suggests text for the UML 2.0 RFP on various issues and presents background information for each. Specific proposals are not given here, only a general sketch of the requirements or problem, issues around that, and area in which the requirements or problem should be solved.

If any of these topics can be handled as part of UML 1.4, then let please us know. Conversely, an appendix (section 5) is provided listing the issues we plan on submitting to the UML 1.4 RTF. If any of these appear to be major revision, also please let us know.

2. RFI questions

We were involved with the UML 1.3 and contributed to the Road Map recommendations it gave for the UML 2.0, which we fully support (see section 2.2). Consequently we have focused in this document on issues that are of particular concern to us. In the interests of brevity we are answering only one question posed by the RFI (see next section).

2.1 Need for major UML revision

A major of revision of UML for business modeling is definitely required. The experience with the OMG workflow group is a case in point. They evaluated UML and decided to omit it from the first draft of their RFP for a process definition language. Each group concerned with business systems is in danger of likewise rejecting UML and requiring, as in the case of the workflow group, damage control on the part of UML committee members. This of course will be a severe drain with the increase in number of other groups considering UML.

Business modeling is a very large market for UML and requires attention to its particular requirements. This is not simply a matter of education. We know from language design that any Turing-equivalent machine can, with enough effort, be used for any application. The same can be said of UML's behavior models. However, UML is an *industrial* standard and is consequently is not concerned with theoretical results. Practical language design requires attention to the specific use cases for that language, and crafting of the language for its particular purpose. Many of the suggestions in this document are made with that goal in mind.

2.2 RoadMap recommendations

We fully support the areas of revision listed in Appendix A of the RFI, even though we have not addressed all of them in this document:

1. Architecture
   - Physical metamodel
   - Guidance on extending UML
   - Kernel cleanup
2. Extensibility
   - First-class extensibility mechanism
   - Improved profile specifications
3. Components
   - Improved semantics and notation for components
4. Relationships
   - More complete semantics for refinement, trace, and composite aggregation.
5. Behavioral Modeling
   - State machines and activity graphs
     - Activity graphs independent of state machines
     - More permissive concurrency
     - State machine generalization
   - Collaborations
     - More complete semantics for patterns
6. Model management
   - Refine notation and semantics of models and subsystems for enterprise architecture.
7. General mechanisms
   - Model versioning
   - Diagram interchange

3. Static Elements

3.1 Relationship

3.1.1 Contextualized association in composites

Figure 1 shows a very common kind of class diagram that is not currently supported in UML. It specifies a composite class that is constructed by associating its parts, as you might find in any CAD diagram. In particular it models that engines used in cars are restricted to powering wheels, but restricted to propellors in boats. The general associations used in these particular examples are shown in Figure 2. Such CAD-style diagrams are semantically equivalent to the following models:

1) Each association between parts that appears in the diagram, for example the POWER association between ENGINE and AXLE, can be translated to a constraint on the composite class that restricts the general association to be used as indicated in the diagram when it is applied to the parts of the composite class.

2) Each part (class) that appears in the diagram, like ENGINE, can be translated to a special subclass used only in this diagram in that particular place. For example, ENGINE can be translated to CAR-ENGINE or BOAT-ENGINE. We call these *role types* or *qua types*, as shown in Figure 3, [2]. The role type names are notated in parentheses above the classes in Figure 1. Then the general associations, like POWER, can be specialized to operate between the role types.

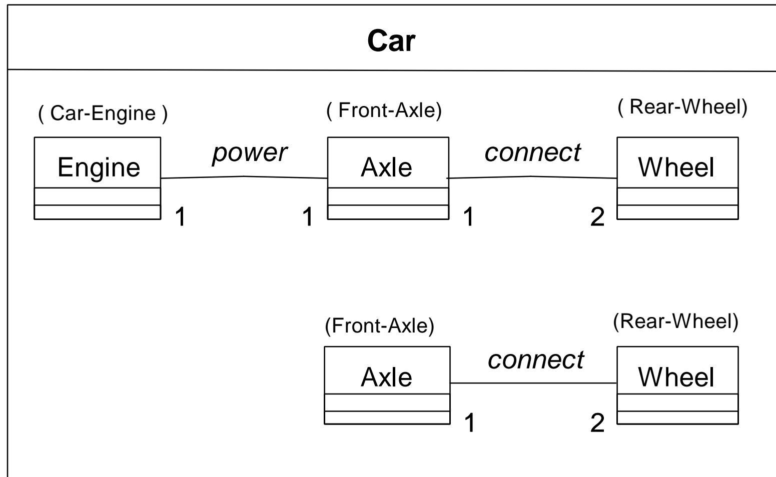
In either case, the user should be able to diagram in the simple way shown in Figure 1, rather than dealing with the complexity of constraints or role types.

From a semantic point of view, the second model has the advantage that it reuses association specialization functionality, which will probably be available in UML 2.0 (see section 3.1.2). For example, the multiplicities between ENGINE and POWERTRANSMITTER are specialized to be more restrictive in cars. This is more explicit semantics than using a constraint note, and certainly much simpler, especially when a class may play different roles in the composite, or use the same association in different ways in each role. For example, the CONNECT association may between WHEEL and AXLE may have different properties for front and back wheels/axles. One might try collaborations as a semantics, but a collaboration instance does not require the associations to be instantiated, just that they are used to send messages to other objects. Perhaps collaborations could be enhanced to handle contextual associations and composition.

Since attributes are conceptually a form of association, as the UML documents explain, all the above discussion applies to attributes as well. For example, a dialog box may be modeled as a composite class, so that multiple versions of it can be opened at the same time by instantiating the class. The controls in each dialog box class have certain values or restrictions on their attributes, for example, a particular button may be red or a particular slider may range from 0 to 100. These are modeled as contextualized initial values and attribute types. Normal associations can be used in this application also, for example to model the tabbing order between controls.

UML cannot currently support these sort of models because it does not treat composition much differently than other sorts of associations. The discussions on composition up to now omit a most essential and common fact: that composites set up contexts for the various associations within it, so that statements about associations apply only inside the composite. The best a modeler can do in UML currently is shown in the Notation Guide, Figure 3-36, page 3-77, which diagrams the parts of a composite class, but no associations between them. In addition, any attribute initial value applied to one of the part classes, the width of a SLIDER for example, would affect all sliders in all windows, not just the window being modeled.

Contextual association models have been built and used in customer applications and products at IntelliCorp since 1985. There have been at least four successful implementations. It is a well-proven modeling approach.



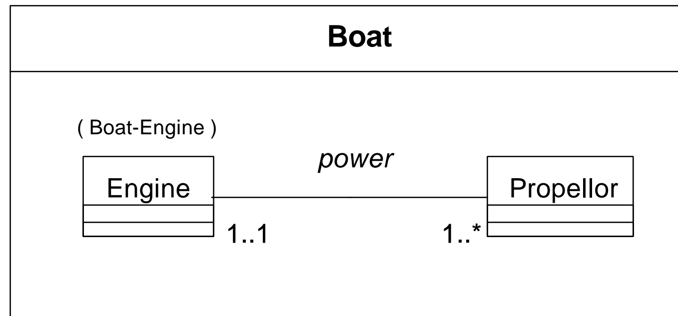


Figure 1: Contextual associations

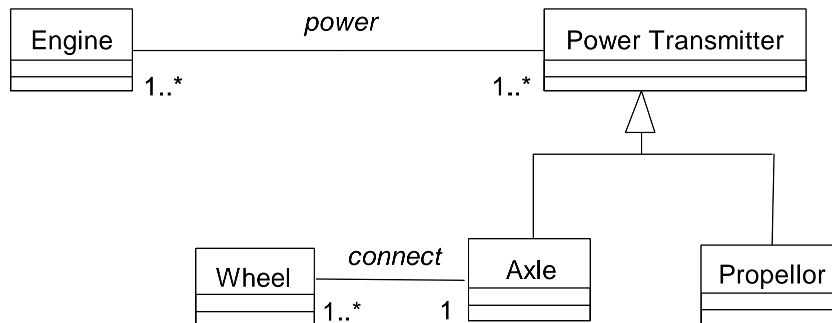


Figure 2: General associations contextualized in the previous figure

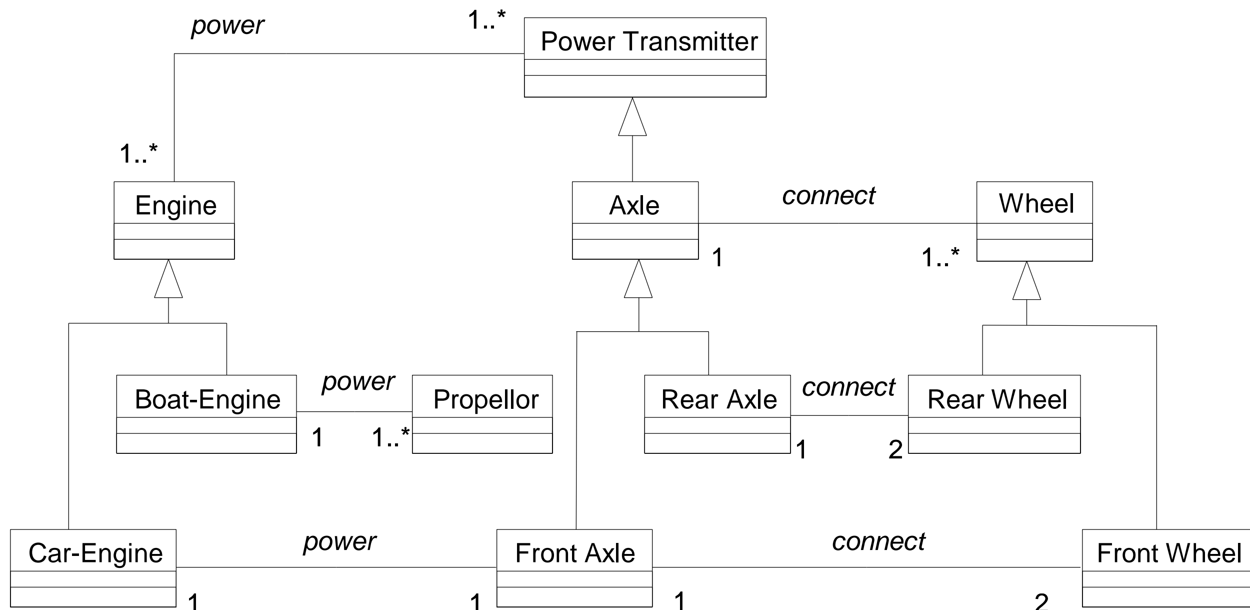


Figure 3: Role type or qua types

Suggested text for RFP:

The proposal should suggest changes that support more straightforward modeling of associations and attributes in the context of a composite class. It should be possible to draw a class diagram for a composite class showing its parts (as classes), associations between these parts, and initial attribute values on these parts, all without constraining how those kind of parts are associated or attributed in other composites.

For example, it should be possible to model a car as having an engine associated with its wheels, at the same time using the engine class in the model of a boat that associates the engine with a propellor. It should be possible to model a class of dialogs, with its controls given initial values, such as the range of a slider, that do not affect the use of those same types of controls in other dialog classes. It should be possible to use the same class more than once in a composite, having different associations and initial attribute values in each case, or even different properties of the same association or attribute. See Bock, Conrad, and James Odell, "A Foundation for Composition," *Journal of Object-Oriented Programming*, 7:6, October 1994, pp. 10-14.

3.1.2 Association specialization

Substitutability is a central issue to association specialization, because the specialized types do not support the same navigation functionality as the generalized ones. For example, suppose an association between CAR and DRIVER is specialized to RACE CAR and RACE CAR DRIVER, with the restriction that only race car drivers can drive race cars. Then programs manipulating drivers and cars cannot link just any driver to any car, as implied by the model. The programmer may not even be aware that CAR and Driver are specialized by some other model. The proposal should address this issue.

The proposal should take as use cases the kinds of association specialization given by McCarthy in the July/August 97 issue of *Journal Of Object-oriented Programming,* page 69. Especially see his figure 13, page 75. The article has many other good points, such as rules for specialization of role properties such as multiplicity, ordering, and qualifiers.

Specialization of association need not imply restriction of objects on all ends of the association, but may restrict only any or no ends (weak and strong roles in McCarthy's terminology, and see McCarthy's figure 13(b-c)). If neither end is specialized, then only properties of the association are specialized, such as multiplicity.

Since attributes are conceptually a kind of association, as the UML documents explain, any changes for association specialization should also be made for attribute specialization.

McCarthy takes the formal view that a specialized association is a different association than the parent association. In particular, he supports specialized associations with different (end) names than the parent association. This would be analogous to renaming an inherited attribute. There are good reasons to do this at the analysis level, for example, the "nose" of a mammal is called a "trunk" on an elephant, and it reflects the general rule that no modeling element should be identified by its name. However, programming languages normally use the end name to traverse associations, and need to access the same association on objects that happen to be subtypes of the end types without being concerned about end name changes.

The name-change feature is a good idea, but would require that language compilers and interpreters know about it, and transparently support substitutability, either by translating the name, or providing a way to uniquify the end name. If we omit this feature, then a number of McCarthy's issues go away, for example, concerning abstract associations. On the other hand, omitting the name-change feature means the model cannot support specializing associations between the same two classifiers (see McCarthy's figure 13(c)). The proposal should address this issue.

Suggested text for RFP:

> The proposal should suggest changes that support modeling of association specialization. It should support:
>
> 1. rules for inheritance of all the meta-features of associations and association ends, such as multiplicity, ordering and so on.

2. specialization of any, all, or none of the classifiers participating in a specialized association.
3. cases where some or all of the classifiers associated by the general and specific associations are collapsed, for example, when one of the classifiers participating in the general association also participates in the specialized association.

The proposal should address the question of substitutability in association specialization. For example, suppose the association between cars and drivers is specialized to race cars and race car drivers, with the restriction that only race car drivers can drive race cars. Then the general association between cars and drivers cannot be manipulated as freely as the model says, because not all drivers can drive all cars.

The proposal should address the issue of whether specialized associations can have different (end) names than their generalizations. Some benefits are that terminology can be specialized with the associations, and association specialization would not require the end types to be specialized. A drawback is that it impairs substitutability, that is, applications using the model will not have consistent association names to use everywhere in the type hierarchy.

The proposal should suggest a graphical notation and an alternative textual annotation.

Submitters are encouraged to review McCarthy's "Association Inheritance and Composition" in the July/August 97 issue of *Journal Of Object-oriented Programming*, page 69. Especially see figure 13, page 75, for examples of requirements 2 and 3 above.

3.2 Extension Mechanism

3.2.1 First-class extension mechanism

The informal term for this topic, "heavy-weight" extension mechanism, may have the connotation that it will be harder to use than the light-weight ones. However, the new mechanism should simply be UML itself, and require little additional training to apply. The problem with the current mechanisms from a user's standpoint is that they define ways of doing things that are available in UML without them, namely subtyping (stereotyping) and attributes (tagged values). The fact that these things are done at a different meta-level should not require a completely new language. Proposals should draw on research in self-reflective languages to define a simple way to extend UML using the core (static) UML itself.

Suggested text for RFP:

The proposal should suggest changes that support modeling of extensions to UML using the Core and Datatype packages of UML. It should not define a new

language for existing capabilities in UML, regardless of the meta-level on which those capabilities are applied.

4. Behavior Elements

4.1 Common Behavior

4.1.1 Publish/subscribe

Sending signals explicitly between objects leads to systems with too much coupling between their parts. Unfortunately, this is designed into UML because objects producing signals usually must know who is interested in those signals, and send them explicitly to those objects. The only exception currently is the "all" target available in send actions (see ObjectSetExpression). This sends the signal to all instances that can receive it, as determined by the underlying runtime system, which is not modeled.

Receptions can be used to model which objects will receive (subscribe to) signals broadcast this way, but these do not express which particular instances of the class actually want to receive the signal at any particular time, or which senders of the signals are acceptable. Signal events likewise are only associated with the signal they are subscribing to, not which objects in particular they wants to receive the signal from. Guards could express this restriction, but it requires that the event pass the source of the signal as a parameter, and in any case, requires repeating the same guard in many places in the state machine if the interesting objects to not change.

The above comments apply to change events also, and perhaps call events if they can apply to operations on objects other than the one a state machine is attached to. One could model the subscribed objects in this expression, it may require repeatedly entering the restriction on the monitored objects if this is constant across the state machine.

It would significantly enhance UML's publish/subscribe capability if modelers could control event subscription to at a more fine grained level. There are various possibilities:

- An action language could have a command for (un)subscribing a particular instance to some set of signals from another instance.
- A class could specify an expression that is evaluated to determine which signals, possibly on which objects, that an instance is interested in being notified about. This could be applied to change events also.
- A class could specify that all its instances are interested in some set of signals or changes from all the instances of another class.

The action language would need to be enhanced to refer to the currently subscribed objects/events. There are probably other subscription schemes that respondents can think of.

Suggested text for RFP:

> The proposal should suggest changes that support modeling of more specific event subscription than is currently available with receptions. Specifically, the changes should support specification of limits to the particular events that an instance can receive and which objects it can receive them from. For example, a class might specify that all its instances are interested in some set of signals from all the instances of another class. Or an expression might be specified that determines more precisely which instances of the other class are interesting. These changes will work in conjunction with the various broadcast mechanisms in UML, such as changes that are detected by change events, and the "all" target ObjectSetExpression for SendAction, which functions as the publishing specification for signals.

4.2 Activity Graph

4.2.1 Activity graphs independent of state machines

There are many difficulties regarding business modeling that are due to activity graphs being a specialization of state machines. Specifics of these are given in the following sections. However, the RFP should contain a general request to address this issue at the meta-model architecture level. As argued in the introduction, Turing-like universal behavior models are inherently unusable for industrial applications. It is not intended that UML surface to the user a "super" behavioral model that covers both state machines and activity graphs. So the primary question remaining is whether state machines and activity graphs should have a common supertype for architectural reasons in the meta-model, and if, so how the collaboration model relates to this supertype. There are benefits to a generic behavior model in that it provides some continuity and comparability between the more specific models. The responses should address this question and make a recommendation.

Suggested text for RFP:

> The proposal should suggest changes that support modeling of activity graphs independently of state machines. It should specify an unambiguous execution semantics for activity graphs. The model and execution semantics should be close to the activity diagram notation and not require complicated mappings to the underlying model.
>
> The proposal should also address whether the UML meta-model should have a common supertype between state machines and activity graphs. It should explain the benefits and disadvantages of both approaches, and how the collaboration model would relate to a common supertype.

4.2.2 Goal/result modeling

It is common in business modeling to focus first on the goal or results that a behavior is meant to achieve, before addressing what the behavior is exactly or who/what will be responsible for it. This could be modeled currently in with postconditions on operations, since operations can be specified without a classifier owning them. However, there is no concise, standard notation for postconditions in state or activity diagrams. The proposal should address this. It should also be able to notate and model operations that have multiple postconditions and have different transitions associated with each postcondition. This explicitly models which effects are associated with which causes.

In some popular business modeling methodologies the goal/result of a behavior is modeled as an event that it brings about [7][8]. The event triggers other behaviors, as in state machines. There is some ambiguity currently in these methodologies, because it isn't clear whether an operation A invoked in one diagram, resulting in say event E, is meant to trigger the transitions following E in all diagrams. The proposal should resolve this ambiguity.

UML currently uses object flow for modeling cause and effect. For example, fixing a roof is an activity with at least two facts true at the end, namely the roof being fixed and the workers being free for other jobs. Each result has a different effect, namely billing the customer and assigning the workers to new jobs. Normally business models will give the roof-fixing activity two resulting events, and link each to the effect that they have. In UML, the resulting events are modeled with an object-flow state that has a signal as its object (see Figure 4). This means the roof-fixing activity outputs two signals, and the billing and worker-reassignment activities take these as inputs. But a different business process may need to bill the customer before the roof is fixed, or reassign workers that are in the middle of a job. So BILL CUSTOMER should take a job description, not a fixed roof. Likewise, fixing a roof should not be required to output the workers that are free, just because in this particular case a later operation requires this information. The UML currently forces business modelers into defining activities that are bound to their usage, thereby impairing reusability.

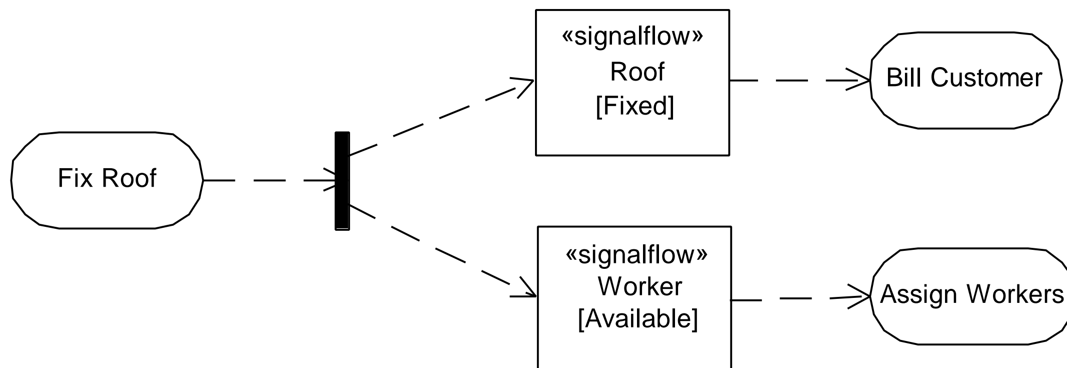


Figure 4: Modeling Resulting Events in UML

Suggested text for RFP:

> The proposal should suggest changes that support modeling and notation for the conditions that an action or subactivity state is expected to bring about. This might be modeled as a postcondition of the operation invoked by an action, or an

event, for example. Actions/subactivities that have multiple resulting conditions can have a different transition for each condition. The proposed model might express an event-driven semantics, that is, the resulting event for an action triggers the next action in the activity diagram. In this case, it should specify the semantics for the application that has multiple activity diagrams containing the same resulting event. See:

> Martin, James, and James J. Odell, *Object-Oriented Methods: A Foundation* (UML edition), Prentice Hall, Englewood Cliffs, NJ, 1998.
>
> Keller, Gerhard and Teufel, Thomas, *SAP R/3 Process Oriented Implementation: Iterative Process Prototyping*, Addison-Wesley, 1998.SAP

4.2.3 Event consumption

State machines currently have little memory of events. Once an event is dispatched from the queue and used to trigger a transition, it is lost, even if it had been previously deferred. Many applications cannot afford just one response to an event, such as in a business environment. Businesses need a history of events on which to base their decisions. It would be very useful if the events dispatched in a state machine or activity graph were kept, and triggers were able to reference them. These past events could have time stamps, and be managed in some way by explicitly removing them, or having an automatic expiration policy. It may be useful to let state machines inspect each other's past events, or post the past events to a globally accessible location.

Suggested text for RFP:

> The proposal should suggest changes to activity graphs that support keeping a history of events that have already been dispatched. It should address how these past events are managed, such as by time stamps, manual or automatic removal on expiration, and so on. It should address the question of whether the past events are accessible only by the activity graph that receives them, or whether they are globally accessible.

4.2.4 Separation of control and object flow

Control flow and object flow need to be separated semantically, so one does not imply the other as they do now in UML, where both are modeled as state transitions. Control and object flow differ in the relative emphasis they place on determining inputs to a behavior versus when that behavior starts. Control flow emphasizes that one step in the behavior starts when another is finished, regardless of whether inputs are available. Object flow emphasizes that a step requires certain inputs in a complete form before the step is allowed to start. This is a completely traditional and common-sense distinction [4][5].

For example, Booch's version of an activity diagram is not allowed in UML currently (see Figure 5 below, reproduced from p 270 of his User's Guide). The two transitions coming

from SHIP ITEM are both state transitions in UML, and their targets are states. This means two states are being activated at the same time, namely RECEIVE ITEM and ITEM[RETURNED], a situation not supported in state machines without explicitly declared parallelism. Also the ITEM[AVAILABLE] has no path out to the final state, which means the machine will never terminate.

Presumably Booch meant that SHIP ITEM outputs ITEM [RETURNED], which in turn is passed farther downstream to RESTOCK ITEM. However, RESTOCK ITEM does not start just because ITEM [RETURNED] is available. It only starts after RECEIVE ITEM is finished, using ITEM [RETURNED] as input. Booch uses the solid and dashed lines as if they had different semantics, instead of both being state transitions as they are currently. To get the semantics he wants, Booch would need to introduce a cumbersome set of forks and joins (see Figure 6).

Object flow refers to the passing of objects/data from one action/subactivity state to another. The only time it could possibly imply control is for action/subactivity states that are not the target of any control transition. In this case, the action/subactivity state might be entered when all the incoming object flows are available. Likewise control flow refers to when an action/subactivity state is entered. The only time it should be possibly be concerned with the inputs to a state is when the state is the target of object flow transitions as well. In this case, the action/subactivity state might be entered when all incoming flows, control and data, have arrived. In any case, the proposal would be expected to specify the rules for interaction of control and object flow.

Suggested text for RFP:

> The proposal should suggest changes to activity graphs that support distinct constructs for control and object flow. The proposal should specify how control and object flow interact. For example, perhaps a state that is the target only of object flow transitions can be entered when all objects are available.

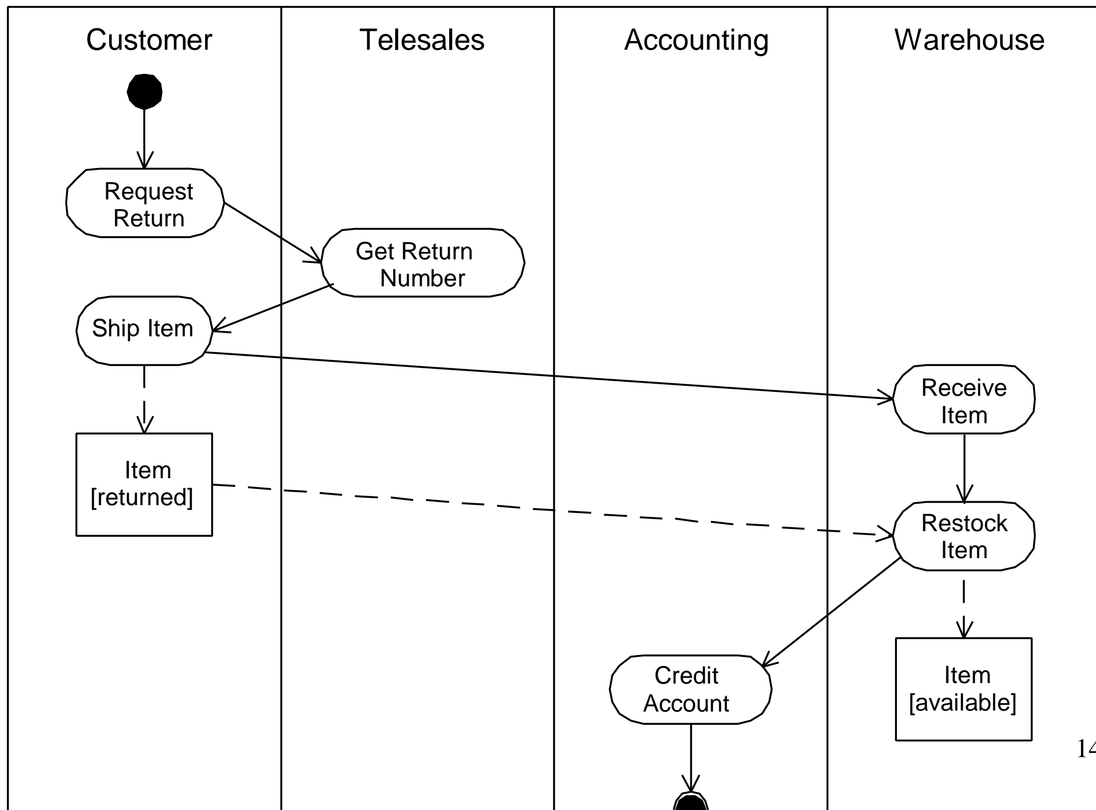


Figure 5: Activity Diagram currently disallowed

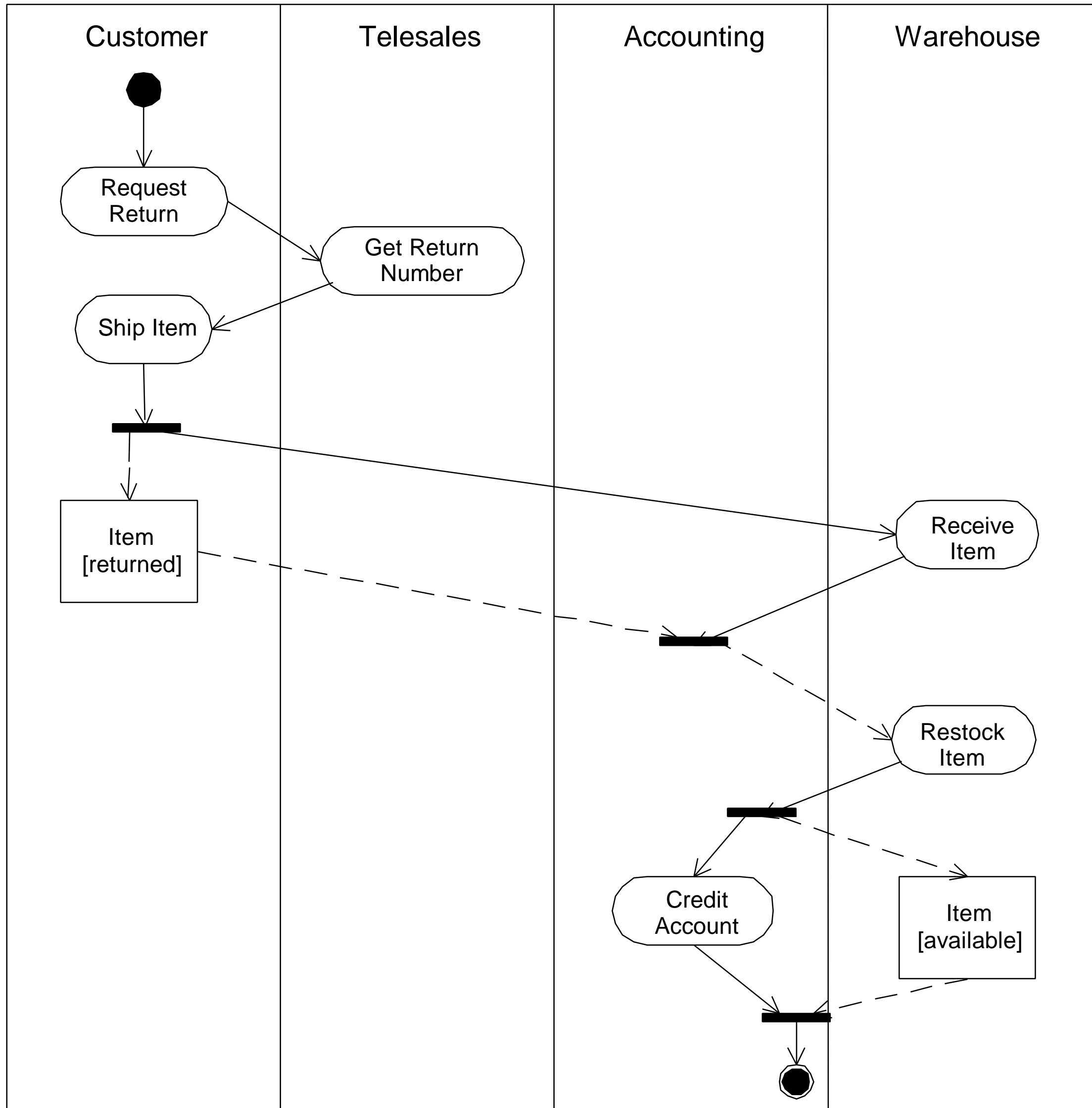


Figure 6: Activity Diagram required in UML currently

4.2.5 Branch to path that does not synchronize

It is a common case in both operating systems and business models to start a parallel process without concern for synchronizing it with the parent process. The WfMC call this a "chained process".

Suggested text for RFP:

The proposal should suggest changes in activity graphs that support modeling of parallel threads that do not join back to parent thread/state.

4.2.6 Events starting a process

It is common in business modeling to start a process based on an external event. For example, a support process is started when a customer calls. Since a new instance of the process is started each time a customer calls, perhaps when another support process is are still operative, this cannot be done with a single state machine. It requires two machines, one repeatedly invoking an asynchronous action that starts the other, as shown in Figure 7, using activity graph notation. The graph waits for the arrival of a customer call signal, and then uses an asynchronous CREATEACTION on a class with its own state machine. This way the state machine is started each time a customer calls the support center. An alternative is to use an asynchronous call action that invokes an operation realized by a state machine. In either case, a class is needed in addition to the two state machines.

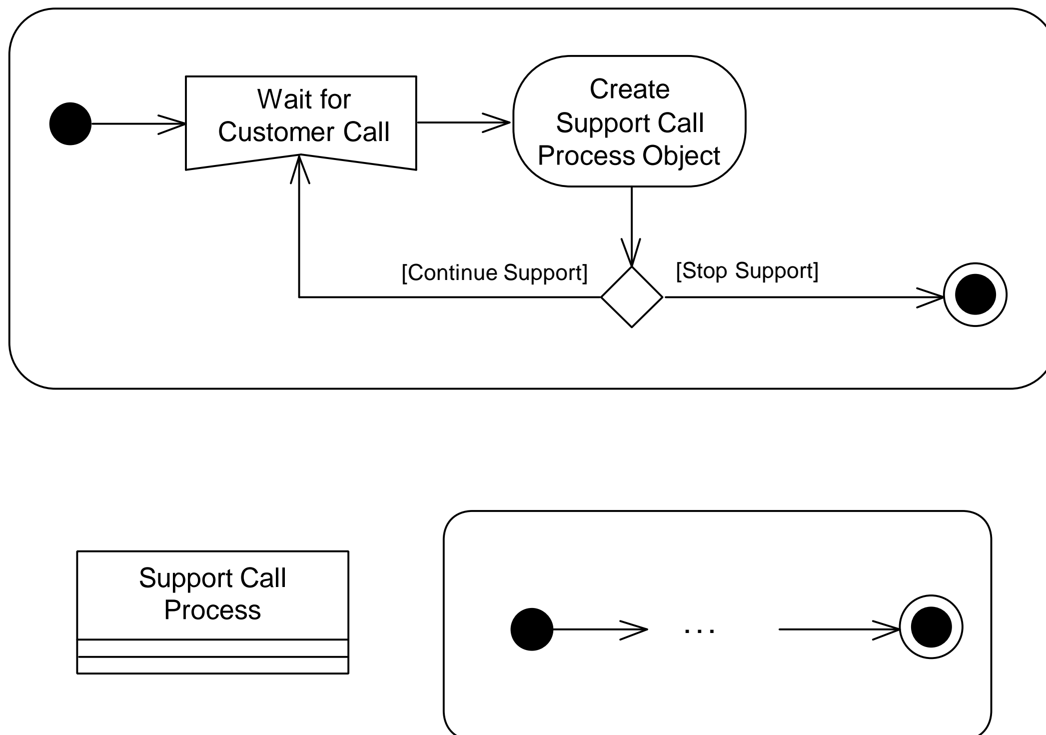


Figure 7: Events starting a process in UML

Business modelers do not expect and often do not need so much complexity to model an event-started process. Figure 8 shows a simpler diagram. The starting event is modeled at the beginning of the process instead of the initial state, notated as a filled triangle. This technique also requires a singleton class, so that the support center can be started and halted by creating and deleting the instance. Some applications may

need to limit the number of invocations of the process, for example, there are limited resources at the call center. The proposal should present a way to model this restriction using the simpler technique.

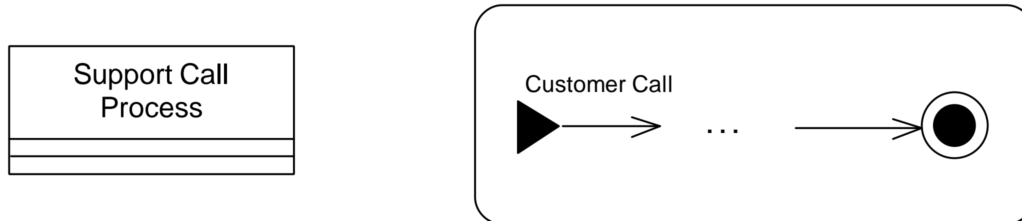


Figure 8: Simpler model for event-starting a process

Suggested text for RFP:

> The proposal should suggest changes to activity graphs that support simpler modeling of processes that are started by an event. The notation and model should be very simple, perhaps limited to a single new element and icon. The model should support parallel invocation of the process by multiple incoming events. It should be able to place restrictions on the number of parallel invocations. The model should support runtime control over when the process will respond to the event.

4.2.7 Starting thread without fork

It is very common in business modeling to wait for an event to happen before proceeding with a behavior. UML currently requires such wait states to have transitions into them, even when this is very cumbersome. Imagine the case shown in Figure 9 with a very large business process diagram. Requiring the visually long transition from the fork is hard to draw in a tool and clutters the diagram. Figure 10 shows an alternative. This was declined in UML 1.3 because it was not clear when the wait state should begin waiting. We think some simple conventions can be adopted to resolve this ambiguity. For example, in the case of Figure 10 it is obvious that the wait state begins when the fork is passed. This is such a common practice among business modelers that they should be given a chance to propose something that has an unambiguous semantics.

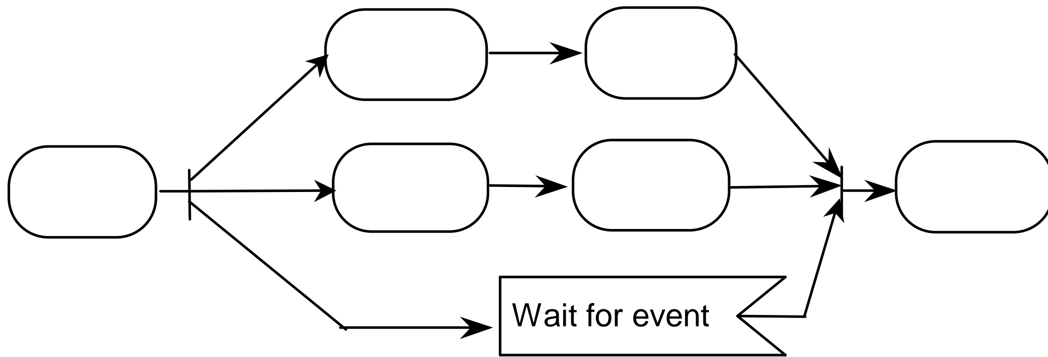


Figure 9: Thread that waits for event

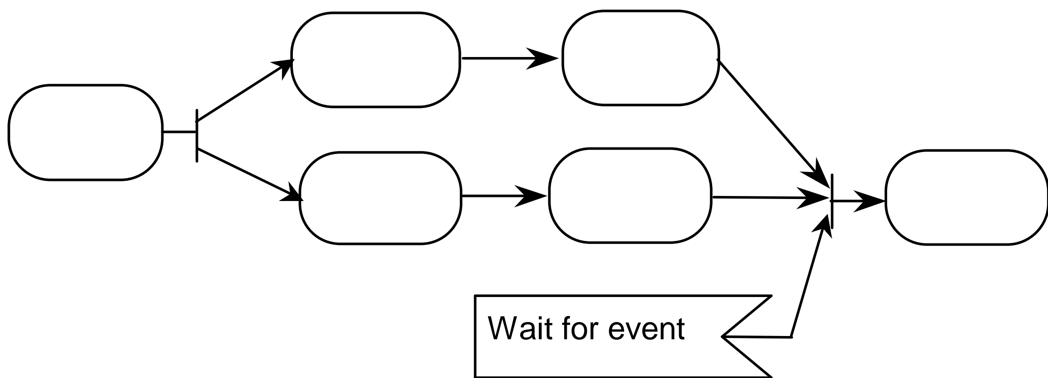


Figure 10: Wait state that starts a thread

Suggested text for RFP:

> The proposal should suggest changes to activity graphs that support modeling of wait states that do not require incoming transitions. Care should be taken that the model and notation express an unambiguous semantics.

4.2.8 Transformation of values from output parameters to input

It is a common workflow application to transform the output of one step in a behavior to the input of another. For example, suppose the call state ACCEPTORDER invokes an operation that returns an order, and it transitions to another call state ADVISECUSTOMER that takes a customer as input (see Figure 11 from SAP). It would be very useful to write an action or other specification that indicates the customer is calculated for ADVISECUSTOMER by retrieving the customer of the order output by ACCEPTORDER.

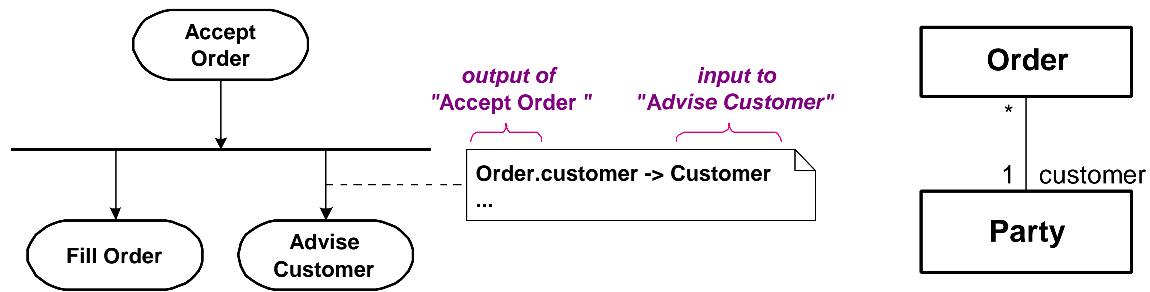


Figure 11: Transformation of outputs to inputs

Object flow states cannot serve this purpose, because they are constrained to not be transformed on the way from output to input (see their well-formedness rules and the usual usage). Putting the transformation to the action of the state is not where business expect it to happen, because the state is simply the invocation of the operation in their view. In addition, two transitions into a state should be able to specify different transformations. If completion events were explicit in UML (see section 4.3.2), they could at least carry the output values to be accessed by actions in the target state. However, it would be most transparent for the business user if the transformation could be part of the transition between the two states.

Suggested text for RFP:

> The proposal should suggest changes to activity graphs that support modeling of transformation of outputs of actions in one state to inputs of actions in states immediately downstream. The transformation can be specified in UML's action semantics or user's language. It would be most transparent for the business user if the transformation could be part of the transition between the two states, so that different transformations could target the same state.

4.2.9 Notation for object flow between input/output parameters

The current activity graph model supports the linking of object flow states to parameters that pass them as output and take them as input. However, there is not a corresponding notation.

Suggested text for RFP:

> The proposal should suggest notation for the meta-association between PARAMETER and OBJECTFLOWSTATE in activity graphs.

Can this be taken as a 1.4 item?

4.2.10 Concise notation for showing the type of object used in a call state

It is very common for readers of activity diagrams to look at a call state and want to see what type of object is having an operation invoked by the action of that state. There is currently no adopted notation for this. Notes are too bulky and non-standard for this application. Without this notation activity diagrams appear non-object-oriented.

Suggested text for RFP:

> The proposal should suggest notation for call states in activity graphs that shows the type of object that is having an operation invoked by the action of that state.

Can this be taken as a 1.4 item?

4.3 State Machine

4.3.1 Interruptible actions

It is a common case to interrupt a composite state or submachine state with an outgoing transition and trigger. However, neither of these works under an object-oriented usage, that is, when the behavior to be interrupted is a method on an object. In this case, an action state must be used to invoke an operation, and action states are not interruptible. It should be possible to interrupt a state-machine method for an operation invoked by an action in a state, by using an outgoing transition with trigger from the action state. It is not sufficient to have a state machine interrupt itself, because the interrupt conditions may be different for various states invoking the same operation.

Suggested text for RFP:

> The proposal should suggest changes to state machines that support modeling the interruption of actions in an state by triggered transactions outgoing from the state, especially in the case where the action invokes an operation implemented by state-machine methods.

Should this be an activity graph issue?

4.3.2 Explicit completion events

The runtime semantics for completion events is too specific. It forces completion events to be handled within the run-to-completion step in which they occur, rather than being queued for selection by the dispatcher:

> [p 2-147 Semantics] A completion transition is a transition without an explicit trigger, although it may have a guard defined. When all transition and entry

> actions and activities in the currently active state are completed, a completion event instance is generated. This event is the implicit trigger for a completion transition. The completion event is dispatched before any other queued events and has no associated parameters.

This means there can be long chains of activity through many states in a single run-to-completion step. It would be more flexible to queue completion events along with the others, to be selected and dispatched by the particular priorities of the implementation. This would allow interleaving of other activities into a long chain of completion transitions. It would also provide an explicit event to be used for dynamic concurrency arguments. Such flexibility would not be inconsistent with any other semantics of UML state machines, and would not prevent implementations from choosing the current semantics. In fact, it would be more consistent with modeling completion events as the real events they implicitly are anyway.

Suggested text for RFP:

> The proposal should suggest changes to state machines that support flexible dispatching of completion events, presumably by modeling them as real events that are put on the event queue.

4.3.3 Boolean event combination

It is a common application of state machines to have conjuncts or other Boolean combinations of events as transition triggers. Currently users must use dummy operations, or the signal receipt notation available in activity graphs (see Figure 12). In either case, cumbersome parallel threads are required, especially if the event combination is a complicated Boolean expression. The semantics document gives only a vaguely worded justification for this:

> Event conjunction is not supported, and the semantics is given in respect to a single event dispatch, to better match the type context as opposed to a general system context. [p 157]

Suggested text for RFP:

> The proposal should suggest changes to state machines that support arbitrary Boolean combination of events as triggers of transitions.

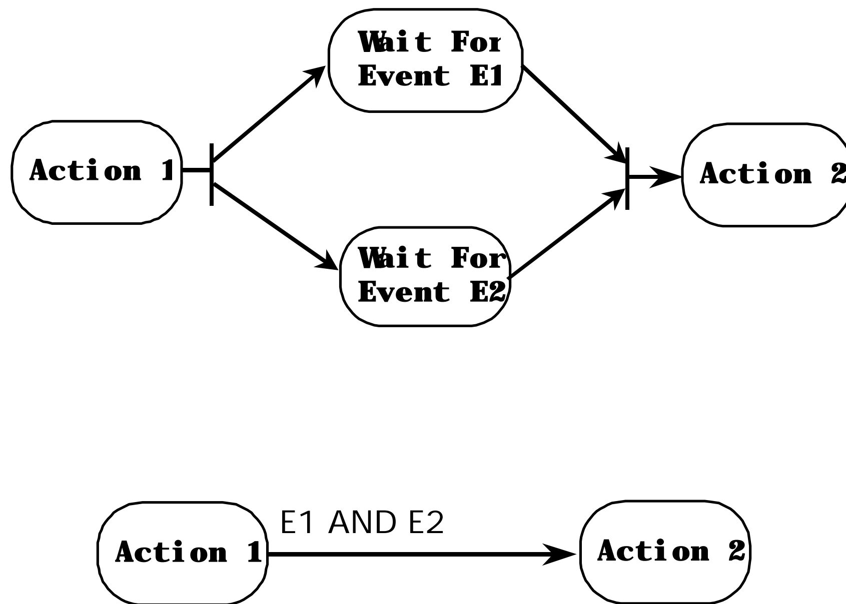


Figure 12: Event combination

4.3.4 Parallel event dispatch

The runtime semantics for event dispatch is too restrictive. It forces sequential processing of events even when parallel dispatch would be much more time-efficient (see Figure 13, from SAP). The simplification taken in UML 1.3 is fine for an early release, but UML 2.0 should improve this area for real-time modeling.

Suggested text for RFP:

> The proposal should suggest changes to state machines that support flexible dispatching of events in parallel to concurrent regions of a state machine. The proposal should address the question of parallel event consumption, which may require multiple event queues per state machine to allow one region to access events consumed by another.

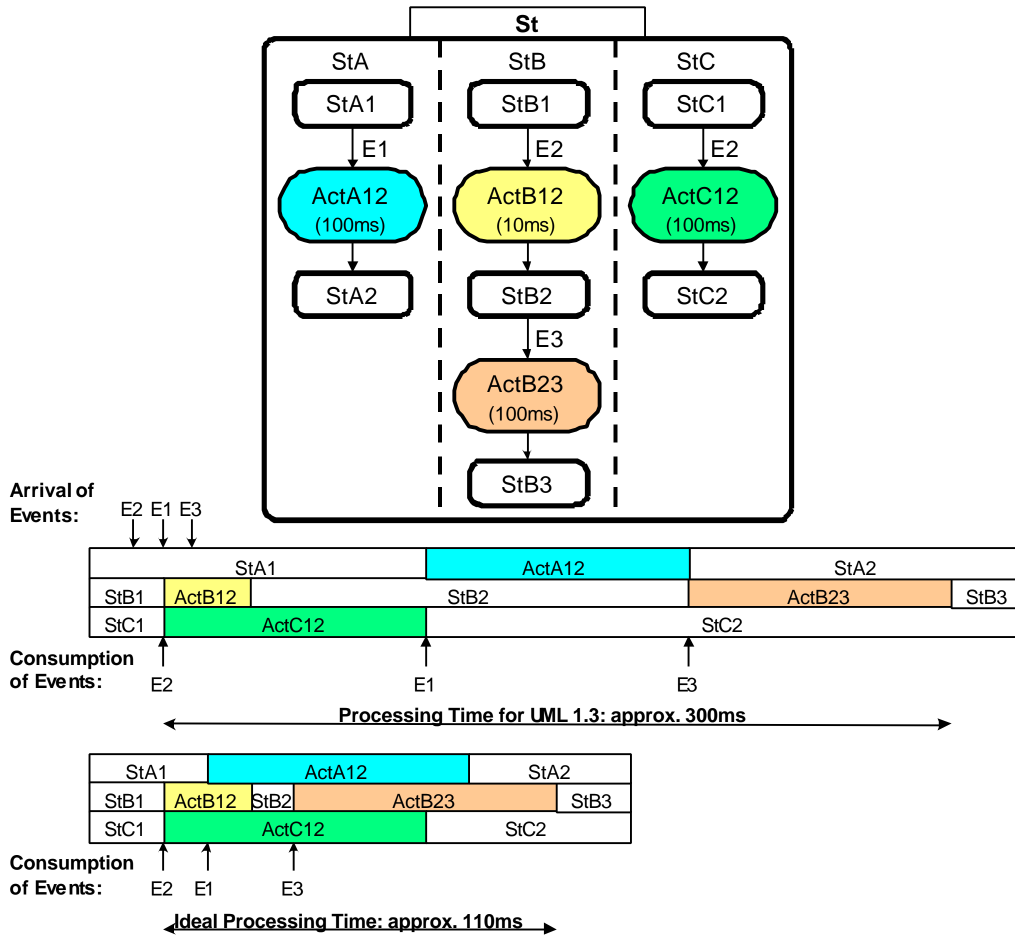


Figure 13: Sequential and parallel event dispatching

4.3.5 Models of execution-time behavior

It is general practice to create constructs that describe execution-time information about a behavior. For example, operating systems assign a data structure for each executing thread, that has attributes like how much CPU time it is taking, and operations like suspend and abort. The UML does not currently support these models fully, because neither operations nor state machine have an instance-level (execution) model. The collaboration model already has this capability.

Suggested text for RFP:

> The proposal should suggest changes to the UML behavior models, state machines and operations in particular, that support execution-time semantics. For example, it should be possible for the user of UML to define attributes for an executing operation or state machine, such as how long they have been running, and operations such as suspend and abort.

4.3.6 Parameterized state machines

Submachine states support partial reuse of state machines by allowing multiple submachine states to refer to the same state machine. However, there is no provision for parameterizing the submachine. The closest construct to parameters is stub states, which provide for transitions into named states, independently of the submachine being invoked. This does not provide the same capabilities as parameters, because parameters provide local data for the entire state machine to act on, whereas stubbed transitions provide data only for the targeted states.

Suggested text for RFP:

> The proposal should suggest changes to state machines that support parameterization, especially when a machine is invoked from a submachine state. These parameters should have the same semantics for the submachine as parameters on operations that have state machines as methods.

5. Appendix: UML 1.4 Issues

These unreported issues listed here in case any would be considered major revisions.

1. States currently do not model the conditions required for an object to be in a particular state. A constraint note can be linked to a state, but there is no specification of when the constraint should be tested. It could be tested when the object enters the state, leaves the state, or at any other time. Even if this were unambiguous, the consequence of violating the constraint is not defined, namely, to transition the machine to a state that has a constraint satisfied by the object. This might be modeled as a change-event trigger on an exiting transition, but it would be redundant with the constraint recorded on the state and with triggers on other transitions leaving the state, thereby impairing maintainability.

   Proposal: define a stereotype of constraint note for the above purpose, with unambiguous semantics.

2. Dynamics concurrency in activity graphs needs some way to access the arguments provided by the concurrency expression. The Reference manual suggests the "implicit" event, but does not define what that is (p 437). Perhaps it is an the action language issue. See section 4.3.2 above on completion events.

3. It is possible to model forks in sequence charts using multiple asynchronous messages. However, it is not possible to model joins, because return messages are considered activators, and multiple activators are not allowed.

   Proposal: allow multiple activators for messages.

4. Make guard evaluation procedure for choice points more explicit. It is not clear from the specification whether all guards are required to be evaluated, even after one is found to be true. This affects performance/real time issues even if the guards have no side-effects.

5. The <<primitive>> keyword/stereotype used in the meta-models of the datatype section are not defined. Isn't clear what level the datatype meta-model elements are at.

6. Description of context role, between state machine and model element, says:

   Each state machine is owned by exactly one model element.

   The meta-model shows 0..1.

7. Flow relationship has the wrong semantics specified for it:

   [p 2-33] It usually connects an activity to or from an object flow state, or two object flow states. It can also connect from a fork or to a branch.

Compare:

<<become>>

Specifies a Flow relationship, source and target of which represent the same instance at different points in time, but each with potentially different values, state instance, and roles. A Become Dependency from A to B means that instance A becomes B with possibly new values, state instance, and roles at a different moment in time/space.

<<copy>>

Specifies a Flow relationship, the source and target of which are different instances, but each with the same values, state instance, and roles (but a distinct identity). A Copy Dependency from A to B means that B is an exact copy of A. Future changes in A are not necessarily reflected in B.

8. Actions should have a ISPARALLEL attribute to specify if the iteration is sequential or parallel.

9. Using a role as the target of a create action does not support instantiation of children of the role classifier. [p 2-112 collaboration semantics].

10. What does it mean for RETURNACTION to be synchronous?

11. The Constraint meta-type, in the Extension Mechanisms meta-model, has two associations with the same association end name on the "opposite" ends ("constrainedElement"). Assuming that UML meta-classifiers should adhere to the OCL for regular classifiers, then this is ill-formed according to OCL 3 of Classifier, p 47:

    [3] No opposite AssociationEnds may have the same name within a Classifier.
    self.oppositeEnds->forAll (p, q | p.name = q.name implies p = q)

    The same may be true for the Collaboration meta-type (the "ownedElement" association end is duplicated), but these two are specializations of an association inherited from ModelElement, so perhaps that is acceptable.

12. See collaboration review for unfixed items (myreviewnote.txt).

13. CreateAction links to only one classifier. It should be multiple.

14. Multi-dimension partitioning (see issue filed by Oliver).

15. Action composition meta-modeled improperly: action sequence inherits from action. Should be Gamma's composition model with action as a sibling of action sequence.

16. Issue 74: branch needs constraints to partition.

17. ownerScope in Feature has the same semantics as targetScope in StructuralFeature. Aren't they clashing?

18. No mapping for this in mapping section, p 3-77:

    [p 3-75, Notation section for Composition] An association drawn entirely within a border of the composite is considered to be part of the composition.

19. What happens when a event is deferred in one region, but not another? Is it left on the queue accessible to both regions, even if it has already been consumed by one of the regions? Semantics says deferred events are kept if not used in one of the regions. So if one region uses it, it is lost, even if it is deferred in the other region. User cannot use event in both regions.

    Reference manual says:

    [p 443, Reference Manual] At the time that an object processes an event, it may be in one or more concurrent states. Each state receives a separate copy of the event and acts on it independently. Transitions in concurrent states fire independently. One substate can change without affecting the others, except on a fork or join caused by a complex transition (described later).

    and refers to an internal queue of events:

    [p 438] Deferred events. A list of events whose occurrence in the state is postponed until a state in which they are not deferred becomes active, at which time they occur and may trigger transitions in that state as if they had just occurred. The implementation of such deferred events would involve an internal queue of events.

20. Typo: The following should refer to "exit" not entry:

    [Semantics p 134] An optional action that is executed whenever this state is exited regardless of which transition was taken out of the state. If defined, entry actions are always executed to completion only after all internal activities and transition actions have completed execution.

6. References

[1] McCarthy, Brendan, "Association Inheritance and Composition", *Journal Of Object-oriented Programming,* 10: 4, July/August 1997, pp 69-81.

[2] Bock, Conrad, and James Odell, “A Foundation for Composition,” *Journal of Object-Oriented Programming*, 7:6, October 1994, pp. 10-14.

[3] Bock, Conrad, “Unified Behavior Models,” *Journal of Object-Oriented Programming*, 12:5, September 1999.

[4] Schlaer, Sally, and Stephan J. Mellor, *Object Lifecycles: Modeling the World in States*, Prentice Hall, 1992.

[5] Bock, Conrad, “Three Types of Behavior Model,” *Journal of Object-Oriented Programming*, 12:4, July/August 1999.

[6] Booch, Grady, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[7] Martin, James, and James J. Odell, *Object-Oriented Methods: A Foundation* (UML edition), Prentice Hall, Englewood Cliffs, NJ, 1998.

[8] Keller, Gerhard and Teufel, Thomas, *SAP R/3 Process Oriented Implementation: Iterative Process Prototyping*, Addison-Wesley, 1998.SAP