UML 2 Activity and Action Models

Conrad Bock, U.S. National Institute of Standards and Technology

This is the first in a series introducing the activity model in the Unified Modeling Language, version 2 (UML 2), and how it integrates with the action model [1]. The series is a companion to the standard, providing additional background, rationale, examples, and introducing concepts in a logical order. This article covers motivation and architecture for the new models, basic aspects of UML 2 activities and actions, and introduces the general notion of behavior in UML 2.

1   BACKGROUND

A focus of recent developments in UML has been on procedures and processes. UML 1.5 introduced a model for parameterized procedures defined by control and data flow to complement the existing state and interaction models [2]. For the first time UML supports first-class procedures that can be used as methods on objects or independently of objects, as occurs when modeling, for example, function libraries with popular programming languages. It also facilitates application of UML by modelers who do not use object-orientation (OO) routinely, such as system engineers and enterprise modelers, and provides them a path to incrementally adopt OO as needed [3]. The flexibility to combine OO with functional approaches considerably widens and integrates the potential applications of UML.

UML 1.1	UML 1.3	UML 1.5	UML 2.0

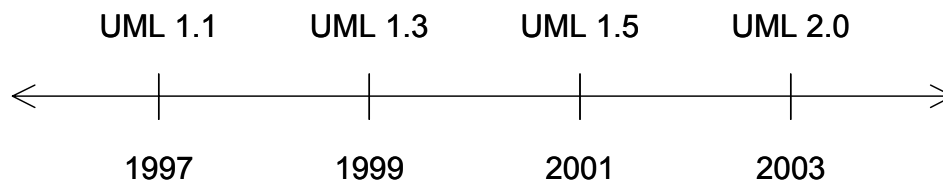| 1997	1999	2001	2003

Figure 1: UML Timeline

UML 1.5 also inherited from earlier versions a form of state machine that was notationally modified to appear as a flow diagram, called the activity diagram. Unfortunately, the underlying state machine semantics restricts expressiveness and is

confusing to users. Especially those not using an OO approach perceive UML 1.x activity models as not working for them. There is also concern that UML 1.5 has multiple models for control and data flow.

In light of this experience, UML 2.0 redefined the activity model to give it a basis in flow modeling intuition, and to integrate it with the UML 1.5 action model. The new activity model supports the control and data flow of UML 1.5 procedures, as well as more general features, such as cycles and queuing. Consequently, UML 2 activities support flow modeling across a wide variety of domains, from computational to physical. This makes it ideal for specifying systems independently of whether the implementation is software or hardware, as in system engineering, and independently of where the system/environment boundary is drawn. Finally, the combination of activities and actions retains the UML 1.x capability of reacting to events, so can be applied to areas requiring that, such as embedded and agent-based systems.

2   MULTIPLE DOMAINS

Although the UML 2 activity and action models are defined independently of application, some features are more appropriate to some domain styles than others. For example, actions for throwing exceptions will probably be used more often by programmers, whereas enterprise modelers are more likely to apply other techniques for atypical flows, such as exception parameters and interrupting regions. This redundancy is unavoidable when creating an abstraction over user groups that do not overlap. However, it is more than made up for by the efficiency in communication based on a common model between domains that are integrated in delivered systems.

To support vendors focusing on particular users, the activity model is packaged in a more fine-grained way than other behavior models, and has more complex dependencies between packages, as shown in Figure 2.
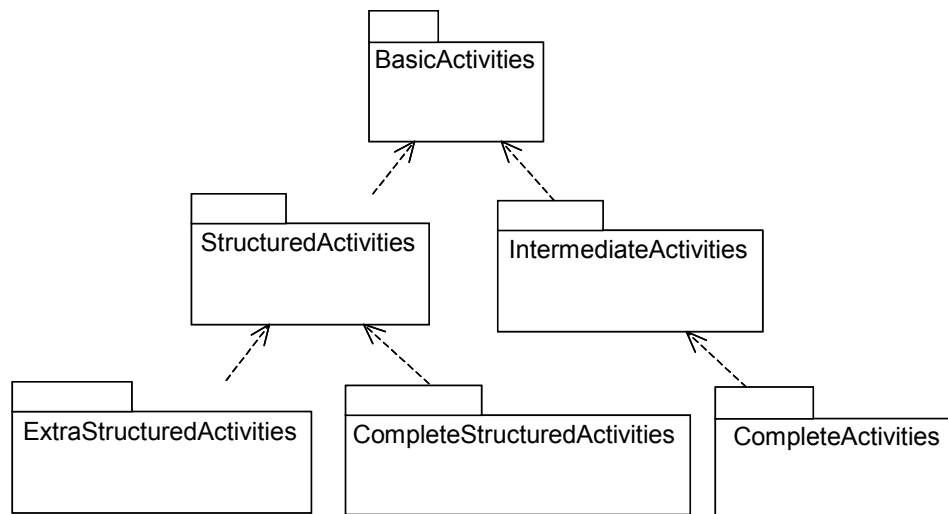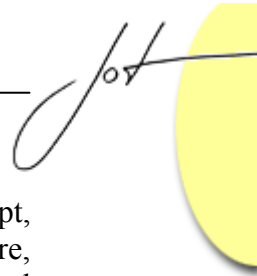

Figure 2: Activity Package Dependencies

The features in common between the applications, such as the abstract action concept, control/data flow, and input/output, are at the root (basic activities). From there, dependencies branches in two directions, one mainly for software modeling (structured activities), and the other for general process modeling (intermediate and complete activities). Structured activities introduce well-nested constructs, for modeling conditionals, variables, try/catch, and so on. Intermediate and complete activities introduce explicit parallelism, partitions, flow-based forms of exception handling, and a number of constructs for finer-grained control of flows. These are orthogonal branches, so they may be combined. For example, structured activities may be folded in with intermediate activities, to support explicit parallelism and structured conditionals at the same time.[1] This series will cover the functionality of the various packages.

Supporting a wide range of applications also requires multiple notations. For example, programmers tend to favor textual notations, while subject matter experts prefer graphical notations. UML addresses this by defining a repository for storing specifications that can be populated from multiple notations. Programmers can use a textual syntax to build and read activity models in the repository, and enterprise modelers can use a graphical notation. The UML repository is a communication medium between multiple notations, and a source for highly directable compilers targeting multiple platforms [4].

3   FLOW MODELS AND SEMANTICS

Behavior models in general determine when other behaviors should start and what their inputs are [3]. In particular, the UML 2 activity models follow traditional control and data flow approaches by initiating subbehaviors according to when others finish and when inputs are available. It is typical for users of control and data flow to visualize runtime effect by following lines in a diagram from earlier to later end points, and to imagine control and data moving along the lines. Consequently a token flow semantics inspired by Petri nets is most intuitive for these users, where "token" is just a general term for control and data values.

UML 2 takes the same approach to behavioral semantics as UML 1.x, namely to define intuitive virtual machines. This enables users and vendors to predict the runtime effect of their models. UML 2 activities define a virtual machine based on routing of control and data through a graph of nodes connected by edges. Each node and edge defines when control and data values move through it. These token movement rules can be combined to predict the behavior of the entire graph.[2] The rules for control and data movement are only intended to predict runtime effect, that is, when behaviors will start

---

[1] The action model should be finely packaged to correspond with activities, but is currently divided into INTERMEDIATEACTIONS and COMPLETEACTIONS only. These contain the predefined actions, such as object creation, setting attributes, and so on. The abstract notion of action is defined in the BASICACTIVITIES package because it is integral to the flow model. See later sections of this article.
[2] It is hoped that the rules are precise enough to be translated to a formal semantics, especially to support proving properties about modeled processes. This is left for future work.

and with what inputs. They do not constrain implementation further than that. In particular, the rules do not imply that activity models must be implemented as a virtual machine corresponding to the token analogy, messaging passing, or any other scheme. It is only necessary that the runtime effects predicted by the virtual machine actually occur in the implementation.

The activity model is defined with few semantic variations, which is the UML term for allowing implementations to choose alternative runtime behavior without recording those choices in the model. Semantic variations cause a model in one implementation to execute differently than the same model in another implementation, with no standard way to tell what the differences are. Variations in activity execution are mostly modeled as attribute values in the user's model. This means they are under user control and are transmitted to other users without requiring implicit alignment of implementations.

4   ACTIVITY NODES AND EDGES

UML 2 activities contain nodes connected by edges to form a complete flow graph. Control and data values flow along the edges and are operated on by the nodes, routed to other nodes, or stored temporarily. More specifically, there are three kinds of node in activity models:

1. Action nodes operate on control and data values that they receive, and provide control and data to other actions.
2. Control nodes route control and data tokens through the graph. These include constructs for choosing between alternative flows (decision points), for proceeding along multiple flows in parallel (forks), and so on.
3. Object nodes hold data tokens temporarily as they wait to move through the graph.

Figure 3 shows the notation for some of the activity nodes to be discussed. Contrary to the names, control nodes coordinate both data flow and control flow in the graph, and object nodes can hold both objects and data.[3]

Activity nodes are connected by two kinds of directed edges:

1. Control flow edges connect actions to indicate that the action at the target end of the edge (the arrowhead) cannot start until the source action finishes. Only control tokens can pass along control flow edges.
2. Object flow edges connect objects nodes to provide inputs to actions. Only objects and data tokens can pass along object flow edges.

---

[3] UML abstracts from objects and data to classifiers in general.

Action Node:

label

Control Nodes:

Decision
and
Merge

Fork
and
Join

Initial
node

Final
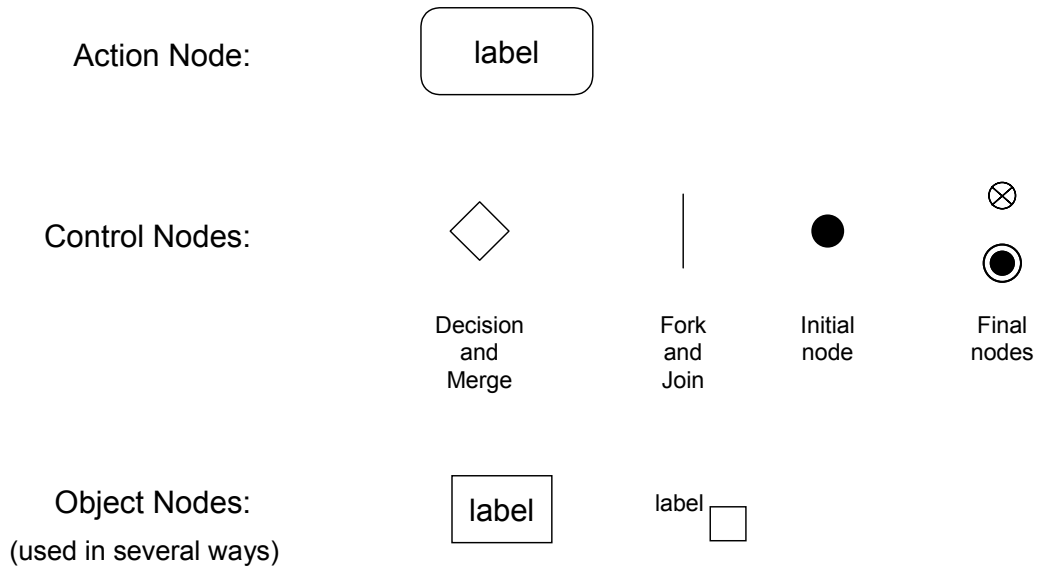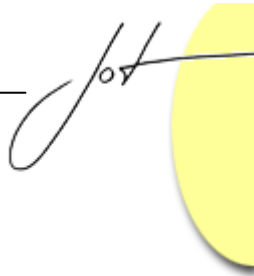nodes

Object Nodes:
(used in several ways)

label

label

Figure 3: Activity Nodes

Figure 4 shows the notation for edges. Control and object flow edges are distinguished by usage. Control edges connect actions directly, whereas object flow edges connect the inputs and outputs of actions (see next section).
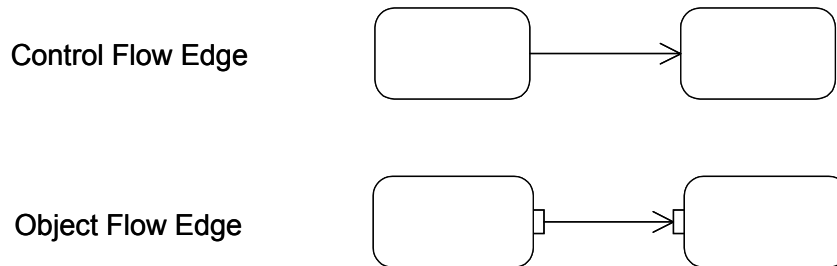
Control Flow Edge

Object Flow Edge

Figure 4: Activity Edges

This rest of this article introduces actions and edges with a small example. Later articles will cover the other kinds of nodes, and more features of actions and edges.

5   ACTIONS

Activity models coordinate actions, some of which may invoke user-defined behaviors, including other activities. All actions are predefined. For example, UML 2 has actions to create objects, set attributes values, link objects together, and to invoke user-defined behaviors. Actions can have inputs and outputs, which are called pins, that are connected

by object flow edges to show how values flow through the activity, provided by some actions and received by others. In the simple cases, all inputs to an action are required to be available for it to begin executing.

Figure 5 shows an example of an action creating a new instance of an ORDER class, then another action invoking a user-defined behavior to fill it. The object creation action creates a blank order that FILLORDER completes. Action nodes are notated with round cornered rectangles.[4, 5] At the top of Figure 5, the small rectangles attached to the actions are input and output pins. The type of object accepted as input or provided as output is shown as an adornment. The notation at the bottom of Figure 5 can be used when the types of input and output are the same. Pins are a kind of object node, so they also hold data values temporarily in the flow.
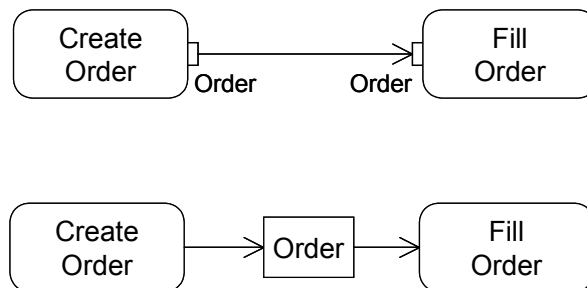

Figure 5: Example Actions and Object Flow Edges

As mentioned before, it is not necessary to use the notation of Figure 5. Programmers will mostly likely prefer a textual notation [4], such as:

```
Order o;
o = new Order;
FillOrder(o);
```

The repository model defined by UML 2 is the same for the above notations, as shown in Figure 6, assuming the variables of the textual language are modeled as dataflow.[6] Each element of the repository is an instance of a metaclass defined in the UML specification, which is the name to the right of the colon. The name of the repository instance itself is to the left of the colon, blank if it is anonymous. The model shows a CREATEOBJECTACTION with an output pin passing its value to the input pin of a CALLBEHAVIORACTION.. The

---

[4] The wording inside the action nodes is not normative, because a standard textual notation for actions is not adopted yet. Also action nodes can be given labels that are more descriptive than the predefined action name.

[5] The round cornered notation is also used by state machines to notate states, unfortunately. This is not a desirable situation. However, it is beneficial to those users and vendors who have been trying to apply state machines to flow modeling applications, for lack of a flow modeling standard until now.

[6] UML 1.5 and UML 2 also support variables if needed.
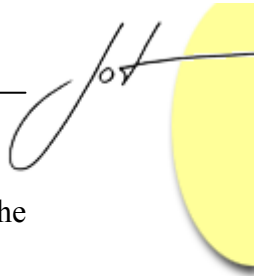
CREATEOBJECTACTION is linked to the user class it instantiates (ORDER), and the CALLBEHAVIORACTION is linked to the user-defined behavior it invokes (FILLORDER).
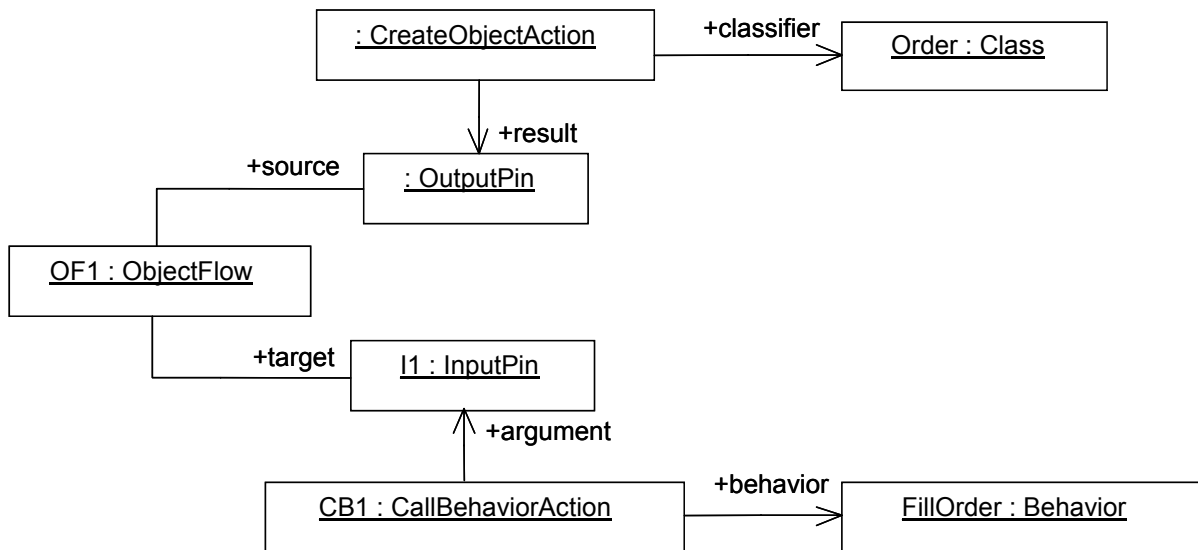

Figure 6: Repository model for Figure 5

The primitive actions for creating objects, invoking user-defined behaviors, and so on, are not technically behaviors themselves, but this is more an artifact of metamodeling than a conceptual distinction. The fundamental distinction is between a specification of dynamic effect and its usage, to support multiple usages of the same specification. For example, the same FILLORDER behavior may be invoked in many activity diagrams, or many times in the same activity diagram, but each invocation will be represented by a separate instance of CALLBEHAVIORACTION in the repository, all referring to the same FILLORDER behavior. This is because each usage of FILLORDER will be in a different flow, or at different points in the same flow, and each usage may have different actions before and after it. For example, Figure 6 has a CREATEOBJECTACTION before FILLORDER, whereas another flow might not.[7]

Actions are also reusable, but this happens to be modeled in a different way than user-defined behaviors. Each action in a flow is a new instance of a single class from the UML metamodel. For example, if an activity contains two object creation action nodes, then the user's repository has two instances of the CREATEOBJECTACTION class from the UML metamodel, and separate sets of pins for each. Reusability is achieved by using multiple

---

[7] Programming languages make the same distinction between a procedure declaration or definition, which has the signature, and statements that call the procedure, providing actual parameters at runtime. In UML 2 terms, a procedure definition is a behavior, and a statement is an action.
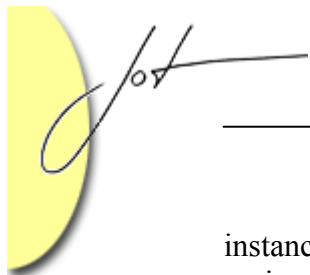
instances of the same UML metaclass as action usages[8]. Later articles will cover the various kinds of actions.

6   ACTIVITIES

Activities are user-defined behaviors, and like all behaviors in UML 2 can be initiated by invocation actions, and support parameters to receive and provide data to the invoker. Parameters are part of the reusable definition of an activity. Parameters are not pins, because pins are used to connect actions in a flow, whereas activities are behaviors invoked by actions (see previous section). However, to access parameter values from actions in the activity, activity parameters are modeled as a special kind of object node used to temporarily hold actual parameter values as they flow into and out of the activity. Figure 7 shows a parameterized activity with a parameter object node connected to pins on actions. Parameter object nodes are shown on the border, with object flow edges connecting them to pins. The type of object held in an object node is usually shown in the label. In this example, the information used to fill the order is provided as an input parameter and passed along to the invocation of the FILLORDER behavior.
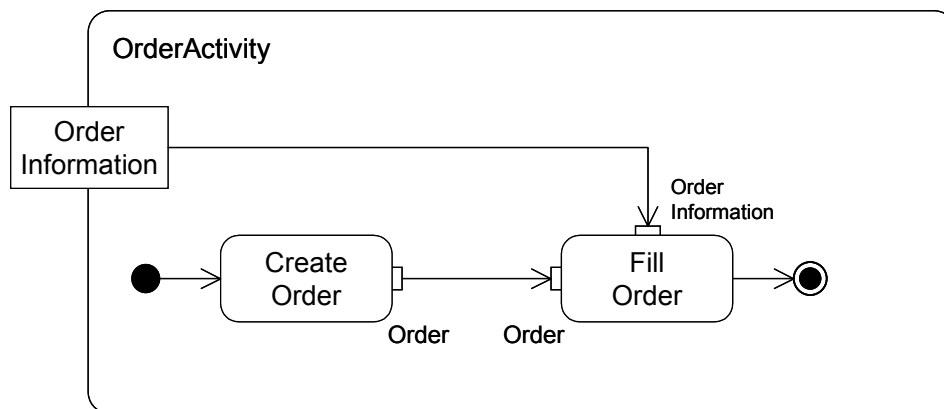

Figure 7: Example Activity

The control nodes at the beginning and end of the flow in Figure 7 are initial and final nodes respectively. When the ORDERACTIVITY is invoked, a control token is placed at the initial node, and a data token with ordering information is placed at the input parameter object node. The control token flows from the initial node to the CREATEORDER action, which begins executing. The data token flows from the parameter to the invocation action

---

[8] It is possible that actions could be modeled in the same way as user-defined behaviors and be standardized as a reusable model library. Then only one predefined action would be needed to invoke all behaviors, whether predefined or user-defined. However, the static requirements of predefined actions, such as CREATEOBJECTACTION requiring a class to instantiate, would need to be recorded in constraints rather than associations in the metamodel. Constraints are generally not as easy to read in the UML specification as metamodel associations.
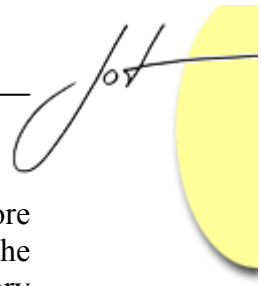
for FILLORDER, which must wait for CREATEORDER to provide its other input before starting. When FILLORDER is done, a control token is passed to the final node and the activity terminates, returning control to execution that started it. A partial repository model for Figure 7 is in Figure 8 below. It connects to the repository model of Figure 6 through the CB1 CALLBEHAVIORACTION. The CREATEORDER action and its flows are omitted for brevity.
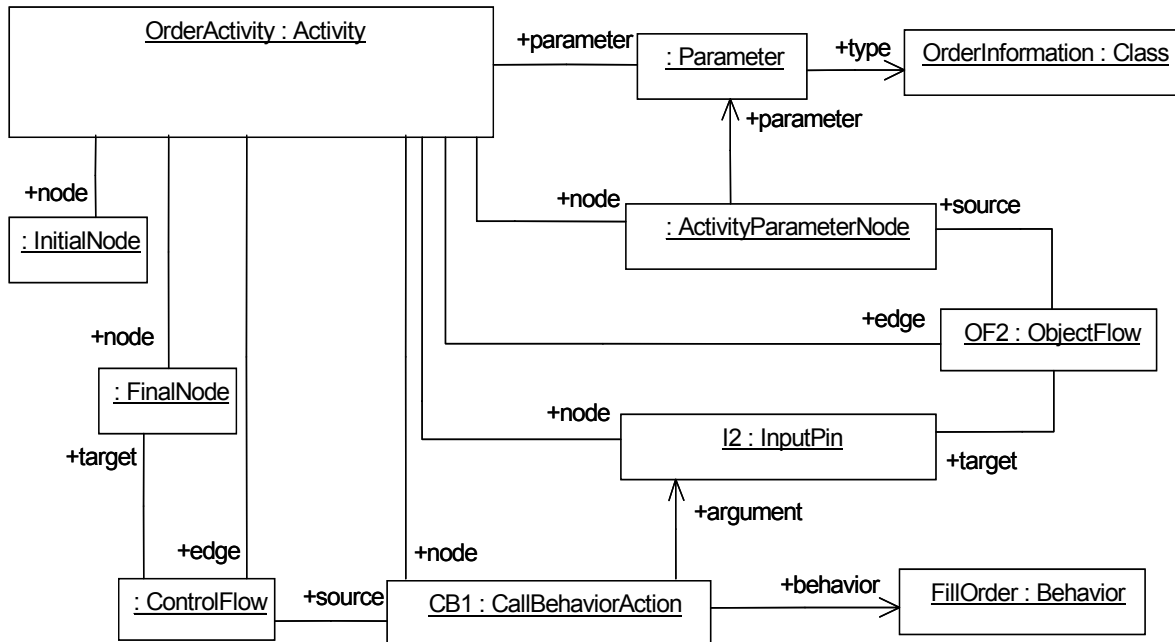

Figure 8: Partial repository model for Figure 7

7 BEHAVIOR IN UML 2

UML 2 supports the concept of parameterized behavior for all the kinds of behavior in UML, not just activities. This means state machines, interactions, and activities all can be parameterized, and be methods on objects or invoked directly, in a uniform way. The upper right of Figure 9 shows an activity model for a behavior called DELIVERMAIL (the curved arrows are not part of UML notation). DELIVERMAIL could be invoked as is with a CALLBEHAVIORACTION, or as a method on the POEMPLOYEE class with a CALLOPERATIONACTION. In either case, the behavior takes an instance of KEY as input. Because behaviors can be invoked directly or as operations, UML 2 provides a path to incrementally adopt object-orientation [3].
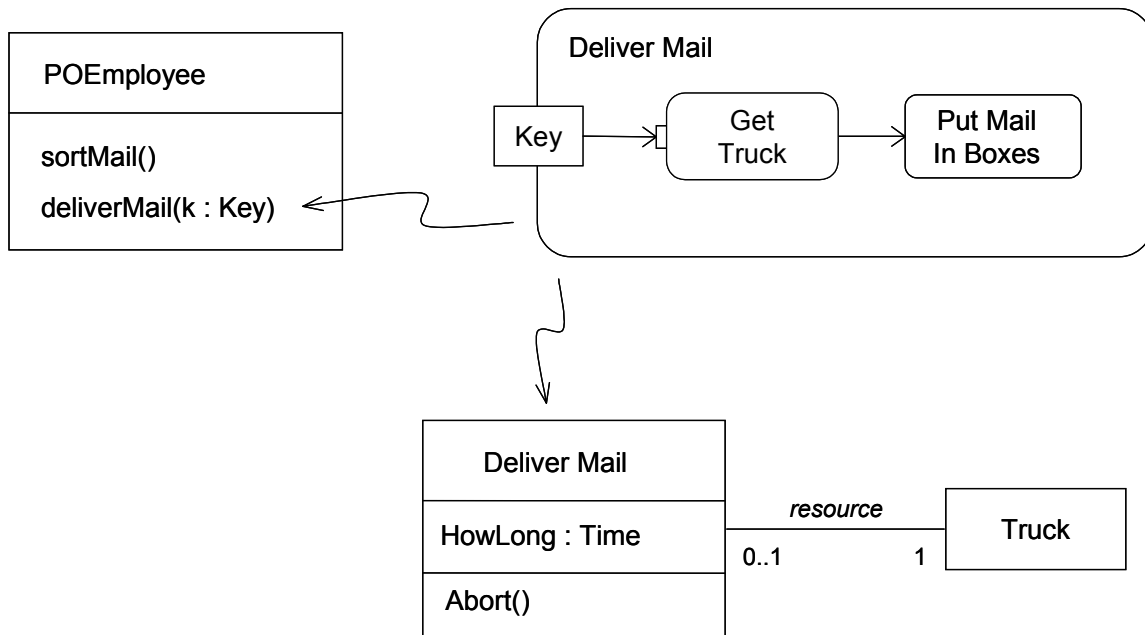
Figure 9: UML 2 Behavior

UML 2 user-defined behaviors are also classes. Each time a behavior is executed at runtime, a new runtime instance of the user's behavior class is created. The instance is destroyed when the behavior terminates. Behavior classes, like all classes, can support attributes, associations, operations, and even other behaviors, such as state machines. This reflects common practice in systems that manage processes, for example, workflow and operating systems. The bottom of Figure 8 shows the behavior class for the DELIVER MAIL activity with an attribute for how long each execution of DELIVER MAIL has been running, an operation to abort the execution, and an association for the truck it is using. Applications can also put state machines on the behavior class to describe the status of each execution, such as NOT_STARTED, SUSPENDED, and so on [5] [6].

UML 2 behavior classes enable the definition of standard functionality for process management, even though UML 2 does not define standard features itself. Behavior class features can be defined by domain standards, vendors, or user groups as reusable model libraries containing abstract behavior classes with normative attributes and operations such as ABORT and so on. Then these classes can be used as supertypes of user-defined behaviors such as DELIVER MAIL in Figure 9.

8 CONCLUSION

This article begins a series on the UML 2 activity and action models. It reviews progress in UML flow modeling, package structure needed to serve the wide range of flow
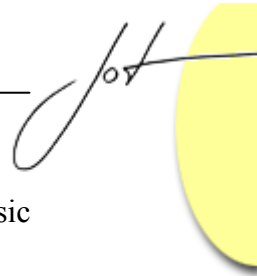
modeling applications, UML's approach to semantics, then introduces some basic elements of activity modeling, and the UML 2 behavior model generally.

ACKNOWLEDGEMENTS

REFERENCES

[1]   U2 Partners, "Unified Modeling Language: Superstructure", version 2.0, 3rd revised submission to OMG RFP ad/00-09-02, http://www.omg.org/cgi-bin/doc?ad/2003-04-01, April 2003.

[2]   Object Management Group, "OMG Unified Modeling Language", version 1.5, http://www.omg.org/cgi-bin/doc?formal/03-03-01, March 2003.

[3]   Bock, Conrad, "Three Kinds of Behavior Model," *Journal of Object-Oriented Programming*, 12:4, July/August 1999.

[4]   Bock, Conrad, "UML Without Pictures", to appear in *IEEE Computer Special Issue on Model-driven Development*, September/October 2003.

[5]   Object Management Group, "Workflow Management Facility Specification", version 1.2, http://www.omg.org/cgi-bin/doc?formal/00-05-02, May 2000.

[6]   Workflow Management Coalition, "Workflow Standard - Interoperability Abstract Specification", http://www.wfmc.org/standards/docs/TC-1012_Nov_99.pdf, November 1999.

About the author

Conrad Bock is a Computer Scientist at the National Institute of Standards and Technology. He is the workgroup lead for activities and actions in the UML 2 submission team, and can be reached at conrad.bock@nist.gov .