JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 7, July-August 2004

UML 2 Activity and Action Models

Part 5: Partitions

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

This is the fifth 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 first article gives an overview of activities and actions [2], while the next three cover actions generally, control nodes, and object nodes. This one describes partitions, which are a way of grouping actions that have some characteristic in common. In particular, they can relate actions to classes that are responsible for them, and highlight the abstraction that activities provide for interaction diagrams and state machines.

1 PARTITIONS

Partitions are groups of actions that highlight information already in an activity, or that will be, and present it in a more compact way. Partitions do not have execution semantics themselves, but because they are redundant with the information in the executable part of the model, tools can automatically update the executable model when the user modifies partitions or their contents. To reduce clutter, tools can also omit the redundant portions of the execution from the diagram, while still keeping them in the model repository for system generation. Figure 1 shows the example used in this article, adapted from [1][3].[1] Each of the areas between the parallel, vertical lines is a partition, and this particular way of notating them is called a *swimlane*. An alternate notation is shown in Figure 2, where partitions are labeled on nodes.

[1] Forks and join are shown for clarity, as suggested by [3], but are not necessary in this example, due to similar semantics for actions [4].

Cite this column as follows: Conrad Bock: "UML 2 Activity and Action Models Part 5: Partitions", in *Journal of Object Technology*, vol. 3, no. 7, pp. 37-56.
http://www.jot.fm/issues/issue_2004_07/column4

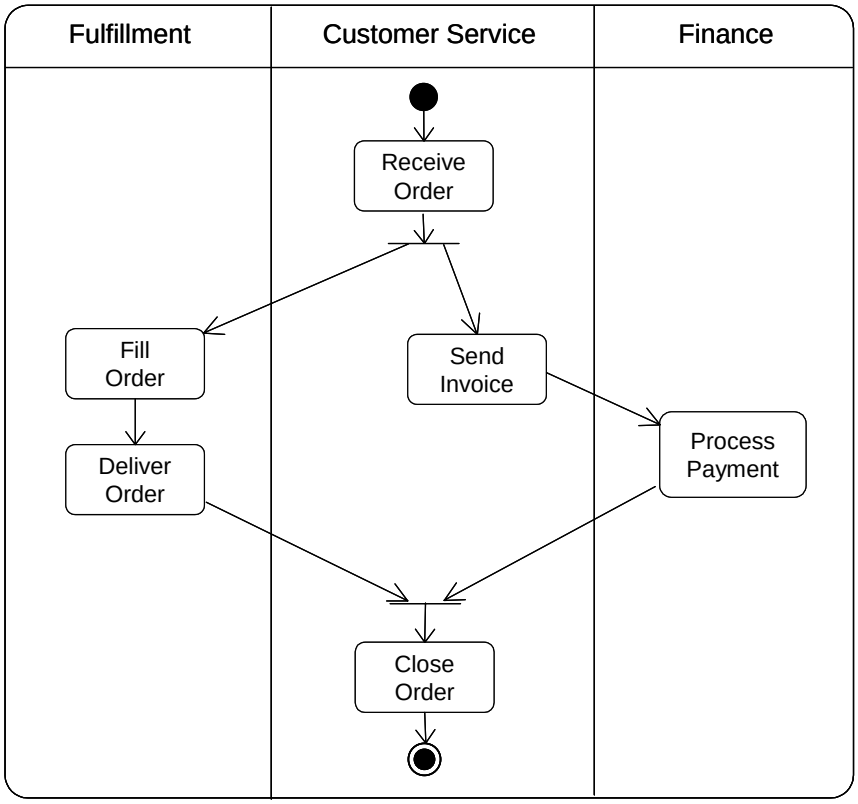

Figure 1: Partition Example, Swimlane Notation

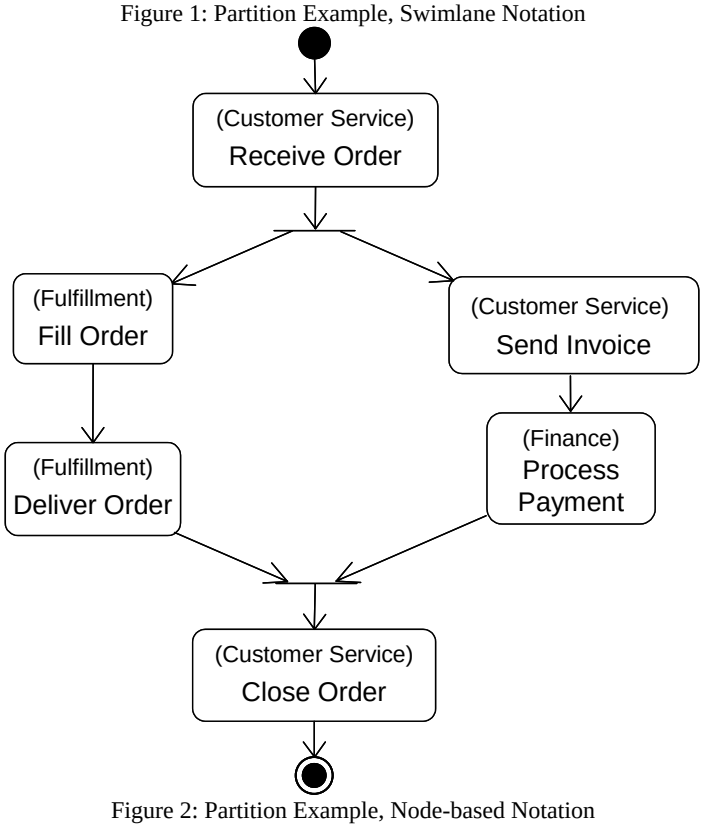

Figure 2: Partition Example, Node-based Notation

Because there is so much information in an activity, and so many ways to highlight it and make it more compact, partitions can be extended by modelers and tools to support applications not explicitly defined in UML. This article describes the ways of using partitions predefined in UML and gives an example of a modeler-defined application. It also discusses the translation of activities to interaction diagrams and state machines in sections 3 and 6.

2 CLASS PARTITIONS

Partitions are often used to indicate what or who is responsible for actions grouped by the partition. The term "responsible" has a wide variety of meanings, but the one defined by UML is that a class supports the behavior invoked by actions in the partition. For CALLOPERATIONACTION, this means the class defines the invoked operation [4]. For CALLBEHAVIORACTION, it means that the class owns the behavior.[2] For example, Figure 3 shows partitions representing classes, as would be appropriate to model generic systems that operate in whatever company installs them.[3] If the action FILL ORDER is to invoke an operation, then the operation must be declared on the FULFILLMENT class. This is shown on action PROCESS PAYMENT with the notation for CALLOPERATIONACTION that indicates the target class of the invocation [4]. When all the CALLOPERATIONACTIONs conform to their containing partitions, the partitions only highlight information already in the activity. The companion class model is shown in Figure 4.[4] Each company will have instances of these classes, which are targets of the operation calls. See second half of this section, and Figure 7.

[2] Same applies to behaviors on nodes other than actions, for example, decision input behaviors on decision nodes [5].

[3] The guillemet notation is used for metaclasses as well as stereotypes, which are both metalevel concepts. Keywords refer specifically to an aspect of the UML metamodel, such as metaclasses or metaproperties.

[4] An alternative approach use partitions representing order and invoice classes, with operations on them instead. This conforms to the conventional object-oriented development style of translating objective nouns to classes and verbs to operations on them. Figure 3 is more typical of web services or agent techniques, which identify active entities that operate on passive ones. These approaches highlight the responsible parties, but have the disadvantage of being more brittle under organizational or industrial change [6]. The alternatives could be harmonized by class partitions representing the sender of the message/operation, combined with other dimensions representing the targets (dimensions are covered in section 5). This will be addressed in revision.

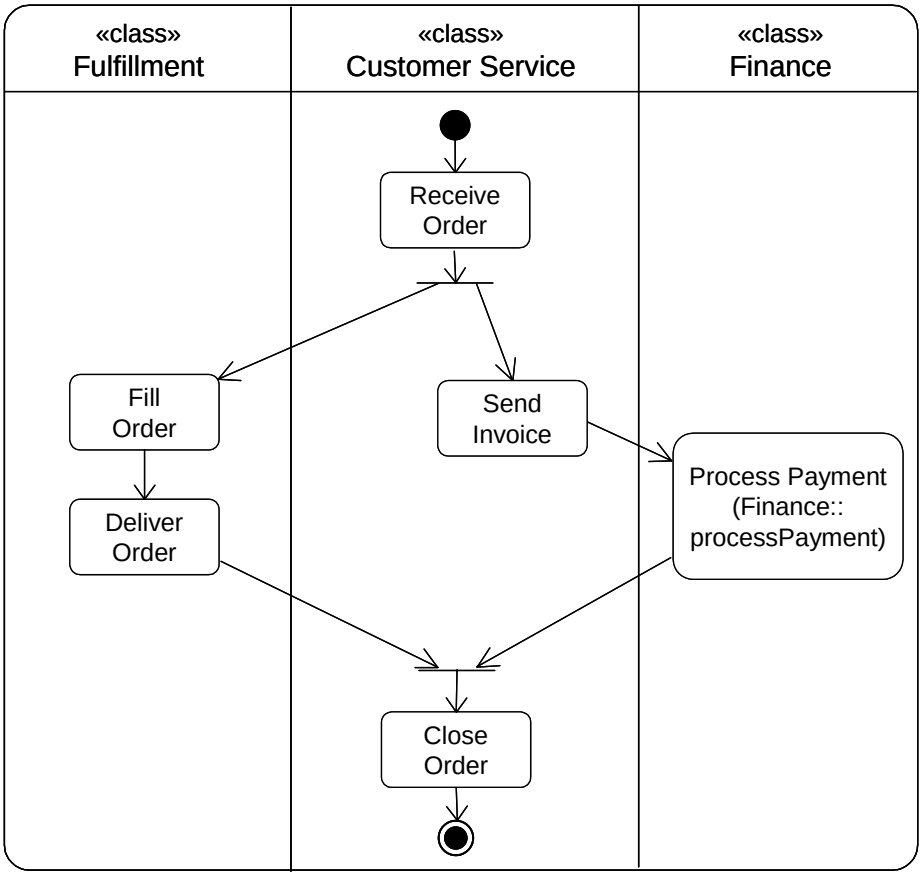

Figure 3: Partitions Representing Classes

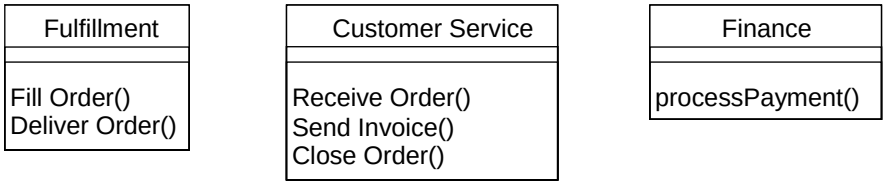

Figure 4: Class Model for Figure 3

Tools can add value to class partitions by automatically maintaining consistency between partitions, actions, and classes as the diagrams are modified. For example, when a CALLOPERATIONACTION action is moved into the FULFILLMENT partition, the tool can automatically move the invoked operation to the corresponding class, from whatever class it was on previously. For applications that are strictly object-oriented, if a class partition is added, the CALLBEHAVIORACTIONS in it can be converted to CALLOPERATIONACTIONS, and operations automatically defined on the class with the

behaviors used as a methods. Tools can also add value by making the diagrams more compact while keeping the full specification in the underlying model repository. For example, the diagram can omit the full CALLOPERATIONACTION notation, which is redundant with the partitions, while leaving the action completely specified in the underlying model.[5]

Once partitions, actions, and classes are consistent, there is the question of which particular instances are targets for the CALLOPERATIONACTIONs. There are a number of ways to show this. One is to add object flows that provide the instance as input to CALLOPERATIONACTION, as shown in Figure 5. This has the obvious disadvantage of clutter, but is explicit about the necessary inputs to the actions. It also makes clear that the customer service department sending the invoice should be the same one that closes the order. An alternative is to use partitions that represent instances directly, but this is only useful for individual scenarios, not for specifying a behavior that must operate on many instances. And it would still require object flows from value pins to pass instances to the CALLOPERATIONACTION [4].

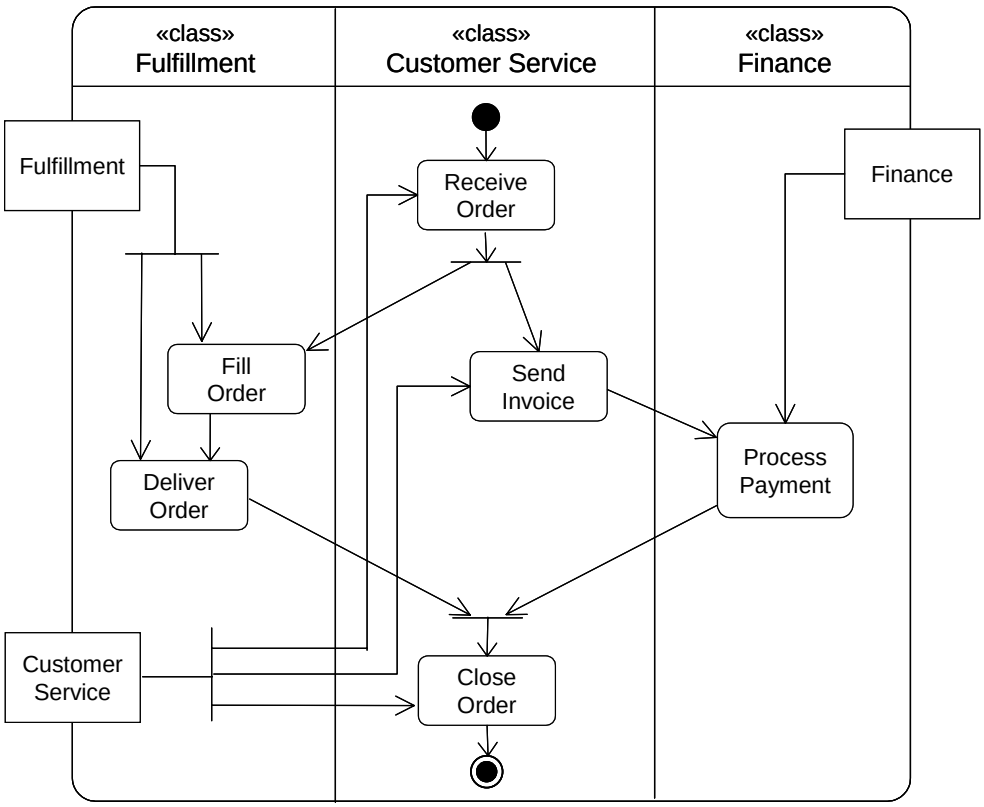

Figure 5: Partitions Representing Classes, with Object Flows

[5] An example repository model of CALLOPERATIONACTION is shown in Figure 8 of [4].

A more concise way to specify the target instances of CALLOPERATIONACTION is by navigation along attributes or associations from the same root object. Figure 6 shows an activity with partitions representing navigation from instances of class COMPANY, which has its own superpartition above the others. The navigated associations are shown in the subpartitions (in UML 2 association ends can be properties of the class from which they navigate).[6] The class model is shown in Figure 7. For each instance of COMPANY, navigating along the links FULFILLMENT, CUSTOMERSERVICE, and FINANCE will give the target instance for operation calls contained by the corresponding partition. This technique assumes that the entire activity is executed in the context of a single instance, for example as a method. Navigation proceeds from the context instance to the required targets of CALLOPERATIONACTION.[7] An alternate notation is shown in Figure 8.[8]

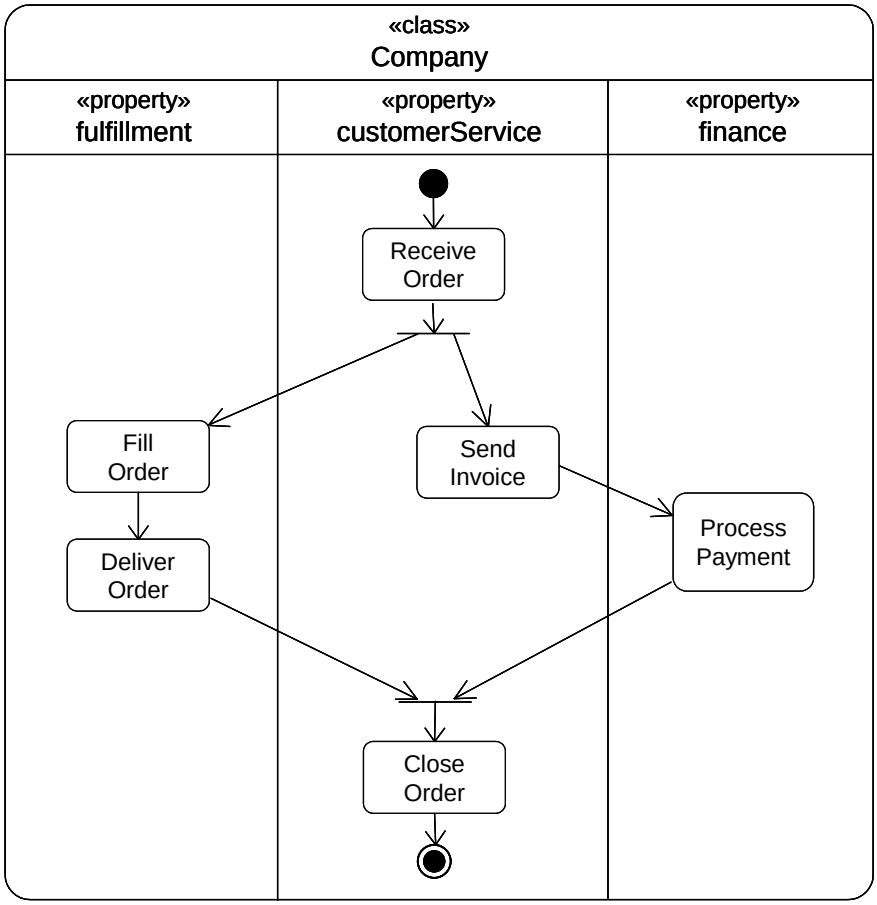

Figure 6: Partitions Representing Properties

[6] A completely class-based decomposition would show just the classes that are navigated through, for example, COMPANY and FULFILLMENT, but this will not tell exactly which instances are the targets of messages, unless the type of the properties are unique in the root class.

[7] Identifying instances by navigating from the same object is the basis of UML 2 composite structure model. This will be covered in a later article.

[8] A tool vendor could hardly be blamed for replacing double colons with a dot notation for nested navigation partitions.

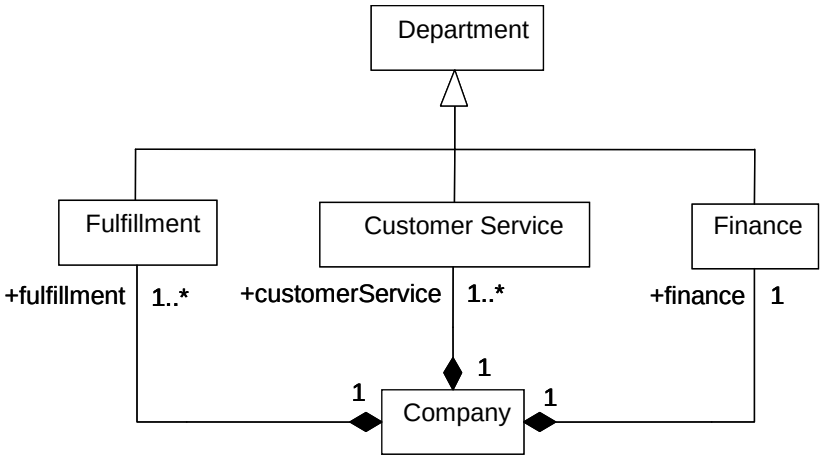

Figure 7: Class Model for Partitions in Figure 6

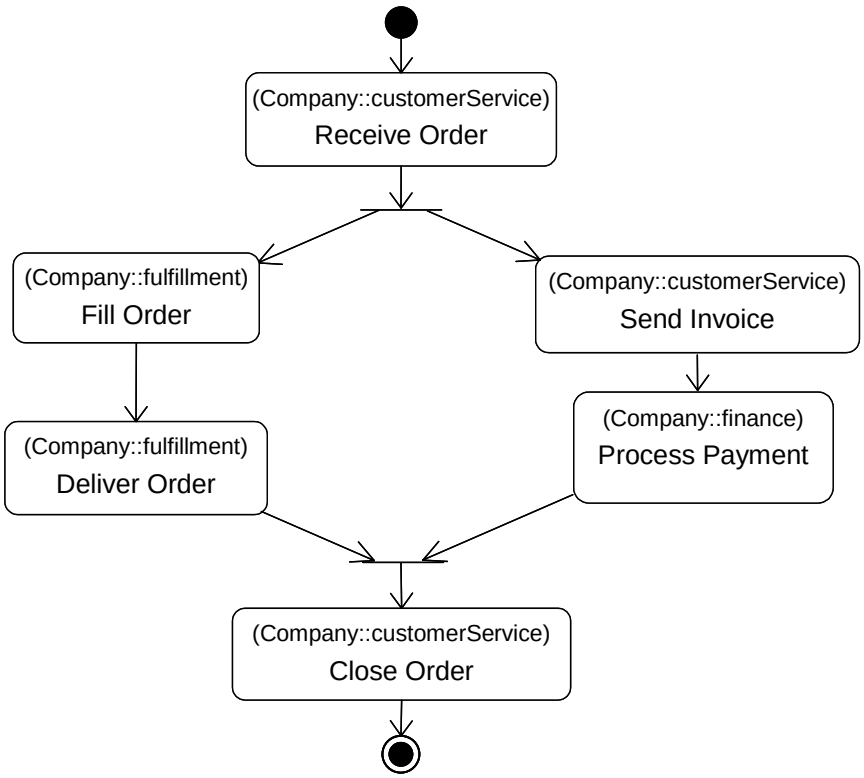

Figure 8: Partitions Representing Properties, Node-based Notation

Tools can add value to property subpartitions by automatically keeping the partitions consistent with the executable model. For example, they can generate the flows and navigation needed to specify the inputs of CALLOPERATIONACTION, as shown in Figure 9, while still presenting Figure 6 as the modeler's view. Figure 9 assumes the activity is a

method on the COMPANY class that has a parameter that is bound to the instance of COMPANY on which the method is invoked. This is shown as the COMPANY activity parameter node on the upper left. The various departments are retrieved from that instance with GETSTRUCTURALFEATUREACTIONs and passed to the actions needing them.

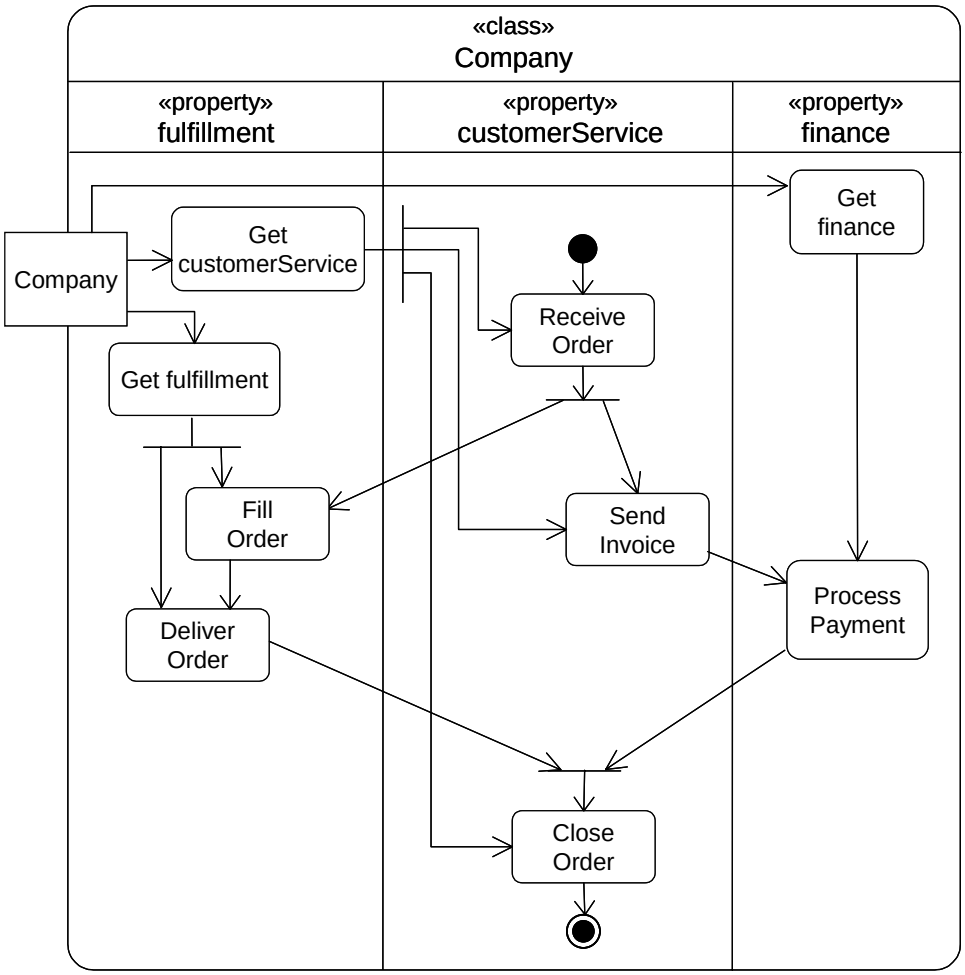

Figure 9: Partitions Representing Properties with Flows

Figures 3 and 6 refine Figure 1, but they do not dictate when refinement happens or if it happens at all. Figure 1 could be used for a long period even without the partitions before details are worked out. This 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 [2]. The flexibility to combine OO with domain-specific approaches considerably widens and integrates the potential applications of UML.

3 RELATION TO INTERACTION DIAGRAMS

Activities are an abstraction of the many ways that messages pass between objects, even with class and property partitions. In particular, the edges in an activity diagram, notated by arrows, can translate to one or many messages between objects, or none at all. This makes activities useful at a stage of development where the primary concern is dependency between tasks, rather than the protocols between objects. When messages are the focus of development, UML interaction diagrams are more appropriate. These have a very different semantics from activities, even though they use similar notations, such as arrows and rectangles. Interactions also define a kind of activity notation that is overlaid on the underlying interaction model, called the *interaction overview diagram*, which has a different semantics from activities, too.

The most important difference between activities and interactions is that edges connecting actions only indicate one action starts after another completes[9], they are not messages[10]. They can be translated to messages, but this involves more than the connected actions. For example in Figure 3, the edge between SEND INVOICE and PROCESS PAYMENT means that processing payment happens after sending an invoice. It does not necessarily imply a message sent between CUSTOMER SERVICE and FINANCE, as shown in Figure 10. In particular, the SEND INVOICE behavior cannot pass a PROCESS PAYMENT message to FINANCE, because SEND INVOICE is complete before PROCESS PAYMENT starts. This ensures that SEND INVOICE is reusable in other activities without restricting what happens before and after it. For example, the customer service department may be involved in other activities that send invoices but have different actions before and after it.

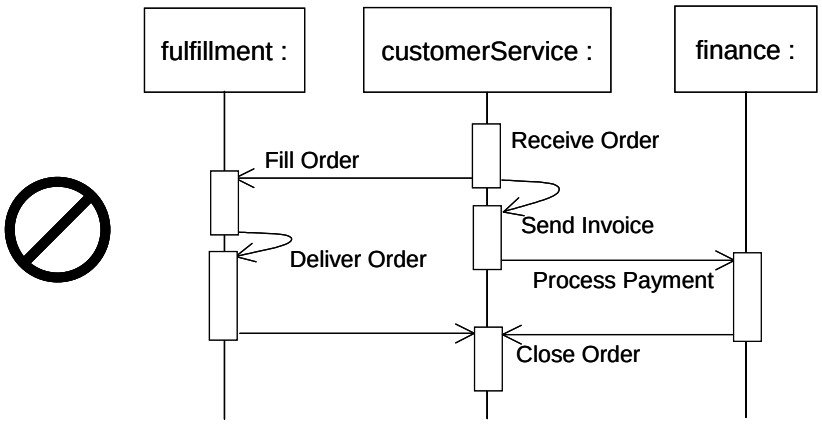

Figure 10: Interaction Diagram that is Inconsistent with Figure 6

[9] Except when used with streaming or optional parameters [4].
[10] This is a common misinterpretation of activities with class partitions, and has significant impact on methodologies that use both diagram types [7].

There are many ways that ordering of action execution might be achieved in a message passing implementation.[11] Figure 11 shows one possibility for the activity of Figure 6.[12] It uses a coordinator object to enforce the execution sequence, for example, a customer resource management system. The diagram says that the coordinator sends a RECEIVE ORDER message to the fulfillment department and after that is done, it sends other messages in parallel as shown, and when those are complete it sends the CLOSE ORDER message.[13] Alternative implementations could have one of the objects from the activity partitions coordinate everything, such as CUSTOMER SERVICE. Or all three objects could take on some portion of the process, such as FULFILLMENT coordinating FILL ORDER and DELIVER ORDER, as shown in Figure 12.[14,15,16]

[11] This is called *choreography* in web services [8].

[12] UML 2 interactions only use the navigation style shown in Figure 11, that is, they assume the targets of messages are found by navigating from a single instance. This is shown with a colon notation in the rectangle at the top of each lifeline. The name before the colon is the property name, the name after is the type of values the property holds, which is omitted in Figure 11.

[13] Interaction diagrams usually omit non-message actions, such as getting and setting attribute values, so there may be actions occurring between messages that are not shown on the diagram. Activities cannot omit steps in a sequence.

[14] In web service choreography, the processes inside interacting objects are private if the objects are independent companies. Figure 12 would be more typical of these applications. For example, the process internal to fulfillment might be hidden, and defined as its own activity diagram. Service-oriented architectures are highly decentralized in this respect. However, participants still agree on the pattern of public interactions between them, and must track where they are in these interactions in order to respond to each other properly. This is called *correlation.*

[15] The curved arrows in the interaction figures are a UML 1.x notation indicating an object that is sending a message to itself. It is not clear from the specification whether this carries over to UML 2, and will be addressed in finalization.

[16] The UML 2 communication diagram, which was called the *collaboration diagram* in UML 1.x, augments interaction diagrams with the connections between objects that are used to determine the targets of messages. Messages are shown passing along these connections, with numbers to show ordering. The relation between these diagrams is being clarified by the response to UML for Systems Engineering [9].

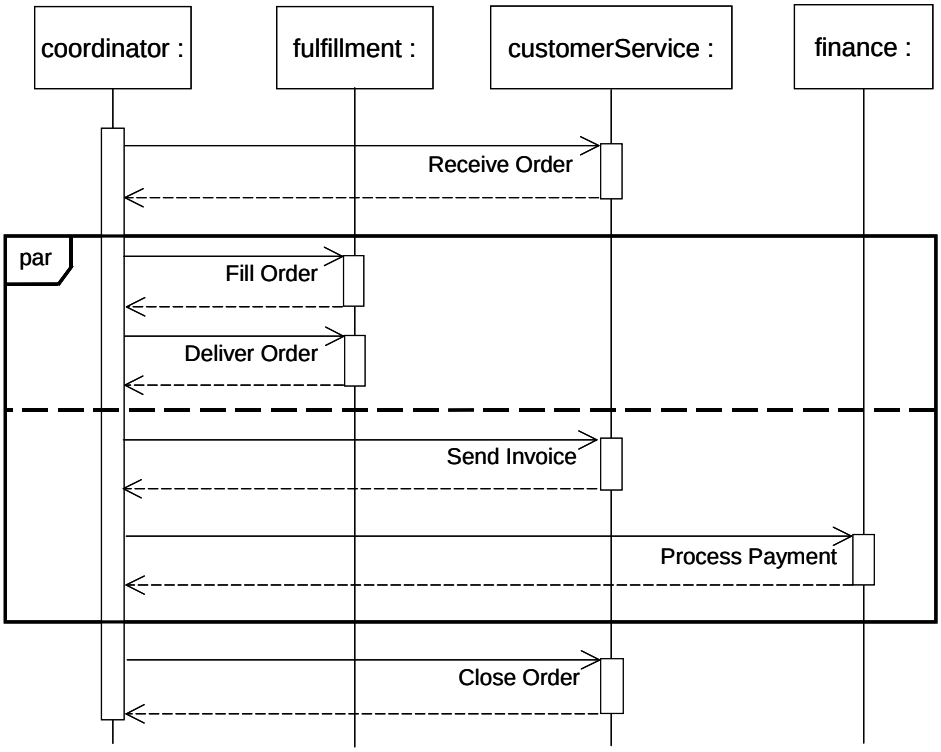

Figure 11: Possible Interaction Diagram for Figure 6

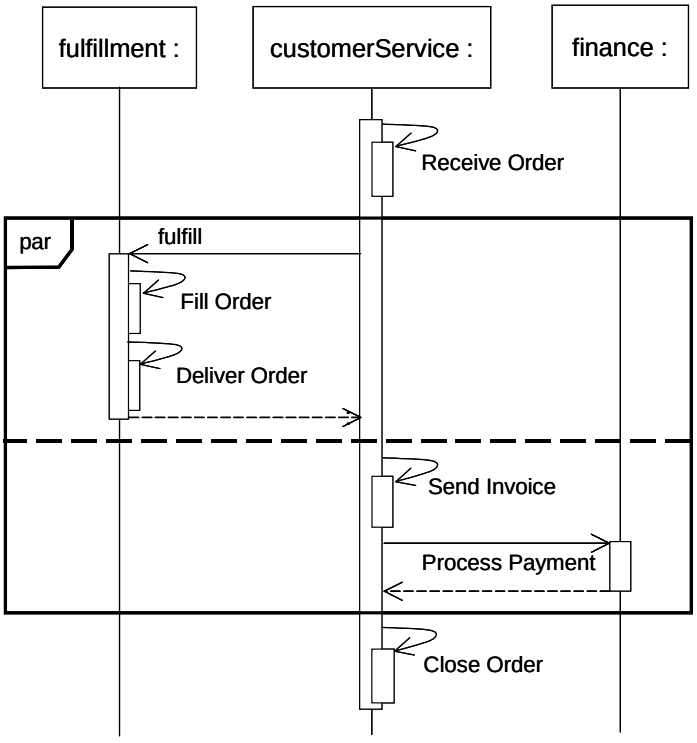

Figure 12: Another Interaction Diagram for Figure 6

The interaction overview diagram is an alternative notation for interactions that looks similar to activities, as shown in Figure 13, but is stored as an interaction in the repository. Each "action" can show a message or messages, or refer to an entire interaction. It is a way to highlight the control aspects of an interaction, but has the disadvantage of being large and hard to draw in some cases. The semantics is completely defined by interactions, rather than activities. For example, the rectangles labeled *ref* mean that the interaction named in the rectangle is copied in at that point, like a programming macro, rather than invoking a behavior as an action would.

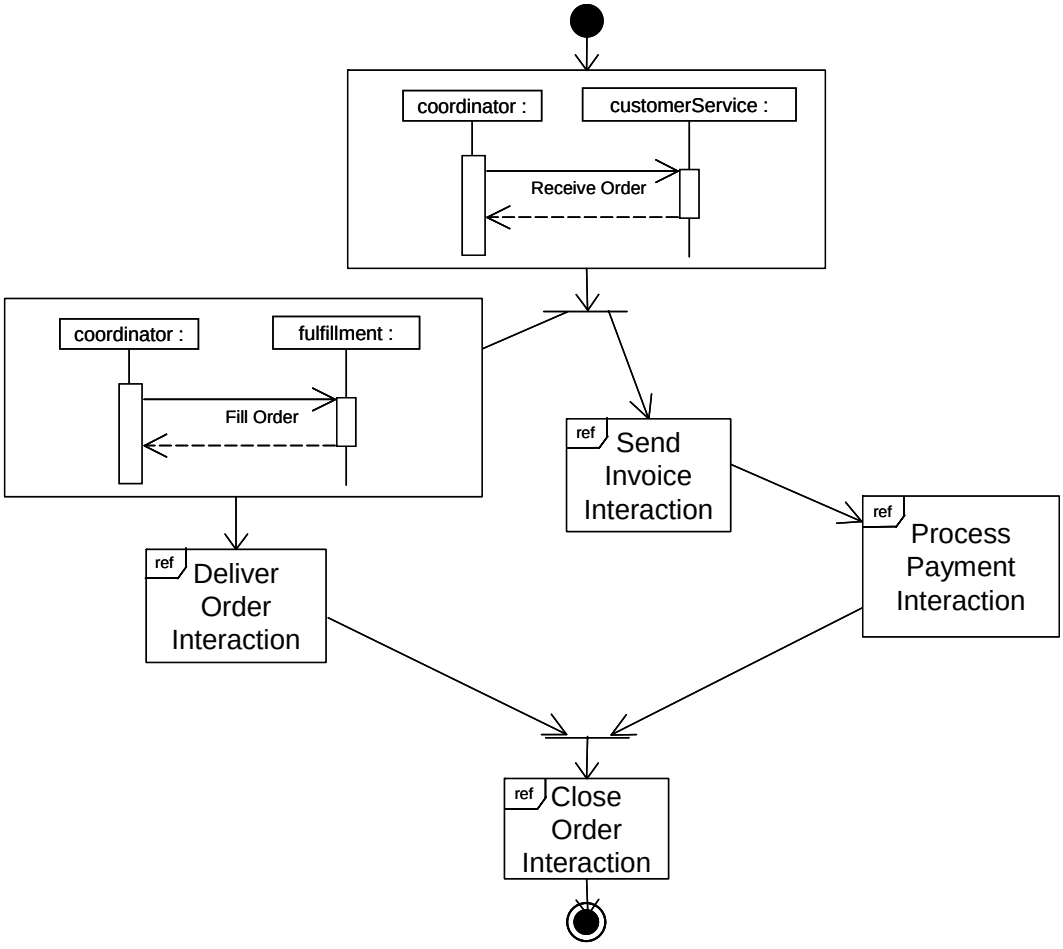

Figure 13: Interaction Overview Diagram

4 BEHAVIOR PROPERTY PARTITIONS

Behaviors in UML 2 are also classes, and their instances are running executions of the behaviors [2]. Behavior instances can carry information about executions, such as how long they have been running, what resources they have locked, as well as operations, such as suspend and resume. Partitions can specify values for executing behaviors. For example, Figure 14 shows the location where each behavior is performed (turned sideways for example in the next section). The property PERFORMINGLOCATION is a modeler-defined property, and can have values defined for its type, LOCATION, as shown by the class model in Figure 15. The COMPANYBEHAVIOR class defines a property to inherit to the various methods implementing the operations in Figure 4. The partitions of Figure 14 indicate the values of PERFORMINGLOCATION of the executing methods. The values are not assigned in the class model, because the methods may be used in other activities requiring different locations for the behavior executions.

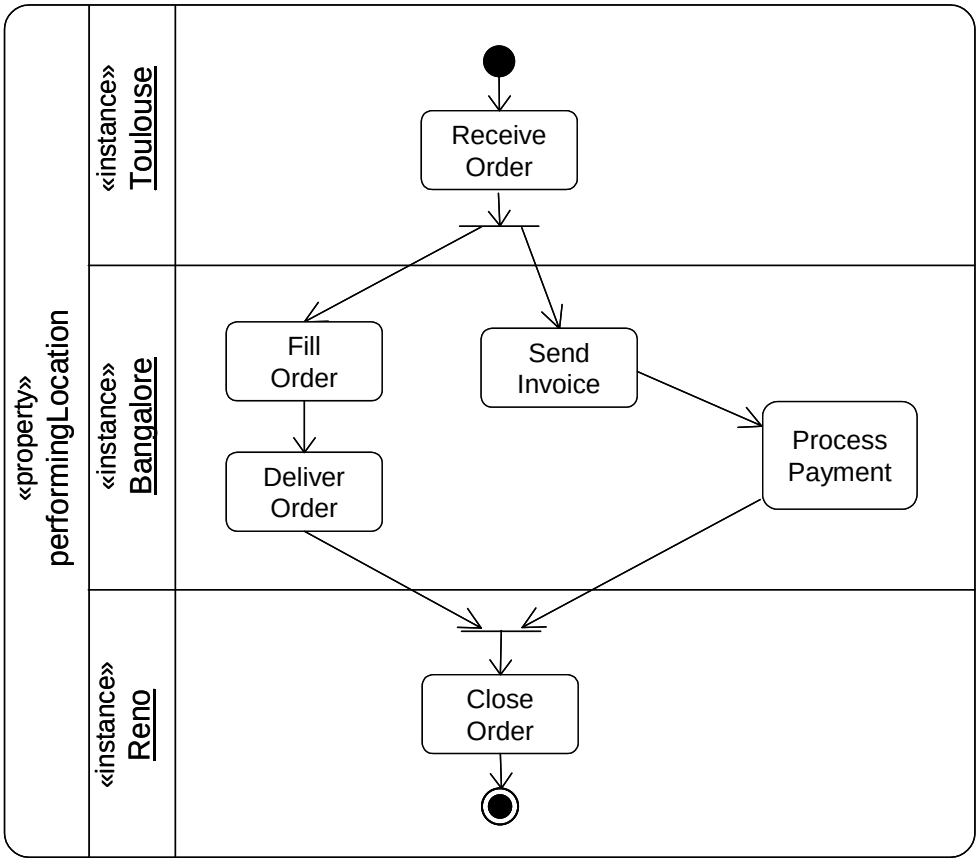

Figure 14: Partition as Attribute Value

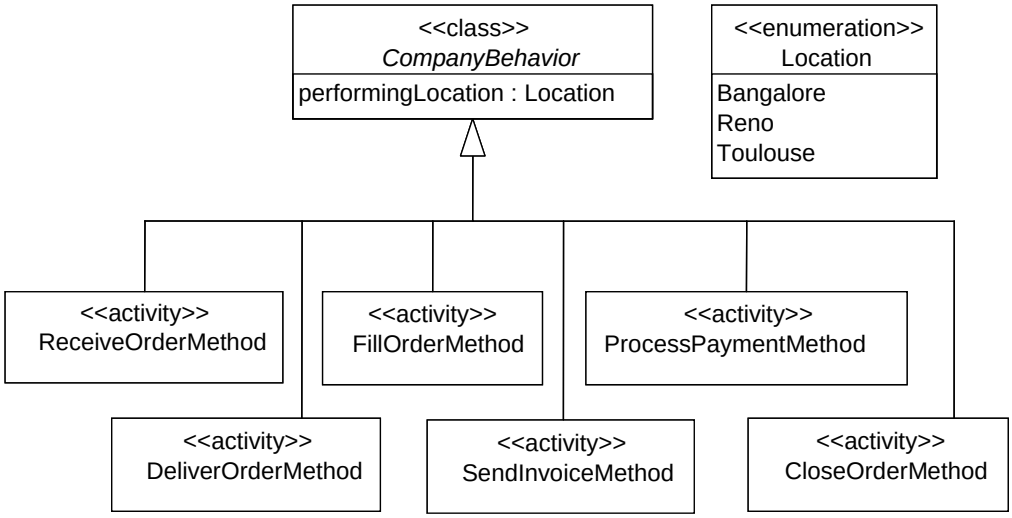

Figure 15: Class Model for Figure 14

5 STRUCTURED PARTITIONS

One way to structure partitions is to nest them, as already shown in Figure 6.[17] Another mechanism is to use more than one dimension, as shown in Figure 16, which combines Figures 6 and 14. Tools can add value to multidimensional partitions by allowing some to be hidden, and supporting different node positions for each dimension, if the application does not require showing them all at once. The node-based notation is shown in Figure 17.

[17] Nested partitions can also represent nested classes.

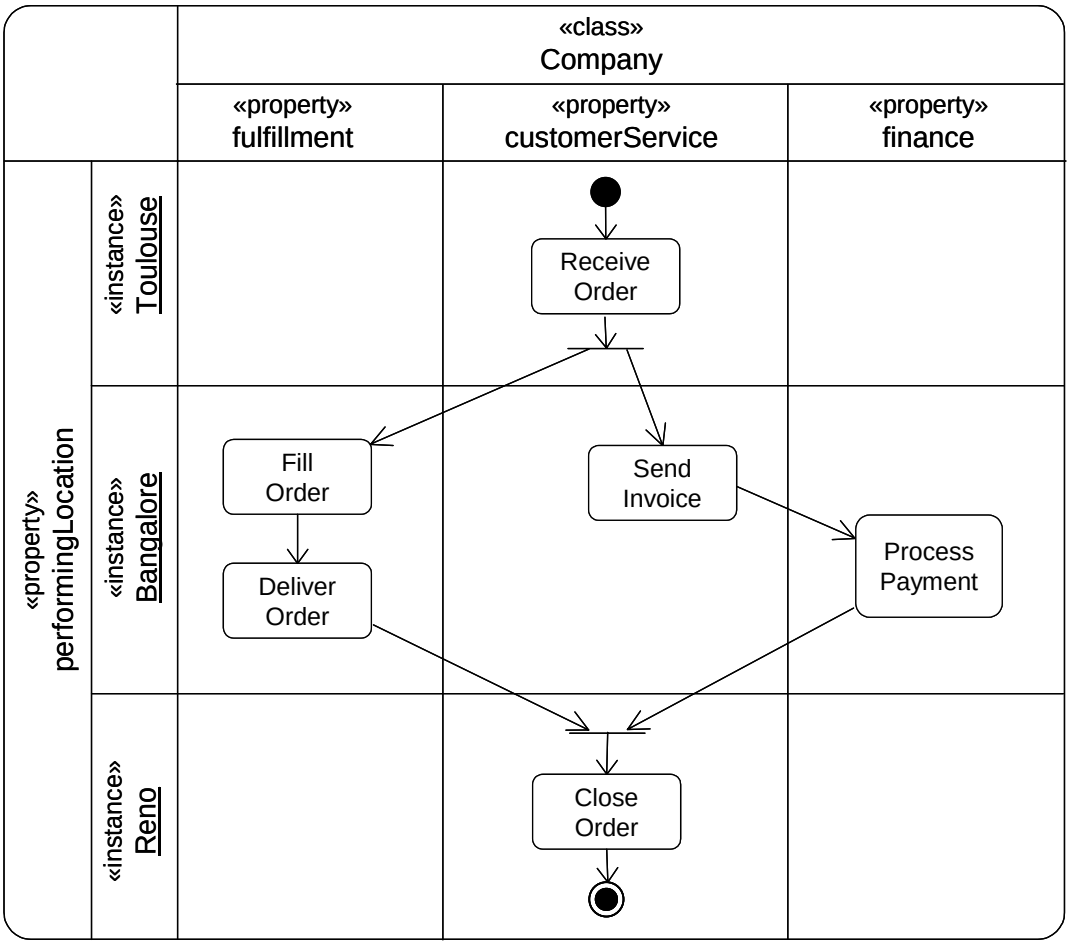

Figure 16: Multi-dimensional Partitions

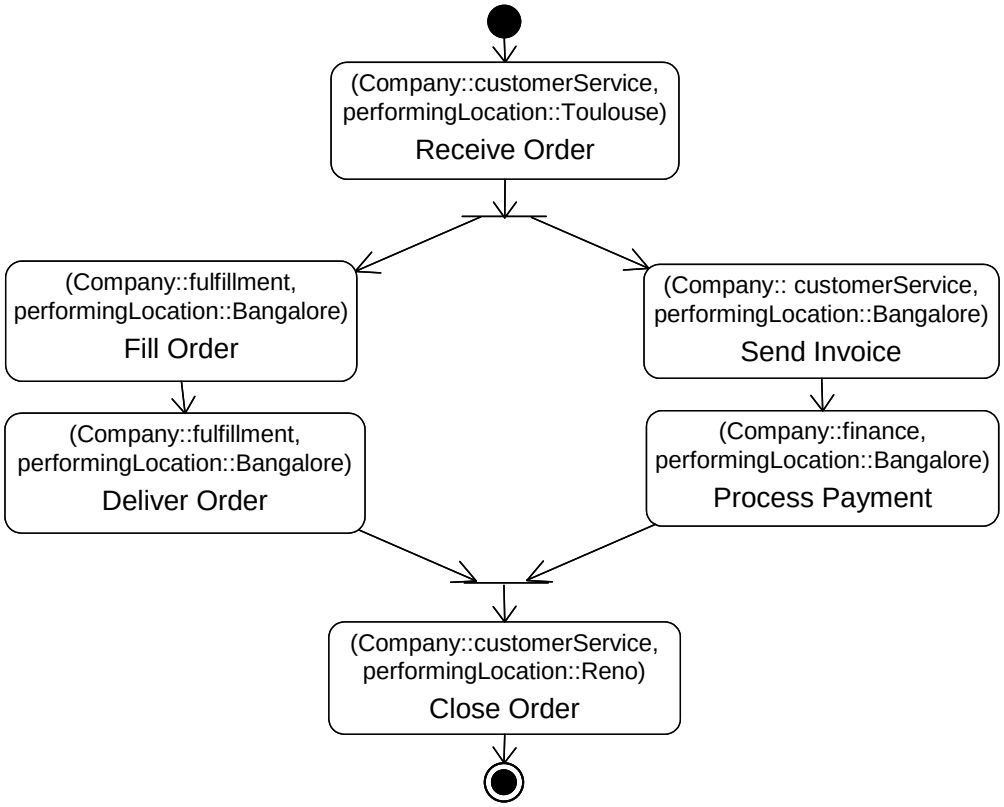

Figure 17: Multi-dimensional Partitions, Node-based Notation

A partial repository model for Figures 3, 15, and 16 is shown in Figure 18.[18] The top-level partitions are marked as dimensions, otherwise the repository cannot tell they are drawn at different angles. The model shows a CALLOPERATIONACTION for FILL ORDER and the partitions containing the action. The partitions at the bottom of the figure for COMPANY and FULFILLMENT indicate that the operation FILL ORDER must be owned by the type of the FULFILLMENT property, namely the FULFILLMENT class. The partitions at the top of the figure for PERFORMINGLOCATION and BANGALORE indicate that the execution of the method dispatched by FILL ORDER must be performed in Bangalore. The location property is inherited from COMPANYBEHAVIOR.

[18] The GENERAL association is derived from an additional metaclass for generalization that is not shown. The SUBGROUP association from partition to the partitions it contains should be named SUBPARTITION for consistency. This will be addressed in finalization.

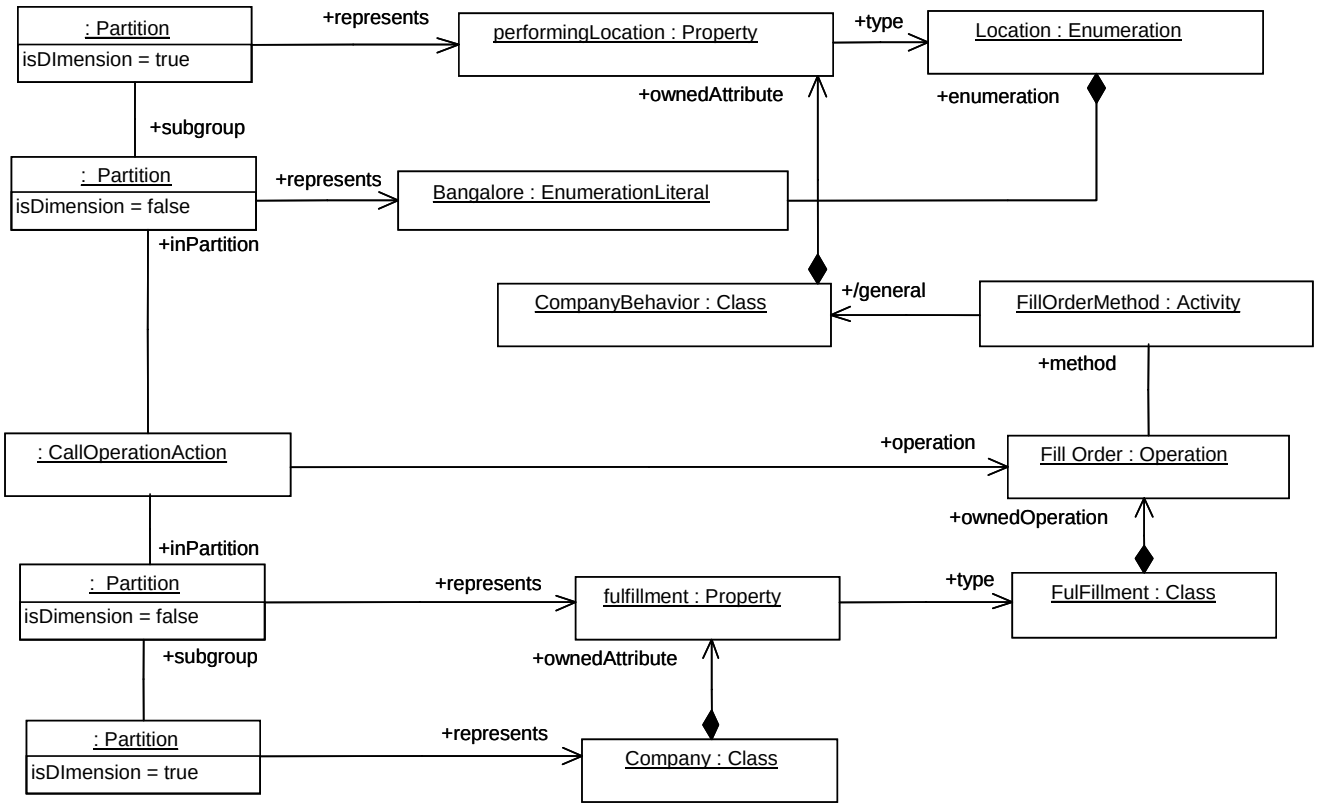

Figure 18: Partial Repository Model for Figures 3, 15, and 16

Partitions can be marked as external to the pattern set by the others in the same dimension, as shown in Figure 19. In this example, the external partition for CUSTOMER is not a part of the company, though presumably there is some navigation from the company to the customer. When using external partitions with multiple dimensions, the repository stores a hidden dimension partition over the external partition and its sibling, which is not shown in the diagram. For example, the COMPANY and CUSTOMER partition will be in a larger partition that is marked as a dimension. Otherwise, the repository cannot tell which dimension the external partition belongs to.

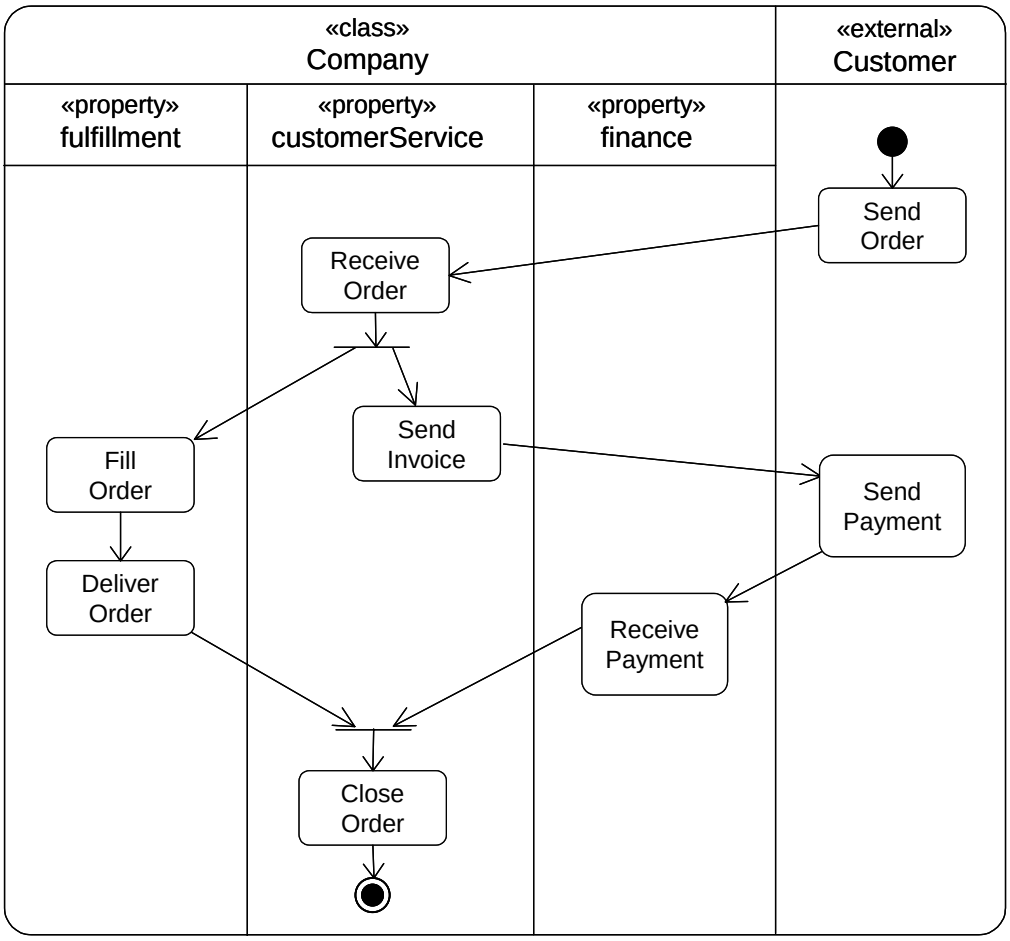

Figure 19: External Partition

6 MODELER-DEFINED PARTITION PATTERNS

The uses of partitions so far are defined in UML, but modelers can also define their own. Figure 20 shows an example of partitions representing states. An instance of ORDER moves through various states based on progress through the activity. There are many state machines corresponding to the activity, partly due to the many possible messaging implementations, as explained in section 3. The state machine for ORDER contains the states given in Figure 20, but the transition triggers between them would depend on whether the order or some other object is coordinating the actions, or both. For those actions coordinated by the order, there are more variations depending on whether this is done by an activity or a state machine. Assuming states coordinate the actions, as in the UML 1.x version of activities, the translation from the activity Figure 20 must account for differences in concurrency semantics between activities and state machines [5]. The one-to-many relation of activities to state machines is another example of the abstraction that activities provide for object implementations.

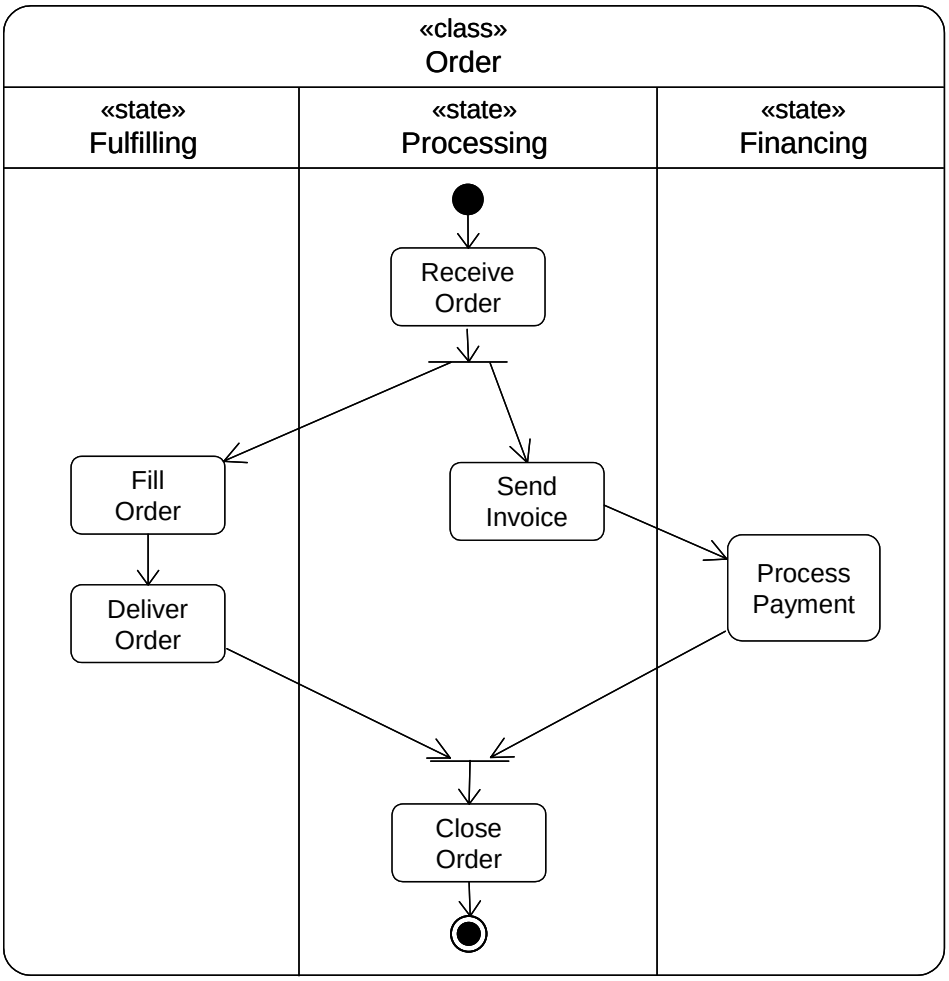

Figure 20: Modeler-defined Partition Pattern

7 CONCLUSION

This is the fifth in a series on the UML 2 activity and action models. It covers partitions, which are a way of grouping actions that have some characteristic in common. Typical usage patterns are described based on the element a partition represents: classes, properties of classes, and properties of behaviors. These patterns are combined using partition nesting and multiple dimensions. Various degrees of refinement are presented, to illustrate partitions used early in development for sketching, and the transition to later stages concerned with the details of execution. Partitions usually identify the entities responsible for actions, which is an abstraction of message passing as supported by interaction diagrams. A modeler-defined usage of partitions is presented to show an activity abstraction of state machines. These examples show how activities focus on task dependency, rather than object or state dependency.

ACKNOWLEDGEMENTS

Thanks to Evan Wallace and James Odell for their input to this article.

REFERENCES

[1] Object Management Group, "UML 2.0 Superstructure Specification," http://www.omg.org/cgi-bin/doc?ptc/03-08-02, August 2003.

[2] Bock, C., "UML 2 Activity and Action Models," in *Journal of Object Technology*, vol. 2, no. 4, July-August 2003, pp. 43-53, http://www.jot.fm/issues/issue_2003_07/column3.

[3] Fowler, M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, September 2003.

[4] Bock, C., "UML 2 Activity and Action Models, Part 2: Actions," in *Journal of Object Technology*, vol. 2, no. 5, September-October 2003, pp. 41-56, http://www.jot.fm/issues/issue_2003_09/column4.

[5] Bock, C., "UML 2 Activity and Action Models, Part 3: Control Nodes," in *Journal of Object Technology*, vol. 2, no. 6, November-December 2003, pp. 7-23, http://www.jot.fm/issues/issue_2003_11/column1.

[6] Odell, James, personal communication, 2004.

[7] Wagenhals, L., Haider, S., Levis A., "Synthesizing executable models of object oriented architectures," in *Journal of the International Council on Systems Engineering*, vol. 6, no. 4, pp. 266-300, October 2003.

[8] W3C Web Services Choreography Working Group, "WS Choreography Model Overview," http://www.w3.org/2002/ws/chor/edcopies/model/ModelOverview.html, December 2003.

[9] OMG Systems Engineering DSIG, "UML for Systems Engineering RFP," http://www.omg.org/cgi-bin/doc?ad/03-03-41, March 2003.

About the author

Conrad Bock is a Computer Scientist at the U.S. National Institute of Standards and Technology, specializing in process models and ontologies. He is one of the authors of UML 2 activities and actions, and can be reached at conrad.bock at nist.gov.