

STRUCTURED NODES IN UML 2.0 ACTIVITIES

HARALD STÖRRLE

Ludwig-Maximilians-Universität München
Oettingenstr. 67, 80538 München, GERMANY
stoerrle@informatik.uni-muenchen.de

Abstract. The upcoming major revision of the UML (see OMG [2003]) has introduced significant changes and additions to “*the lingua franca of Software Engineering*”. Within the UML, activity diagrams are particularly prominent, since they are the natural choice when it comes to the modeling of web-services, workflows, and service-oriented architectures. One of the most novel concepts introduced are so called structured nodes (StructuredActivityNodes in the metamodel). This concept includes features like loops, expansion regions, collection valued parameters, and data streaming.

Building on substantial previous work by the author, the purpose of this paper is to understand better these new concepts and notations, and actually defines a semantics for them. Since the UML standard is still immature in some parts, this article is restricted to those concepts, for which a reliable interpretation is currently possible. This article is followup to Störrle [2004d].

ACM CCS Categories and Subject Descriptors: D.2.1, D.2.2, D.2.10, D.3.3

Key words: UML 2.0, activity diagrams, structured nodes, loops, collection-valued parameters, streaming

1. Introduction

The modeling of business processes and workflows is an important area in industrial software engineering, and, given that it crucially involves domain-experts which are usually non-programmers, it is one of those areas, where model-driven approaches definitely have a competitive edge over code-driven approaches. As the UML has become the “*lingua franca of software engineering*” and is the cornerstone of the Model Driven Architecture initiative of the OMG, it is a natural choice for this task. Within the UML, activity diagrams are generally considered to be the appropriate notation for modeling business processes, workflows, and system-level behaviors, such as the composition of web-services. Unfortunately, the ActivityGraphs¹ of UML 1.5 have certain shortcomings in this respect, one of which is the lack of structuring mechanisms within ActivityGraphs. The only way to model non-linear control- and data flow is the DecisionNode (depicted by a diamond-shaped symbol), that is, a goto, with all its negative repercussions.

In programming languages, the goto-construct has long since been abandoned. Modern programming languages offer a whole range of structuring constructs like

¹ Adopting the convention of the standard, words in “CamelCaps” refer to meta-classes.

different kinds of loops, properly nested if/then/else or case/otherwise expressions, exceptions, transactions and more. None of these are present in UML 1.x. Unsurprisingly, some people consider the ActivityGraphs of UML 1.5 as “*spaghetti-diagrams*”. And consequently, the OMG has addressed this problem by adding the concept of StructuredActivityNode (“*structured node*”, for simplicity) in the new version of the UML.

However, all of these constructs are new in the UML. For some of them, there is the experience from programming languages or other notations to guide us. For instance, exceptions in, say, Java, or conditional nodes in Nassi-Shneiderman-Diagrams. Some other constructs, however, seem to be new altogether, like ExpansionRegions. In either case, there is little to no experience yet as to how these constructs integrate with the rest of UML activity diagrams.

Also, it is currently unclear what the precise semantics of these new concepts is, for in the standard this is described only in a very superficial way, without the formal rigor necessary for practical model exchange, or enactment, or verification tools. In fact, the standard does not even provide adequate examples to illustrate the intention behind these constructs and to clarify their meaning.

This paper thus explores the new notions both syntactically and pragmatically, striving to provide a plausible and viable interpretation for the various types of Structured Activity Nodes, including at least a partially formal definition of their semantics. Be warned that the UML is a moving target, however, and so this paper is restricted to those concepts for which an interpretation can be currently provided with sufficient validity; for the remaining concepts, some interpretations are offered, but they are more on the speculative side (see section 5).

2. Activity diagrams

Activity diagrams serve many purposes, during many phases of the software life-cycle, and are a tool for many roles:

- They are intended for being used for describing all process-like structures, e.g. business processes, software processes, use case behaviors, web services, and algorithmic structures of programs, though other purposes like hardware-descriptions are also conceivable.
- Activity diagrams are thus applicable throughout the whole software lifecycle, i.e., during business modeling, acquisition, analysis, design, testing, and operation, and in fact in many other activities.
- Thus, they are intended for usage not just by Software-Architects and Software-Engineers, but also by domain specialists, programmers, administrators and so on.

Activity diagrams are best explained by means of an example (see Figure 2.1, adapted from OMG [2003, Fig. 203, p. 290]). The example consists of two diagrams, a class diagram (left) and an activity diagram (right). Note, that the class Order has a third compartment that contains a reference to the activity diagram

OrderBehavior. This way, the class diagram sets the necessary context for the activity diagram, providing us with an inscription language for the activity diagram (see Section 3.1 for more on contexts).

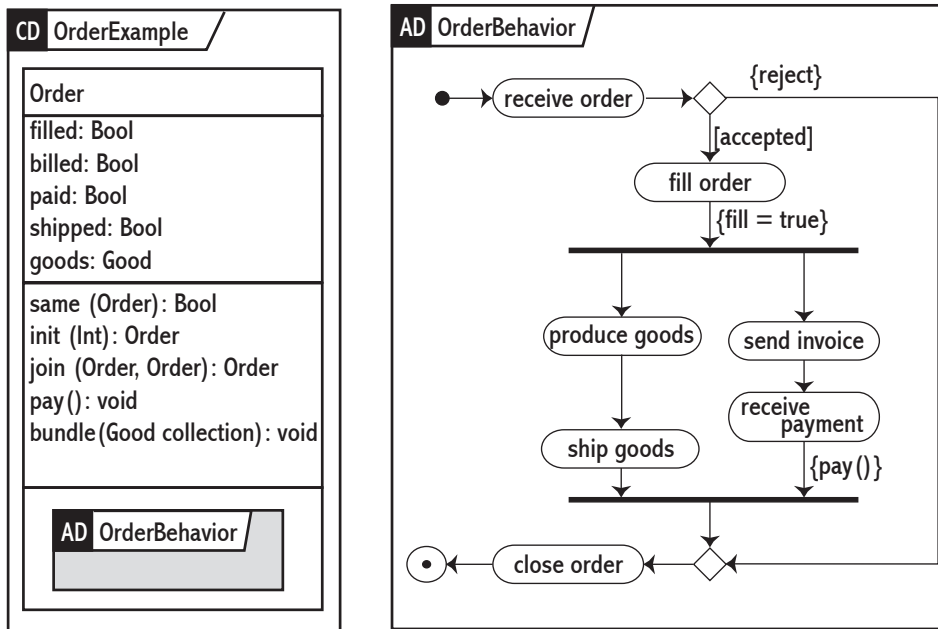


Fig. 2.1: An introductory specification consisting of a class diagram (left), and an activity diagram (right). The icon in the third compartment of class Order is a reference to the Activity specified in diagram OrderBehavior.

Concerning on the activity diagram, the following behavior is intended. First, an order is received by the action `receive order`. The diamond-shape represents a decision node. The expression with square brackets on the first branch asserts that if this branch is taken, the context is in state `accepted`. Note that guards like this may, but need not correspond to attributes of the context. Next, the action `fill order` is executed, the behavior in curly brackets is invoked, and a fork (the first black bar) is reached. The fork splits the path of control flow into two.² On the left path, the actions `produce goods` and `ship goods` are executed. On the right path, the actions `send invoice` and `receive payment` are executed. After `receive payment`, the behavior `pay()` is invoked. Both paths are pursued concurrently. When they have both completed their execution, the join (the other black bar) may take place, and the action `close order` is executed. Returning to the decision node above (after `receive order`), its second branch invokes the behavior `reject`. Note, that this behavior is not specified by the context.

In general, this is only an intuitive example, not a complete one: a model compiler would probably reject it, or at least warn the user that the system is underspec-

² It is currently undefined by the standard what happens to data tokens here.

ified. It is also arguably the case, that the problem in question should be modeled in a different way, but for the sake of argument, I will stick to this example, since it is the one used throughout the standard itself.

This example goes beyond the UML standard, in that each diagram is put into a frame denoting its type and name, while the standard defines this only for interaction diagrams. It is good practice, however, and should be followed, whenever specifying a system. Observe that it is only this feature that allows us to specify `OrderBehavior` separately in a diagram of its own, and reference it from within the third compartment of `Order`. The procedure proposed in the UML standard would require us to draw the whole Activity (and only a single one!) in the third compartment.

3. Activities

A detailed discussion of the concrete and abstract syntax of UML 2.0 Activities, the semantic domains of procedural and colored Petri-nets, respectively, and the semantic mapping of control- and data-flow, and exceptions of Activities is found in Störrle [2004b], Störrle [2004a], and Störrle [2004c].

In UML, the abstract syntax is defined by the metamodel (also called abstract syntax). The various notations and diagram types are the concrete syntax. The concept underlying the notation of activity diagrams is called Activity. In this section, this concept and related metaclasses are explained.

3.1 Abstract syntax

Compared to UML 1.5, the concrete syntax of activity diagrams has remained mostly the same, but the abstract syntax and semantics have changed drastically. In the new UML metamodel, the whole area of Activities has been structured into a graph of packages of increasing expressive power (see Figure 3.2).

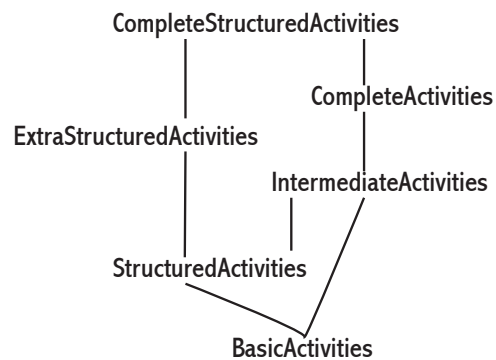


Fig. 3.2: Levels of expressiveness in activity diagrams.

Figure 3.3 shows a small portion of the metamodel concerned with StructuredActivityNodes. An Activity consists of several ActivityEdges and ActivityNodes. The nodes are connected by the edges, so that an Activity is basically a graph of ActivityNodes.

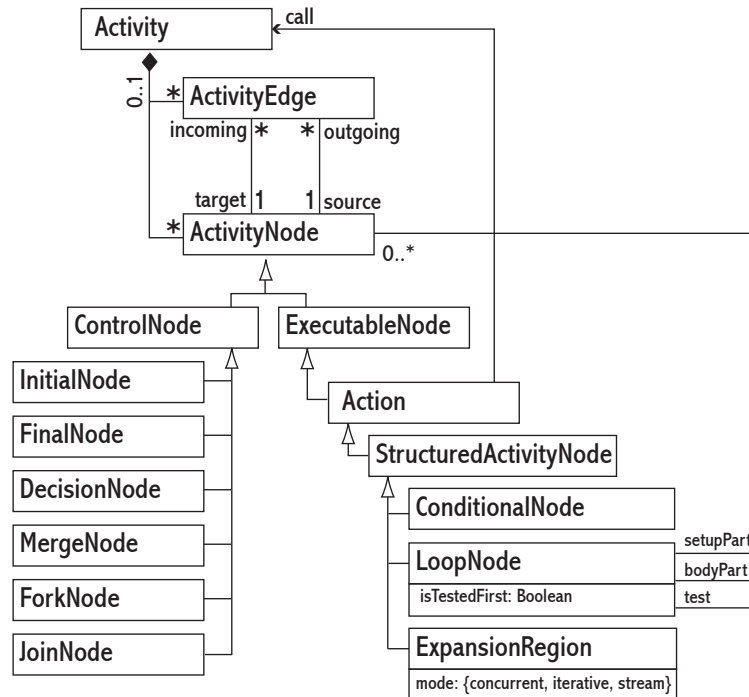


Fig. 3.3: The UML 2.0 metamodel as far as ExpansionNodes are concerned (simplified).

ActivityNodes may be separated into two groups: ControlNodes and ExecutableNodes. Each of these groups has several subclasses. The subclasses of ControlNode are rather similar to the corresponding concepts in UML 1.5. For the purposes of this article, it is sufficient to consider only Actions and StructuredActivityNode. The latter has three subclasses, ConditionalNode, LoopNode, and ExpansionRegion, which are examined in sections 4.1, 4.2, and 5 in turn.

As an aside, the metaclass StructuredActivityNodes is a subclass of Action³

The standard also states that “*behaviors, as such, do not exist on their own [..]. If a behavior operates on data, that data is obtained from the host object. [..] A behavior has access to the structural features of its host object.*” (cf. OMG [2003, p. 369]) This is sometimes described as the embodiment of behaviors in UML 2.0 (see Figure 3.4 for the relevant portion of the metamodel). It may be visualised in the way shown in Figure 2.1.

³ StructuredActivityNode is also a subclass of ActivityGroup, but since this is only “*a generic grouping construct for nodes and edges*” and has “*no inherent semantics*” (cf. OMG [2003, p. 301]), they may be safely ignored here.

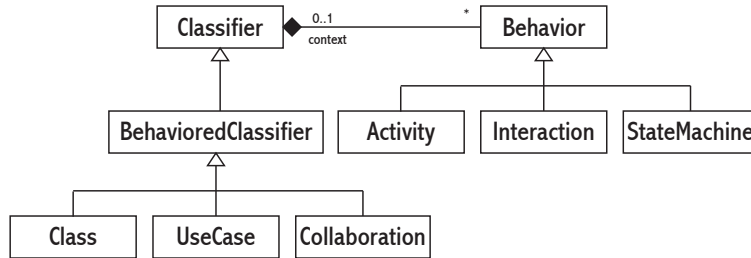


Fig. 3.4: Metamodel of Embodiment

3.2 Semantic domain

Since the standard stipulates that Activities “use a Petri-like semantics” (cf. OMG [2003, p. 292]), it is natural to use Petri nets as the semantic domain. Petri-nets have the additional advantages of providing an almost trivial mapping for basic Activities (see next section), something that is not at all easy in other formalisms. Grace to the straightforward mapping, traceability is also ensured and thus the semantics as such and every translated model can be easily validated. Consider Figure 3.5 for an example. There, the activity diagram of Figure 2.1 is mapped intuitively to a Petri-net and a CCS-expression. Obviously, the structure of the model remains visible (literally) in the Petri-net translation, but is lost in the CCS-translation.

Capturing hierarchical abstraction and data-flow requires high-level nets, of course. In Störrle [2004b, 2004a, 2004c], I have shown how control-flow, procedure calling, data-flow, and exceptions in UML activity diagrams can be mapped to different variants of Petri Nets. For the purposes of this paper, flat higher-order nets are sufficient. For pragmatic reasons such as availability of good tool support, I propose to use Colored Nets Jensen [1992].

3.3 Semantic mapping

While in UML 1.5, activity diagrams have been defined as a kind of State Machine Diagrams (ActivityGraph used to be a subclass of StateMachine in the Metamodel), there is now no such connection between them: “Activities are redesigned to use a Petri-like semantic” (cf. OMG [2003, p. 292]). The intuition of the semantic mapping is presented in the first six compartments of Figure 3.6.

For elementary Activities, the mapping to Petri-nets is rather simple, as the example in Figure 3.5 already suggested. Intuitively, Actions that are ExecutableNodes become net transitions, ControlNodes become net places or small net fragments, and ActivityEdges become net arcs, possibly with auxiliary transitions or places. This intuition is described in detail in Figure 3.6. There, each field shows the intuition of the translation for one area of Activities. Each activity diagram fragment (left column) maps to the corresponding Petri-net fragment (right column). The

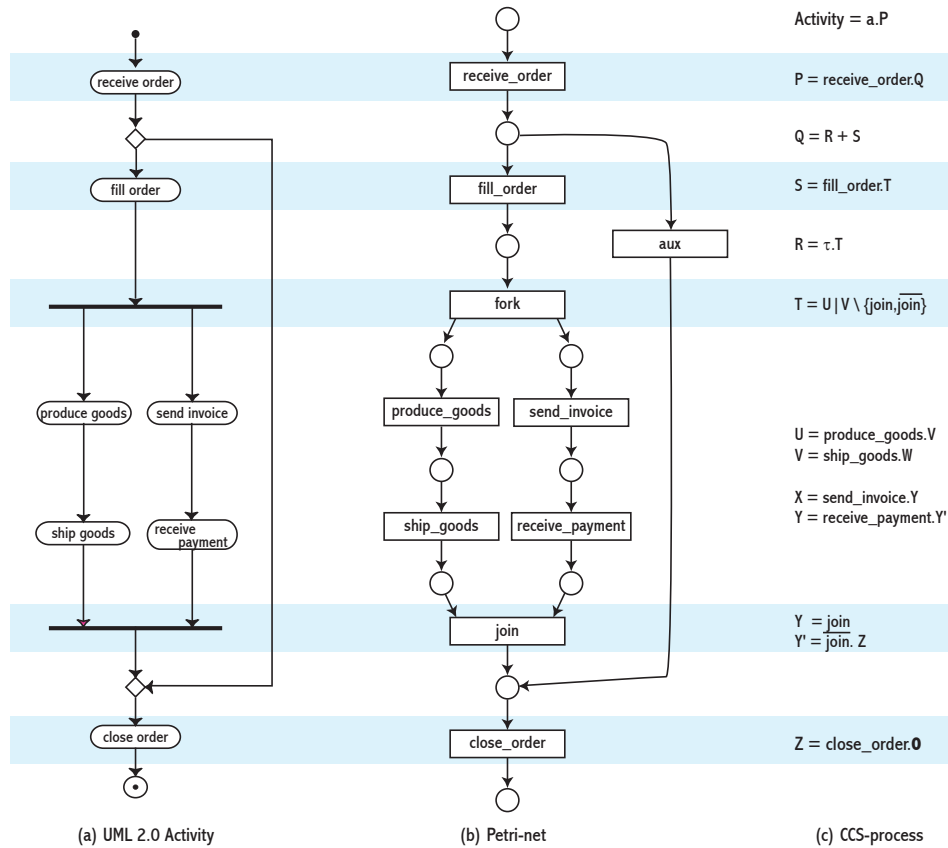


Fig. 3.5: Comparison of two semantic domains by example: (a) basic activity diagram, (b) P/T-net, (c) CCS-process.

details are defined in Störrle [2004b, 2004a]. Exceptions are left out (see Störrle [2004c] for this issue).

For data-flow, the mapping is not quite as easy, and also requires colored Petri-nets as the semantic domain to cover data-types, guards, and arc-inscriptions. As Actions may call Activities, transitions should be able to call nets like procedures. Obviously, this goes beyond traditional P/T-nets, so that Störrle [2004b] resorts to procedural Petri-nets (first described in Kiehn [1989]). I require that each Activity is represented by a separate boxed and named activity diagram similar to UML 2.0 Interaction Diagrams (cf. Figure 4.10 and Störrle [2003]), each of which may then be transformed separately into a plain Petri-net. These individual nets are held together by a refinement relationship exploited at run-time, i.e., when a family of Petri-net is executed.

With such a mapping, standard Petri-net tools may be applied to validate the mapping, and in fact to validate models. See Figure 3.7 for a screenshot of using the CPN Toolset for this purpose on the running example.

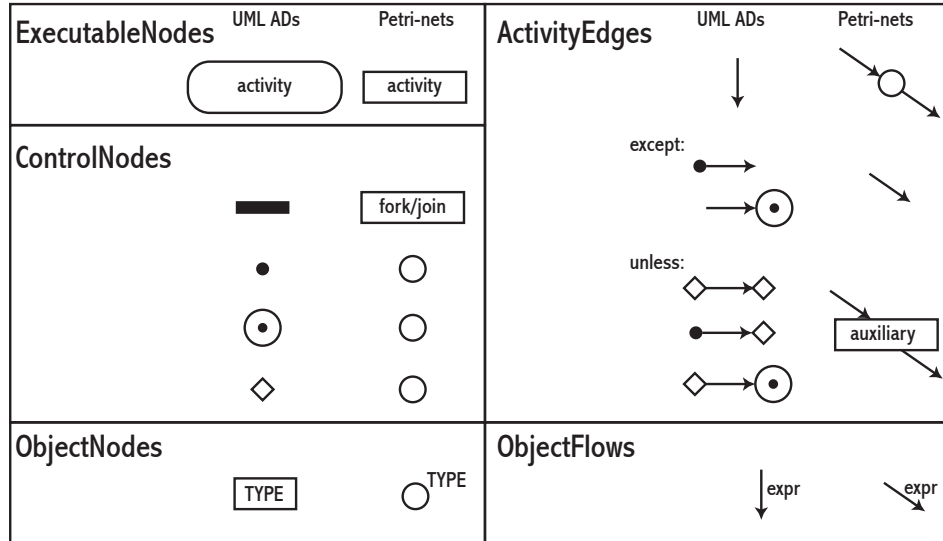


Fig. 3.6: The intuition of the semantic mapping for Activities. Actions that call Activities are represented as Petri-net transitions with a double outline. They are translated into refined transitions of a procedural Petri-net.

4. Simple StructuredActivityNodes

In this section, ConditionalNodes and LoopNodes are explained. They can be explained in terms of syntactic sugaring of more elementary constructs, the semantics of which has been defined in Störrle [2004b, 2004a] (see also Section 3.3 above).

4.1 ConditionalNodes

A ConditionalNode is a kind of set of guarded commands: “a conditional node is a structured activity node that represents an exclusive choice among some number of alternatives.” (cf. OMG [2003, p. 313]). Each consists of a set of “clauses, [each consisting] of a test section and a body section”. When executing a ConditionalNode, all tests are executed. Then, the body section of one of those that yielded true is chosen nondeterministically and executed. Alternatively, “sequencing constraints may be specified among clauses”.

The standard gives no hint concerning the concrete syntax of a conditional node. Thus, I propose a representation similar to that of ExpansionRegion, namely a dashed box with Pins (i.e., ObjectNodes) for the input- and output-parameters and one compartment for each pair of condition and consequence, separated by dashed lines (see Figure 4.8, left).

In order to reduce complexity, I propose that the test sections be side-effect free. Thus, it is probably the easiest to use the guards introduced with DecisionNodes. Also, I suggest that the body section consist of single Actions. These may call upon other Activities, of course. Then, ConditionalNodes are just syntactic sugar,

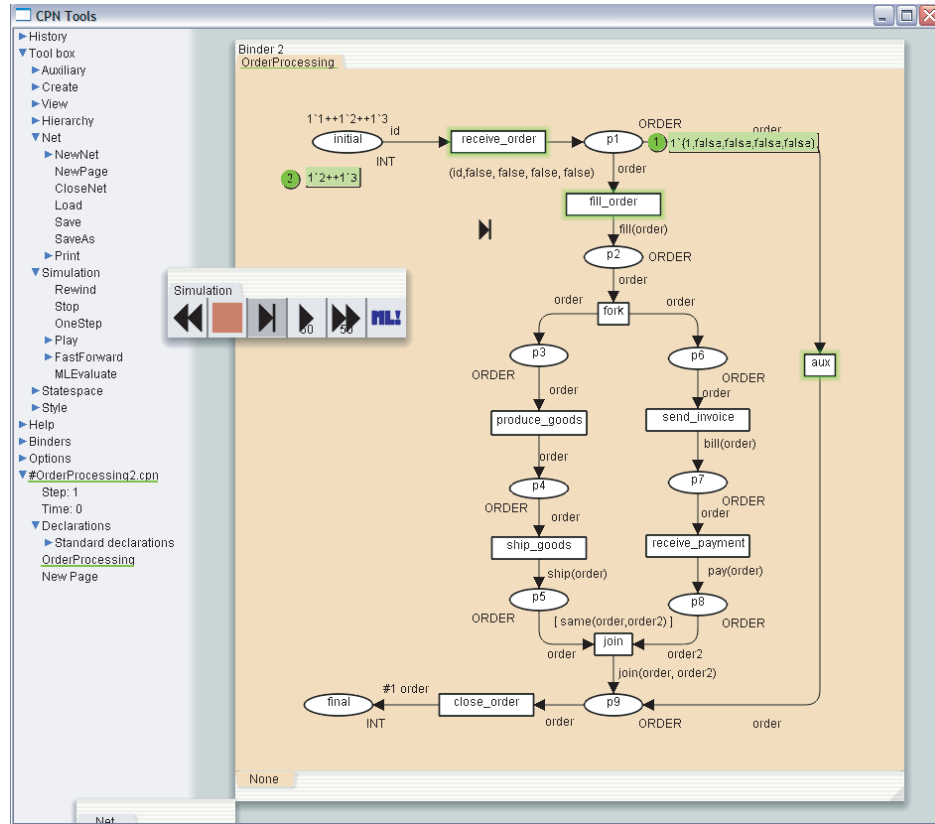


Fig. 3.7: Validation the semantic mapping in general and specific models in particular using the CPN Toolset.

and their meaning is best defined by an expansion into more basic constructs. Figure 4.8 (right) shows how this may be done.

4.2 LoopNodes

A loop node has “*setup, test, and body sections*” (cf. OMG [2003, p. 341]), all of which consist of a set of ActivityNodes. Also, “*the test section may precede or follow the body section*” (ibid.), depending on the value of the attribute `isTestedFirst` (see Figure 3.1). Also, “*the setup section is executed once on entry to the loop, and the test and body sections are executed repeatedly until the test produces a false value*” (ibid.).

Additionally, a loop node may have several OutputPins for outputting various values, including the loop variable, the test value, and so on (these are ignored in the remainder). The initial value for the loop variable is provided on an InputPin. Note, that intermediate values may be stored in the context.

As with conditional nodes, the standard gives no hint on the concrete syntax of a

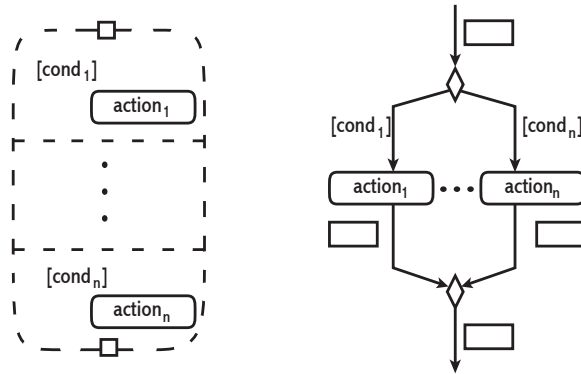


Fig. 4.8: ConditionalNodes are syntactic sugar: sugared form (left), and its meaning (right).

LoopNode. I propose to also represent it similar to an ExpansionRegion (and my proposal for ConditionalNodes) as a dashed line with ObjectNodes for the input- and output-parameters and compartments for setup, test, and body regions separated by dashed lines, see Figure 4.9 (a).

There is only a single possible interpretation of LoopNodes: setup, test, and body compartments are individual Actions (possibly refined by Activities) in a simple Activity like those presented in Figure 4.9 (b) and (c). These two possibilities implement both while-do and repeat-until loops, that is, loops where the condition to continue execution is tested before or after executing the loop body. If the former case is intended, the `isTestedFirst`-attribute of LoopNode is set to true, in the latter case, it is set to false.

Another interpretation is presented in Figure 4.9 (d). There, the test is treated similar to the guards in ConditionalNodes. This interpretation is not admissible according to the standard, however, since test is modeled explicitly is an Activity-Node.

There is no syntactic feature in the standard to distinguish between while- and repeat-loops. Thus, I propose to use the sequence of the body and test sections in a LoopNode, that is, if the test section stands above the body section as in Figure 4.9 (a), the test is executed first, implementing a while-loop. For until-loops, the sections are simply interchanged. This would avoid an additional inscription, and make the type of loop very obvious even for people that are not familiar with the underlying concepts.

4.3 Open questions concerning LoopNodes

So far, interpreting the standard has been rather straightforward. There are some questions concerning the interpretation of LoopNodes, however. For instance, the standard declares that a LoopNode “*is a costructured activity node that represents a loop with setup, test, and body sections.*” (cf. OMG [2003, p. 341]). Unfor-

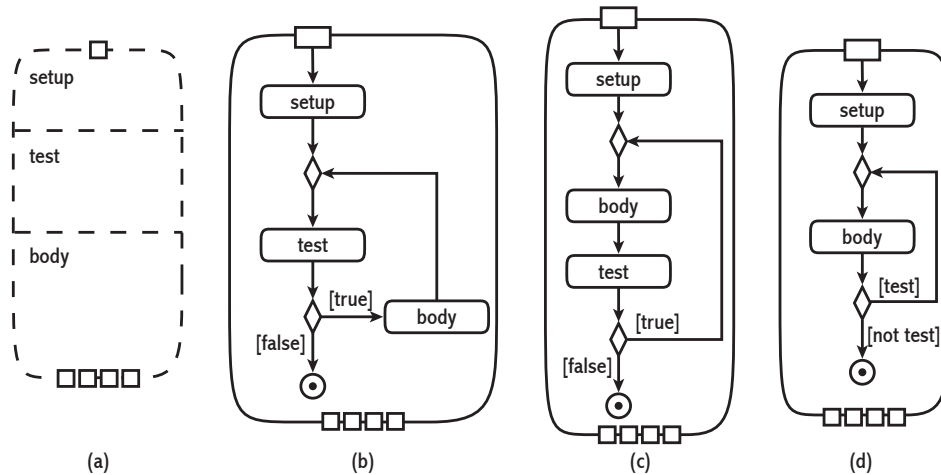


Fig. 4.9: LoopNodes as syntactic sugar: sugared form (a) and possible expansions (b–d).

tunately, the meaning of the word “*costructured*” remains opaque. The standard goes on by saying that “*each section is a well-nested subregion of the activity whose nodes follow any predecessors of the loop and precede any successors of the loop.*” (cf. OMG [2003, p. 341]). This seems to indicate that a LoopNode is just a kind of additional structure on top of an Activity. That is, the activity diagram fragment shown in Figure 4.10 (left) is identical to the Activity diagram fragment of Figure 4.10 (right), and the LoopNode structure shown there has no meaning at all, and is just a kind of reading guide that overlays the structure of the Activity. But then, why are there LoopNodes in the first place?

Also, consider the interaction of loops with exceptions. A natural construction one would expect to be admissible (and simple enough semantically) is the situation shown in Figure 4.11 (a) and (b).⁴ What is the scope of the exception—is the body section also an InterruptibleActivityRegion? Or should it be possible that the exception is raised in the test and/or setup, too, as suggested by Figure 4.11 (c)?

Actually, both situations make sense. So I propose to follow the interpretation shown in Figure 4.10 (b) where the LoopNode is a plain Action refined by the Activity of, say, Figure 4.9 (b). This node is now also the protected node of the exception. This way, it is possible to raise an exception anywhere in the loop. If the scope of the exception is supposed to be more restricted, the Activity that refines the “*looping action*” node must raise an exception.

5. ExpansionRegions

In this section, ExpansionRegions are explained. The description of ExpansionRegions in the UML standard exhibits far more errors and inconsistencies and is

⁴ On the syntax and semantics of exceptions in UML 2.0 activity diagrams, see Störrle [2004c].

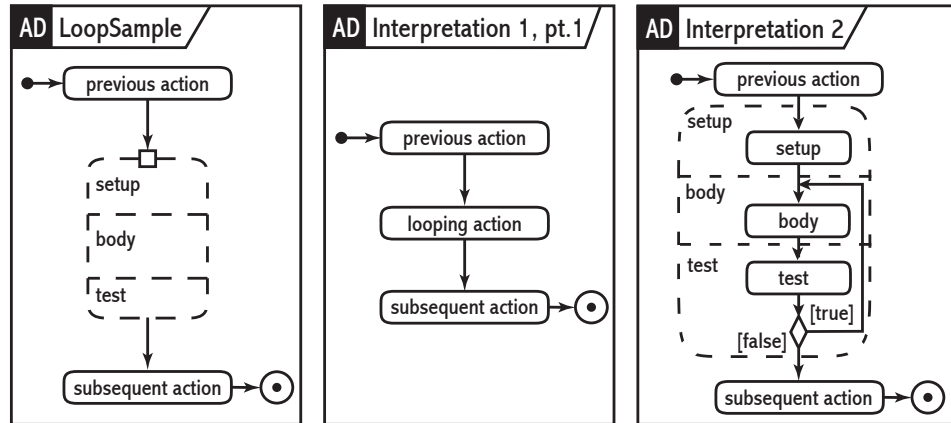


Fig. 4.10: LoopNodes in context: a fragment of an Activity containing a LoopNode (left), interpreting a LoopNode as an Action calling an Activity as described in Figure 4.9 (middle), and an alternative interpretation where the LoopNode is expanded in its context (right).

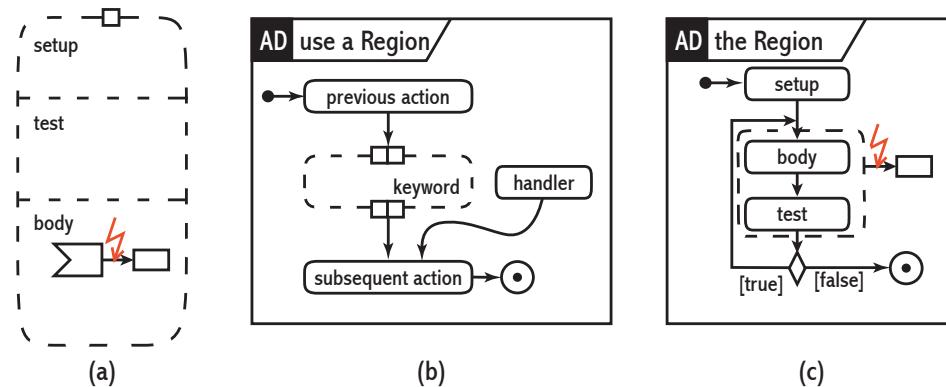


Fig. 4.11: Interactions between structured nodes and exceptions (a); the context of an Expansion-Region (b) may call upon an Activity defined, say, like (c). In this example, the raising scope of the exception (the InterruptibleActivityRegion) has been set to encompass both the body and the test section.

much less detailed than the descriptions of basic activities or LoopNodes and ConditionalNodes. Also, there is a fairly straightforward intuition for loops and conditionals in terms of their practical usage and value, even if this is not documented in the standard. For ExpansionRegions, however, such a pragmatic intuition is not obvious. So, to some degree, this section is speculative in exploring possible intentions of the standard. This is nevertheless a worthwhile effort, however, given the significance of the UML and MDA.

5.1 Intuition

ExpansionRegions are ActivityNodes that process collections of elements as a unit. The standard declares that when “an execution of an activity makes a token available to the input of an ExpansionRegion, [it] consumes the token and begins execution. The ExpansionRegion is executed once for each element in the collection” (cf. OMG [2003, p. 326]). In a way, thus, an ExpansionRegion can be viewed as a kind of map-function from one collection to another.

Now consider an example (the standard does not provide one). In Figure 5.12, the running example from Figure 2.1 has been extended to cope with orders where the attribute goods is not just a single item of type Good but a Good collection. For instance, instead of ordering a single item, a whole batch might be ordered by one customer. While the administrative processing like general ordering procedure, invoicing, payment and so on are the same as for orders consisting of just one item, the actual production must now be executed once for each item in the batch. This could be modeled explicitly, or one could use an ExpansionRegion for this purpose.

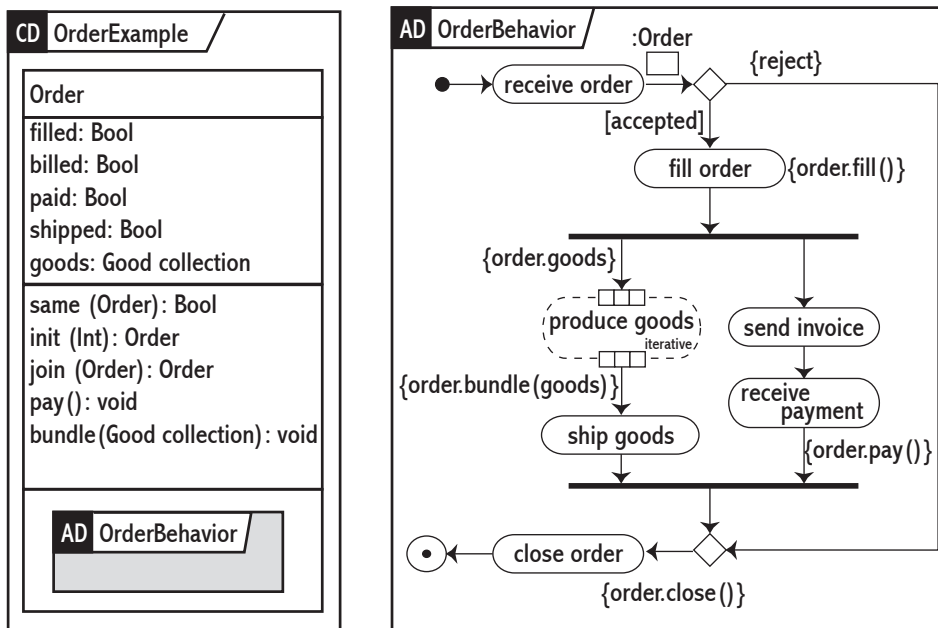


Fig. 5.12: An example for using an ExpansionRegion in an activity diagram.

Please note the keyword in the lower right corner of the ExpansionRegion in Figure 5.12. It indicates the mode of processing the arguments. It may be implemented in four different ways (not counting mixtures of these).

iterative that is, in a loop where each of the elements is treated in turn, but processing starts only on the complete set of arguments;

streaming is similar to iterative in that only one element is processed at once, but processing of the first element may start even though further elements have not yet arrived;

parallel meaning that all elements are processed in lockstep, starting, proceeding, and ending together, irrespective of the actual time needed to process individual elements, and so possibly creating idle times;

concurrent is similar to parallel in that all arguments are treated potentially at the same time, but independent of each other rather than in parallel.

Following the UML standard, it is possible to specify three of these variants by setting the mode-attribute of `ExpansionRegion`. It may carry the values `iterative`, `stream`, and `concurrent`. Beware of the last mode setting, however: due to a spate of printing mistakes in the standard, the word `parallel` is used in all but one place instead of the word `concurrent`. The behavior explained is definitely concurrency, though, not parallelism: “*the execution may happen in parallel, or overlapping in time, but they are not required to*” (cf. OMG [2003, p. 326]). To achieve parallelism, the standard would have to specify the execution as “*parallel in lockstep, beginning, proceeding, and ending simultaneously*” or similar.

At this point, an important feature of Activities is encountered. The standard states that “*the expansion region is executed once for each element in the collection (or once per element position, if there are **multiple** collections).*” (cf. OMG [2003, p. 326, emphasis added]) This seems to imply that an Activity is a kind of dataflow-computer with different parameters flowing through it—similar to a Petri-net, in fact⁵, and not like an individual run.

Unlike simple `StructuredActivityNodes`, `ExpansionRegions` can not be explained easily in terms of more elementary constructs. Thus, a semantics should be defined in terms of a more fundamental formalism, i.e., Petri-nets. In addition to the reasons given above, there is another reason why `ExpansionRegions` can not be covered by elementary Petri-nets: if there are several computations going on at the same time, they must be isolated from each other to avoid unintended interactions. Thus, a macro-like expansion strategy, as has often been proposed for high-level Petri-nets, is not the best choice here. In Störrle [2004b], this problem has already been solved silently by the very definition of procedural Petri-nets.

So, using the procedure call semantics we already discussed for `LoopNodes` and `ExpansionRegions`, this problem is avoided altogether (details on the formalism are found in Störrle [2004c]). Thus, similar to the treatment of `LoopNodes` as proposed in Figure 4.10 (middle), `ExpansionRegions` should be translated as refined Actions (see Figure 4.11 b and c), where various calls to the same refinement transitions executes in its own state space. Depending on the mode of the `ExpansionRegion`, different refinement nets must be used (see Figure 5.13). What these refinement nets look like will be explained in the following sections for each kind of `ExpansionRegion`.

⁵ Also, this raises again the question how well activity diagrams are suited for, say, workflow modeling, and whether Activities represent a workflow type/schema or an instance.

5.2 Iterative ExpansionRegions

In an iterative ExpansionRegion, “*the executions of the region must happen in sequence, with one finishing before another can begin. [...] Subsequent iterations start when the previous iteration is completed. During each of these cases, one element of the collection is made available to the execution of the region as a token during each execution of the region.*” (cf. OMG [2003, p. 326]). This mode is treated semantically as shown in Figure 5.13 (left). While similar to a loop node, an iterative ExpansionRegion does not define any order in which the items are processed.

At this point, a small digression concerning the tool used is in place. The nets in Figures 5.13 and 5.14 have been drawn, simulated, and analysed using CPN Toolset (see CPN Tools Team [2004]). Therefore, the inscriptions are Standard ML code (the programming language used for inscriptions in the CPN Toolset, cf. Paulson [1991]), with some special conventions. For instance, E is the color of plain tokens (“*black dot tokens*”, that is, the regular Petri-net tokens), and the only value of this type is e. Multisets are represented like `1‘true++2‘false`, which means: one token of value `true` and two tokens of value `false`. Tuples are written as `(x,y)`, and lists as `(head::tail)`.

Returning to iterative ExpansionRegions, the collections there have been implemented as lists, simply for convenience. Other implementations (in particular multisets) are of course possible. In the net, the region is represented by a transition. It may be refined to call another net in the sense of procedural Petri-nets (cf. Kiehn [1989], Störrle [2004b]). The region may start processing the first element of the collection right away by taking it from the list and, after processing it, adding it to a result list. When all elements have been processed, the Input collection is depleted (i.e. has become the empty list), and the resulting list is reversed to achieve the original order again (this last step may be omitted of course, if the collection is unordered).

5.3 Concurrent ExpansionRegions

In a concurrent ExpansionRegion, “*the execution may happen in parallel, or overlapping in time, but they are not required to.*” (cf. OMG [2003, p. 326]). This mode is treated semantically as shown in Figure 5.13 (right). There, the collection is first split up into its elements. The number of elements is tracked in the place Counter. Then, each element may be processed by the Region. Observe, that a transition may fire concurrently to itself as often, as there are tokens activating it. As soon as the first results are produced, they may be collected again by the join collection transition. If all results are processed, the Counter has been decreased to zero, and the output may be produced.

Note that it would be not easy at all to realize a truly lockstep-parallel execution mode of an ExpansionRegion, since then it is not sufficient to use a procedure call or macro mechanism. Instead, it would be necessary to resort to a kind of net folding.

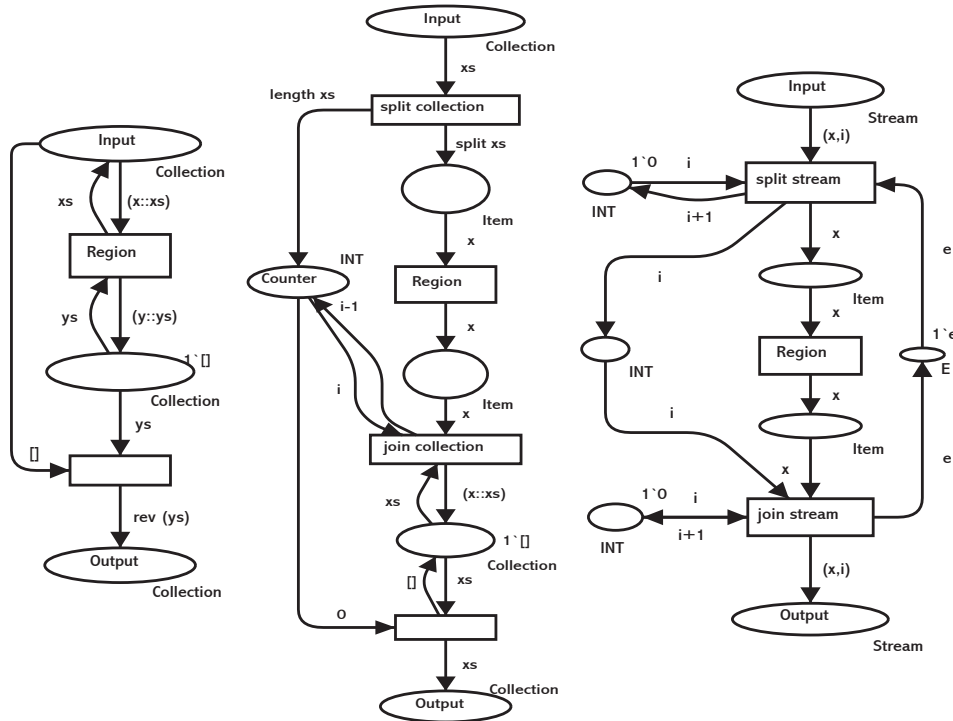


Fig. 5.13: Semantics of ExpansionRegions: iterative mode (left), streaming mode (middle), and concurrent mode (right).

5.4 Streaming ExpansionRegions

In an ExpansionRegion with mode stream, “there is a single execution of the region, but its input place receives a stream of elements from the collection. The values in the input collection are extracted and placed into the execution of the expansion region as a stream [...]. Such a region must handle streams properly or it is ill defined. When the execution of the entire stream is complete, any output streams are assembled into collections of the same kinds as the inputs.” (cf. OMG [2003, p. 326]). Thus, at any given time during the stream processing, some elements of a stream may have been processed already, some may be being processed in the very instant, and some may still be awaiting processing. This is in contrast to other collection-valued ExpansionRegions, where all elements of a collection must be present *before* the processing starts, and all elements of the collection must be processed *before* the Region is terminated.

Starting with the simple case of an individual stream, this behavior may be captured by the net shown in Figure 5.13 (middle). Collections may be represented by tagging the elements with sequence numbers, i.e. $\text{Element} \times \text{SequenceNumber}$.

First of all, the stream is split into sequence numbers and elements proper. While

the element is processed by the Region, the sequence number is passed by. The complement place ensures proper synchronisation.

The mappings provided in this and the two previous sections may be validated again using CPN Toolset (see Figure 5.14).

6. Conclusion

6.1 Summary

In this paper, the concepts related to `StructuredActivityNodes` are examined. `LoopNodes` and `ConditionalNodes` may be defined as syntactic sugar. `ExpansionRegions` may be defined by mappings to high-level Petri-nets. Some problems concerning the concrete and abstract syntax and the semantics have been uncovered in the standard (concrete syntax of `Conditional-` and `LoopNodes`, scope of exceptions, single/multiple inputs), and some possible solutions have been proposed.

There have been several proposals for semantics of activity diagrams, but most of these aim at UML 1.x. For UML 2.0, there are only Barros and Gomes [2003] and Störrle [2004b, 2004a, 2004c] dealing with the semantics, and Bock [2003a, 2003b, 2003c, 2004a, 2004b] dealing with some of constructs on an intuitive level.

6.2 Related work

There are two kinds of related work that are to be considered. On the one hand, there are predecessors and relatives of activity diagrams, that is, all the other notations that server similar purposes. Table 6.15 shows a comparison of the expressiveness of some of these notations.

- **UML** The most obvious comparison is of course between the activity diagrams in the various UML versions (cf. OMG [1998] and OMG [2003] vs. OMG [2003]). While the basic constructs remain more or less unchanged, version 2.0 adds several powerful features, including exceptions, arbitrary fork/join-conditions, `LoopNodes`, `ConditionalNodes`, and different kinds of `ExpansionRegions`.
- **formal methods** The earliest notations for processes are P/T-nets (see Baumgarten [1996], Murata [1989]). While exhibiting desirable and interesting theoretical properties, they have never quite reached industry. High level nets like colored Petri-nets (CPNs, see Jensen [1992]) have increased expressive power, but have not caught on, despite a number of successful case studies. The same is true for other formal methods for concurrency like Process algebras (e.g., CCS, see Milner [1989]). They have the additional drawback of being a purely textual notation, thus lacking visual appeal.
- **business processes** In business process modeling, (extended) Event-Process-Chains (eEPC) are very popular (cf. Keller *et al.* [1991], Scheer [1995]), in particular in conjunction with ERP packages such as SAP R/3. eEPCs and similar notations are now in widespread use, despite their limited expressiveness, rather restricted tool-support, and the persistent lack of adequate formal foundations.

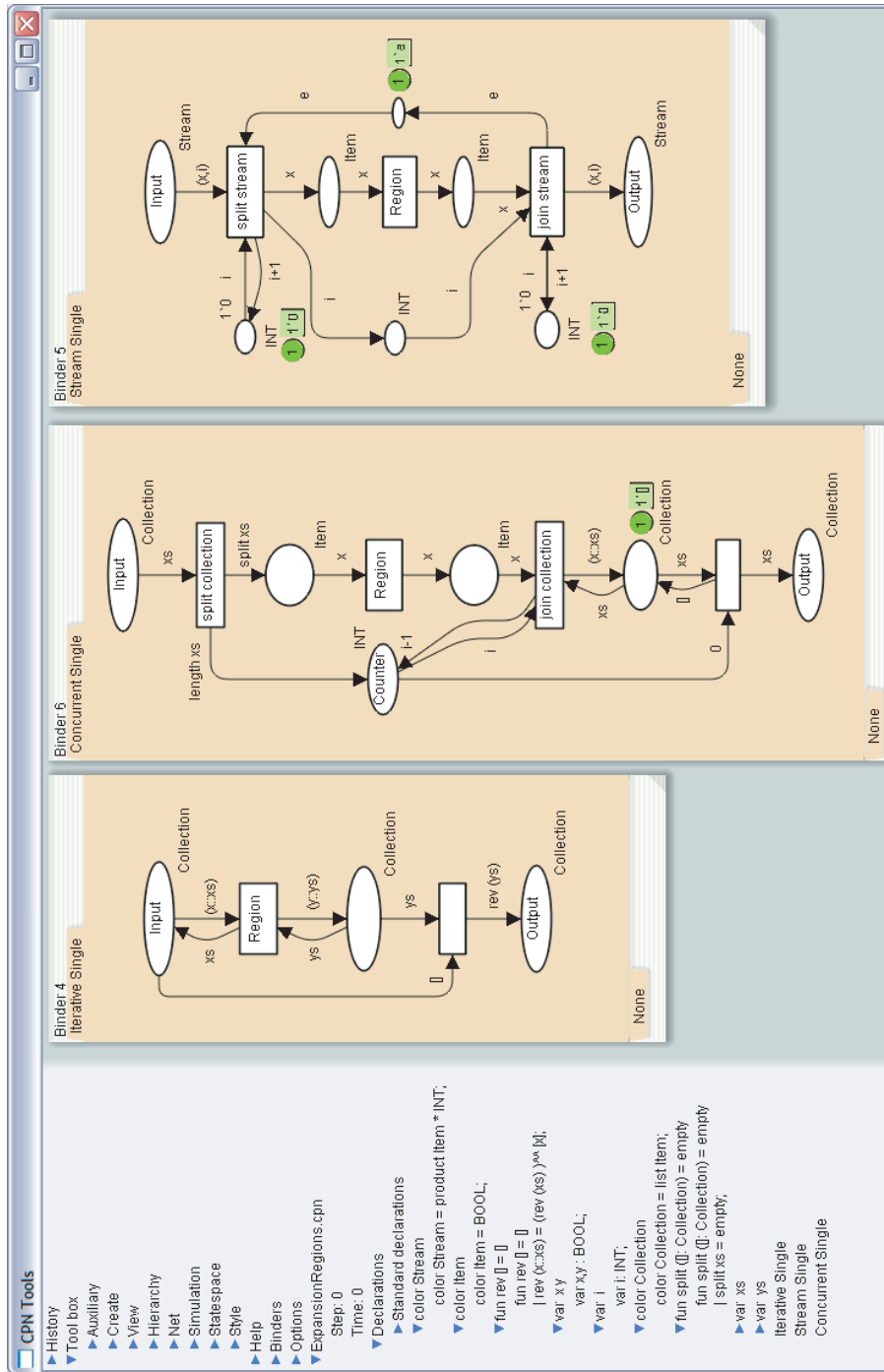


Fig. 5.14: Validation of semantic mappings for ExpansionRegions using CPN Toolset.

Notation	Control-flow capabilities					Abstraction capabilities			Data-flow capabilities		
	and/or/xor forks/joins	irregular nesting	exceptions	trans-actions	macros	conditional nodes	loop nodes	basic data	stream processing	collection values	
Name	√	-	-	-	(√)	-	-	√	-	-	
ADs 1.x (UML 1.x activity diagrams)	√ ⁶	√	-	-	√ ²	√	√	√	√	√	
ADs 2.0 (UML 2.0 activity diagrams)	√	√	-	-	√	-	(√)	-	-	-	
CCS (Calculus of Communicating Systems)	√	√	-	-	-	-	-	-	-	-	
P/T-nets (Place/Transition-nets)	√	√	-	-	-	-	-	-	-	-	
CPNs (colored Petri-nets)	√ ⁶	√	-	-	(√)	-	-	√	√	√	
eEPC (extended Event-Process-Chains)	√	-	-	-	√	-	-	(√) ²	-	-	
DFD (SADT Data-Flow Diagrams)	(-) ³	-	-	-	√	-	-	(√) ²	-	-	
IDEF-3 (Process Description Capture Method)	(-) ³	-	-	-	√	-	-	(√) ²	-	-	
BPEL4WS, v.1.1 ⁶ (Business Process Execution Language for Web-Services)	√	-	-	-	(√)	-	-	(√)	-	-	

¹ There is a straightforward extension to CCS that does provides value passing.

² Data-flow is treated only implicitly, i.e., as a consequence of control-flow.

³ Concurrency is explicitly modeled, but may occur.

⁴ Version 2.0 does not differ much from version 1.1.

⁵ Arbitrary expressions are allowed.

⁶ Arbitrary logical expressions are allowed.

Fig. 6.15: Comparing notations for model processes.

- **general purpose** Another notation with a similar scope, if slightly more oriented towards IT applications rather than the domain as such are data-flow diagrams (DFD, see Gane and Sarson [1979]) and IDEF-3 (see National Institute of Standards and Technologies [1993]). Both of these originate with the structured methods of the 70's and 80's, and provide by and large similar expressiveness as UML 1.x activity diagrams.
- **Workflows** On an even more detailed level, there are workflow description languages that feed into workflow engines. As one representative, consider BPEL4WS (see Andrews *et al.* [5 May 2003]). Like CCS, BPEL4WS lacks an adequate graphical representation. It is conceivable, though, that activity diagrams be used as a front-end for BPEL4WS.

Obviously, UML activity diagrams offer several new concepts and notations that are not present in any of the traditional business process modeling languages. It has been argued that such a rich language is difficult to handle, and too powerful for many tasks or users. However, UML activity diagrams are intended as a notation for modeling processes of all kinds, underlining the general idea of a *unified* modeling language, i.e., one for all purposes and users. In practical applications, thus, the available expressive means may be tailored to fit the respective users. For instance, concepts like exceptions might be disallowed in an organisation for use in modeling business use cases.

On the other hand, there is a body of work concerning the semantics of UML activity diagrams. Since the UML standard has been written from scratch as far as activity diagrams are concerned, most of the previous work examining UML activity diagrams (see Allweyer and Loos [1998], Apvrille *et al.* [2001], Bolton and Davies [2000], Bolton and Davies [2000], Börger *et al.* [2000], Dumas and ter Hofstede [2001], Eshuis [2002], Eshuis and Wieringa [2001], Eshuis and Wieringa [2001], Eshuis and Wieringa [2001], Eshuis and Wieringa [2002], Eshuis and Wieringa [2003], Gehrke *et al.* [1998], Li *et al.* [2001], Petriu and Sun [2000], Pinheiro da Silva [2001], Rodrigues [2000]) has become obsolete. See Figure 6.16 for a comparison.

In particular, structured nodes which have not been there in the UML 1.5 have not been addressed so far. Also, it seems that so far, only very little has been published on the UML 2.0 activity diagrams: Barros and Gomes [2003] examines expansions and streaming in a intuitive way, focusing on shared input pins and some aspects of streaming.

In a series of articles Bock [2003a, 2003b, 2003c, 2004a, 2004b], one of the main figures behind Activities in the UML 2.0 has tried to supplement the standard by a more elaborate description, still lacking a formal semantics, though. While rather instructive in many respects, important aspects are not covered, in particular, ExpansionRegions.

In a series of articles Störrle [2004b, 2004a, 2004c] I provide formal definitions of the semantics of control-flow, procedure call, data-flow, and exceptions in UML 2.0 Activities, respectively. This paper builds on the latter.

The most up-to-date account of the state of the discussion on the UML in general is found in the OMG's Issues-Database OMG [2004]. Currently, it seems as if the

authors, references	UML version	semantic domain	control flow	data flow	hierarchy	exceptions	streaming	structured nodes	rigor
Allweyer and Loos [1998]	0.9	-	wf	√	-	-	-	-	low
Apvrille <i>et al.</i> [2001]	1.x	LOTOS	wf	-	-	-	-	-	medium
Böger <i>et al.</i> [2000]	1.x	ASM	wf	-	√	-	-	-	medium
Bolton and Davies [2000, 2000]	1.3	CSP	wf	-	-	-	-	-	low
Eshuis [2002], Eshuis and Wieringa [2001]	1.x	algorithm	wf, nwf	-	-	-	-	-	high
Eshuis and Wieringa [2001, 2001]	1.x	LTS	wf, nwf	-	-	-	-	-	high
Gehrke <i>et al.</i> [1998]	1.0	PN	wf, nwf	(-)	-	-	-	-	medium
Pinheiro da Silva [2001]	1.x	LOTOS	wf, time	-	-	-	-	-	low
Rodrigues [2000]	1.x	FSP	wf	-	-	-	-	-	low
Li <i>et al.</i> [2001]	1.x	LTS	wf	(-)	-	-	-	-	high
Störkle [2004b]	2.0	PPN	wf, nwf	-	√	-	-	-	high
Störkle [2004a]	2.0	CPN	wf, nwf	√	-	-	-	-	high
Störkle [2004c]	2.0	ECPN	(wf, nwf)	(√)	(√)	√	-	-	medium

PN = simple P/T-Petri-nets
 PPN = procedural Petri-nets
 CPN = colored Petri-nets
 ECPN = exception colored Petri-nets
 LOTOS = Language of Temporal Ordering Specifications
 CSP = communicating sequential processes
 FSP = finite state processes
 LTS = labeled transition systems
 ASM = abstract state machines

Fig. 6.16: Comparative categorization of the previous work leading to this article (in column “control-flow”, wf means well-formed, and nwf means non well formed. The degree of rigor is approximate, where a completely formal definition is high rigor, examples and some formalism is medium, and mere text is low.

OMG actually makes a move towards formal semantics after all.

6.3 New questions

Building on this work, one may now turn to questions like what refinement and composition means for Activities, or how the new constructs work in the field, which subsets are suitable for which tasks, and whether there are any omissions. For instance, treatment of transactions is definitely an important issue, but is currently not supported.

Also, it would be interesting to apply traditional complexity and size-measures to activity diagrams. For instance, how does the replacement of explicit loops by loop nodes affect the cyclomatic complexity of an activity diagram.

Finally, the combination with other parts of the UML must be examined, in particular the relationship to Interactions and StateMachines, whose natural semantic domains must be related to Petri-nets. Also, examples soon become too complex for manual treatment, and so tool-support is needed. In Störrle [2004a], I have shown how standard Petri-net tools may be applied to verify properties of UML 2.0 activity diagrams, using a Petri-net semantics.

References

- ALLWEYER, THOMAS AND LOOS, PETER. 1998. Process Orientation in UML through Integration of Event-Driven Process Chains. In *International Workshop «UML» '98: Beyond the Notation*. Ecole Supérieure des Sciences Appliquées pour l'Ingénieur—Mulhouse, Université de Haute-Alsace, 183–193.
- ANDREWS, TONY, CURBERA, FRANCISCO, DHOLAKIA, HITESH, GOLAND, YARON, KLEIN, JOHANNES, LEY-MANN, FRANK, LIU, KEVIN, ROLLER, DIETER, SMITH, DOUG, THATTE, SATISH, TRICKOVIC, IVANA, AND WEERAWARANA, SANJIVA. 5 May 2003. Business Process Execution Language for Web Services (v1.1).
- APVRILLE, LUDOVIC, DE SAQUI-SANNES, PIERRE, LOHR, C., SÉNAC, PATRICK, AND COURTIAT, JEAN-PAUL. 2001. A New UML Profile for Real-Time System Formal Design and Validation. In *Proc. 4th Intl. Conf. on the Unified Modeling Language («UML» 2001)*, Number 2185 in LNCS. Springer Verlag, 287–301.
- BARROS, JOÃO P. AND GOMES, LUÍS. 2003. Actions as Activities as Petri nets. In *Proc. Ws. Critical Systems Development with UML*, 129–135.
- BAUMGARTEN, BERND. 1996. *Petri-Netze. Grundlagen und Anwendungen..* Spektrum Akademischer Verlag, Heidelberg.
- BOCK, CONRAD. 2003a. UML 2 Activity and Action Models. *J. Object Technology* 2, 4 (July/August), 43–53.
- BOCK, CONRAD. 2003b. UML 2 Activity and Action Models: Actions. *J. Object Technology* 2, 5 (September/October), 41–56.
- BOCK, CONRAD. 2003c. UML 2 Activity and Action Models: Control Nodes. *J. Object Technology* 2, 6 (November/December), 7–22.
- BOCK, CONRAD. 2004a. UML 2 Activity and Action Models: Object Nodes. *J. Object Technology* 3, 1 (January/February), 27–41.
- BOCK, CONRAD. 2004b. UML 2 Activity and Action Models: Partitions. *J. Object Technology* 3, 7 (July/August), 37–56.
- BOLTON, CHRISTIE AND DAVIES, JIM. 2000. Activity graphs and processes. In *Proc. Intl. Conf. Integrated Formal Methods (IFM)*, LNCS. Springer Verlag.
- BOLTON, CHRISTIE AND DAVIES, JIM. 2000. On giving a behavioural semantics to activity graphs. In *Proc. Intl. Ws. Dynamic Behavior in UML Models: Semantic Questions. Technical Report No. 0006 of the Ludwig-Maximilians-Universität, München, Inst. f. Informatik*, 17–22.

- BÖRGER, EGON, CAVARRA, ALESSANDRA, AND RICCOBENE, ELVINIA. 2000. An ASM Semantics for UML Activity Diagrams. In *Proc. 8th Intl. Conf. Algebraic Methodology and Software Technology (AMAST)*, Number 1816 in LNCS. Springer Verlag, 293–308.
- CPN TOOLS TEAM. 2004. CPN Tools Manual. Tech. report, Univ. of Aarhus.
- DUMAS, MARLON AND TER HOFSTEDE, ARTHUR H.M. 2001. UML Activity Diagrams as a Workflow Specification Language. In *Proc. 4th Intl. Conf. on the Unified Modeling Language (<<UML>> 2001)*, Number 2185 in LNCS. Springer Verlag, 76–90.
- ESHUIS, HENRIK. 2002. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, CTIT, U. Twente.
- ESHUIS, RIK AND WIERINGA, ROEL. 2001. A formal semantics for UML Activity Diagrams - Formalising workflow models. Tech. Report CTIT-01-04, U. Twente, Dept. of Computer Science.
- ESHUIS, RIK AND WIERINGA, ROEL. 2001. A Real-Time Execution Semantics for UML Activity Diagrams. In *Proc. 4th Intl. Conf. Fundamental approaches to software engineering (FASE)*, Number 2029 in LNCS. Springer Verlag, 76–90.
- ESHUIS, RIK AND WIERINGA, ROEL. 2001. An Execution Algorithm for UML Activity Graphs. In *Proc. 4th Intl. Conf. on the Unified Modeling Language (<<UML>> 2001)*, Number 2185 in LNCS. Springer Verlag, 47–61.
- ESHUIS, RIK AND WIERINGA, ROEL. 2002. Verification support for workflow design with UML activity graphs. In *Proc. 24th Intl. Conf. on Software Engineering (ICSE)*. IEEE, 166–176.
- ESHUIS, RIK AND WIERINGA, ROEL. 2003. Comparing Petri Net and Activity Diagram Variants for Workflow Modelling - A Quest for Reactive Petri Nets. In *Petri Net Technology for Communication-Based Systems*. DFG Research Group “Petri Net Technology”, 321–351.
- GANE, CHRIS AND SARSON, TRISH. 1979. *Structured Systems Analysis: Tools and Techniques*. Prentice Hall.
- GEHRKE, THOMAS, GOLTZ, URSULA, AND WEHRHEIM, HEIKE. 1998. The Dynamic Models of UML: Towards a Semantics and its Application in the Development Process. Tech. Report 11/98, Institut für Informatik, Universität Hildesheim.
- GOGOLLA, MARTIN AND KOBRYN, CHRIS, EDITORS. 2001. *Proc. 4th Intl. Conf. on the Unified Modeling Language (<<UML>> 2001)*, Number 2185 in LNCS. Springer Verlag.
- JENSEN, KURT. 1992. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. I*. EATCS Monographs on Theoretical Computer Science. Springer Verlag.
- KELLER, G., NITGENS, MARKUS, AND SCHEER, AUGUST-WILHELM. 1991. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerte Prozessketten (EPK). Tech. Report 089, Institut für Wirtschaftsinformatik, Uni Saarbrücken.
- KIEHN, ASTRID. 1989. *A Structuring Mechanism for Petri Nets*. Dissertation, TU München.
- LI, XUANDONG, CUI, MENG, PEI, YU, JIANHUA, ZHAO, AND GUOLIANG, ZHENG. 2001. Timing Analysis of UML Activity Diagrams. In *Proc. 4th Intl. Conf. on the Unified Modeling Language (<<UML>> 2001)*, Number 2185 in LNCS. Springer Verlag, 62–75.
- MILNER, ROBIN. 1989. *Communication and Concurrency*. Prentice Hall.
- MURATA, TADAO. 1989. Petri Nets: Properties, Analysis and Applications. *Proc. IEEE* 77, (April), 541–580.
- NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGIES. 1993. Integration Definition for Function Modeling. Tech. report, Computer Systems Laboratory, NIST.
- OMG. 1998. OMG Unified Modeling Language Specification (version 1.3). Tech. report, Object Management Group.
- OMG. 2003. OMG Unified Modeling Language Specification (adopted formal specification, version 1.5). Tech. report, Object Management Group.
- OMG. 2003. OMG Unified Modeling Language: Superstructure (final adopted spec, version 2.0). Tech. report, Object Management Group.
- OMG. 2004. The OMG UML Issues Database. Tech. report, Object Management Group.
- PAULSON, LAWRENCE C. 1991. *ML for the Working Programmer*. Cambridge University Press.
- PETRIU, DORINA C. AND SUN, YIMEI. 2000. Consistent Behaviour Representation in Activity and Sequence Diagrams. In *Proc. 3rd Intl. Conf. <<UML>> 2000—Advancing the Standard*, Number 1939 in LNCS. Springer Verlag, 369–382.
- PINHEIRO DA SILVA, PAULO. 2001. A proposal for a LOTOS-based semantics for UML. Tech. Report UMCS-01-06-1, Dept. of Computer Science, U. Manchester.

- REGGIO, GIANNA, KNAPP, ALEXANDER, RUMPE, BERNHARD, SELIC, BRAN, AND WIERINGA, ROEL, EDITORS. 2000. *Proc. Intl. Ws. Dynamic Behavior in UML Models: Semantic Questions. Technical Report No. 0006 of the Ludwig-Maximilians-Universität, München, Inst. f. Informatik.*
- RODRIGUES, ROBERTO W.S. 2000. Formalising UML Activity Diagrams using Finite State Processes. In *Proc. Intl. Ws. Dynamic Behavior in UML Models: Semantic Questions. Technical Report No. 0006 of the Ludwig-Maximilians-Universität, München, Inst. f. Informatik*, 92–98.
- SCHEER, AUGUST-WILHELM. 1995. *Business Process Engineering. Reference Models for Industrial Enterprises*. Springer Verlag.
- STÖRRLE, HARALD. 2003. Semantics of Interactions in UML 2.0. In *Human Centric Computing Languages and Environments*. IEEE Computer Society, 129–136.
- STÖRRLE, HARALD. 2004a. Semantics and Verification of Data-Flow in UML 2.0 Activities. In *Proc. Intl. Ws. on Visual Languages and Formal Methods (VLFM'04)*. IEEE Press, 38–52.
- STÖRRLE, HARALD. 2004b. Semantics of Control-Flow in UML 2.0 Activities. In *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Springer Verlag, 235–242.
- STÖRRLE, HARALD. 2004c. Semantics of Exceptions in UML 2.0 Activities. Tech. Report 0403, Ludwig-Maximilians-Universität München, Institut für Informatik.
- STÖRRLE, HARALD. 2004d. Semantics of Expansion Nodes in UML 2.0 Activities. In *Proc. 2nd Nordic Ws. on UML, Modeling, Methods and Tools (NWUML'04)*.