

# Semantics of UML 2.0 Activities with Data-Flow

Harald Störrle

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

**Abstract.** One of the major improvements of UML 2.0 over UML 1.5 is the reengineering of Activity Diagrams. It is claimed in the standard that they now have a Petri-net like meaning. In this paper, this claim is examined by defining a denotational semantics for Activities based on Colored Petri-nets. The definition closely following the UML 2.0 standard. It covers flat control-flow, and data-flow, but excludes exception-handling, expansion-regions, and procedure-calling. Going along, several points are raised that require clarification in the standard.

**Keywords:** UML 2.0, Activity Diagrams, denotational semantics, data-flow, modeling of web-services, workflows, and service-oriented architectures

## 1 Introduction

### 1.1 Motivation and goal

Modeling business processes and workflows is an important area in software engineering, and, given that it typically occurs very early in a project, it is one of those areas where model-driven approaches definitely have a competitive edge over code-driven approaches. Activity Diagrams are widely considered as appropriate for this task, and similarly, they are the natural choice (within the UML), when it comes to modeling web-services, and system-level behaviors.

Compared to UML 1.5, the concrete syntax of Activity Diagrams in UML 2.0 has remained mostly the same, as far as flat control-flow is concerned. Everything else, however, has changed dramatically. The changes affect the concrete syntax of data-flows, all of the abstract syntax, and, particularly, the semantics: while in UML 1.5, Activity Diagrams have been defined as a kind of State Machine Diagrams, there is now no such connection between the two: “*Activity replaces ActivityGraph in UML 1.5.*” (cf. [24, p. 292]) and “*Activities are redesigned to use a Petri-like semantics instead of state machines*” (cf. [24, p. 292]). But is the semantics really “*Petri-like*”? How far does the analogy reach: to which degree can Petri-nets actually be considered the semantics of Activity Diagrams?

### 1.2 Approach

In order to find out, we examine the UML 2.0 standard and try to define a formal semantics in terms of Petri-nets. Traditional P/T-nets, however, are not suitable since Activity Diagrams also feature procedure-call and data-flow facilities. While the issue

of procedure-calling has been addressed in [31] using the notion of procedural Petri-net systems, this paper turns towards the data-flow issue. There are several extensions to the basic Petri net model that are capable of modeling data-flow, generally subsumed under the title of “higher-order nets” [20], with [19, 16] probably being the best known dialects.

### 1.3 Related work

While there is a rather large body of work on UML 1.x Activity Diagrams, it seems that so far, hardly any work (except [3] and [31]) has been published on the UML 2.0 Activity Diagrams—and the UML standard has been written from scratch as far as Activity Diagrams are concerned. Yet, some considerations and ideas of previous works still apply, and so it is worthwhile looking at the previous work on UML 1.x Activity Diagrams, too. Here, there are four distinct categories of contributions.

First, there are “pragmatic” approaches that look into the pragmatics of Activity Diagrams, examining their usage either for comparing the expressive power of Activity Diagrams with that of (commercial) workflow description languages and workflow management systems (e.g. [8, 1, 14]), or for examining their methodological relationship to other diagram types of the UML (e.g. [27]). The main contribution of these approaches lies in exploring the potential of Activity Diagrams for certain purposes, and to interpret and develop the standard in such a way that the specific requirements of some particular purpose are better realized. While indeed discussing some semantic issues in doing so, the approaches in this category do not provide a semantics. Also, the new standard now defines the semantic model in mind—Petri nets—and lists the intended usage areas of Activities, namely “*procedural computations*”, “*workflows*”, and “*system level processes*” (cf. [24, p. 284]). Thus, considerations as to the appropriateness of particular semantic domains or about the modeling requirements of one way of modeling versus the other have now become obsolete.

Second, there are some approaches (e.g. [28]) that treat Activity Diagrams as a subclass of StateMachines, as declared by UML 1.x. This has always been a controversial issue, and has now disappeared from the standard.

Third, there are “operational” approaches, defining the meaning of Activity Diagrams “by interpreter”, that is, to give them a meaning in terms of some execution mechanism such as a (commercial) workflow execution system, or a more or less formal execution algorithm (cf. [12, 11]) or analysis procedure (cf. [21, 9, 30, 10, 13]).

Fourth, there are those contributions that define a formal semantics in the proper sense, that is, some kind of mapping from Activity Diagrams or ActivityGraphs into some formal domain, e.g. [2, 6, 5, 4, 9, 10, 15]. These approaches may be categorized along the following three axes:

**domain** the semantic formalism into which Activity Diagrams are mapped;

**rigor** the degree of formality employed in defining the mapping, ranging from a set of examples to a mathematical function;

**expressiveness** the degree of coverage of Activity Diagram notions that is mapped, i.e., control-flow, non well-formed control flow, data flow, and hierarchy (flat/macro-expansion-style vs. procedure call).

See Figure 1 for a direct comparison of the contributions of the last two categories.

authors, references	semantic style	semantic domain	expressiveness			rigor
			control flow	data flow	hierarchy	
Allweyer et al. [1]	by example	–	wf	√	√	low
Apvrille et al. [2]	by algorithm	LOTOS	wf	–	–	medium
Börger et al. [6]	denotational	ASM	wf	–	√	medium
Bolton & Davies [4, 5]	by compiler	CSP	wf	–	–	low
Eshuis & Wieringa [9, 10]	informal	algorithm	wf, nwf	–	–	high
Eshuis & Wieringa [12, 11]	by algorithm	LTS	wf, nwf	–	–	high
Gehrke et al. [15]	by example	PN	wf, nwf	(–)	–	medium
Rodrigues [30]	informal	FSP	wf	–	–	low
Li et al. [21]	by algorithm	LTS	wf	(–)	–	high
Störrle [31]	denotational	procedural PN	wf, nwf	–	√	high
this paper	denotational	colored PN	wf, nwf	√	–	high

**Fig. 1.** Comparative categorization of the previous work (wf means well-formed, and nwf means non well formed, other abbreviations explained in text).

Let’s look at some of the contributions of the fourth category in more detail. Gehrke et al. [15] also use Petri nets as their semantic domain, but interestingly, they use places to represent ActivityStates (Activities in UML 2.0), possibly misled by the passive sounding name in the UML 1.5 metamodel. Their work being focused on other issues, Activity Diagrams are treated only in passing and many interesting features are left out, including hierarchy and data-flow.

Börger et al. [6] use Abstract State Machines as their semantic domain, and this is the only other semantics that treats SubactivityStates, i.e. procedure calling of Activities. However, the mapping is only given by example, based on concrete syntax and excludes non well-formed control flow. And, of course, it is based on UML 1.3, not on UML 2.0.

Eshuis and Wieringa [9, 10] have published a spate of papers dealing with various aspects of Activity Diagrams, including a kind of operational semantics using labeled transition systems. Again, hierarchy and data-flow are left out. See Figure 1 for a comparison of the approaches mentioned.

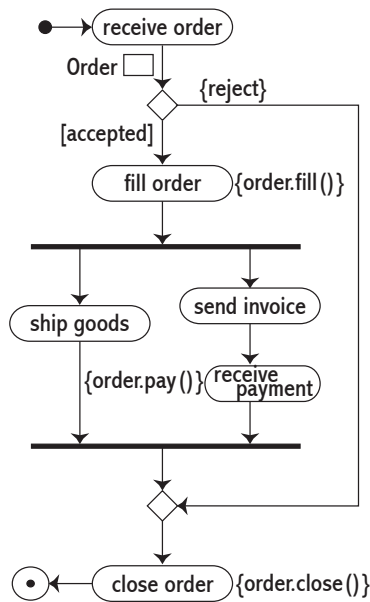
Surprisingly, all of the formal semantics have left out data-flow, possibly underestimating both the practical importance and the theoretical richness of this aspect. For practical purposes, modeling of data-flow is instrumental—industrial applications simply need this facility in all but the simplest settings. This is acknowledged by the fact that authors with a practical background (like [1]) include this aspect.<sup>1</sup>

<sup>1</sup> These authors also cover other aspects of practical value which, unfortunately, cannot be covered here due to lack of space.

The same is true for industrial workflow definition languages, and for languages for describing web-services (cf. [26, 35, 22]) even if they lack industrial perspective.

## 2 Activity Diagrams in UML 2.0

In this section, we discuss Activities in UML 2.0 with a particular emphasis on the differences to UML 1.x. The Activity Diagram shown in Figure 2 (abridged from the standard) will serve as a running example throughout this paper.



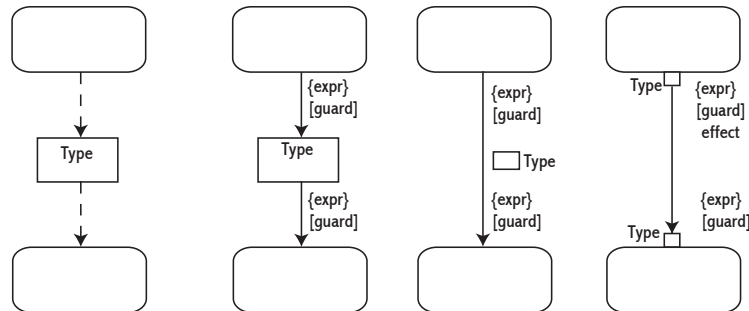
**Fig. 2.** A sample Activity Diagram with data-flow annotation, adapted from [24, p. 303, Fig. 219].

### 2.1 Concrete syntax

The concrete syntax of Activity Diagrams is changed only slightly with respect to control-flow, but has some interesting (and problematic) differences with respect to data-flow. One notable extension is the flexibility now provided by swim lanes, which are close to simulating a kind of use case maps (cf. [7]). It does not influence the behavior of an Activity, however, and may thus be ignored here. SubactivityStates have vanished, nesting now being accomplished by calling subordinate Activities from the

Actions that define the behavior of superordinate Activities. Control-flow is now denoted by ObjectNodes and ObjectFlows.

The standard allows three different notations for data-flows (cf. Figure 3). First, there is a notation similar to that of UML 1.5, where data-flows are specified explicitly. The only difference in UML 2.0 is that dashed arrows have been replaced by solid arrows.

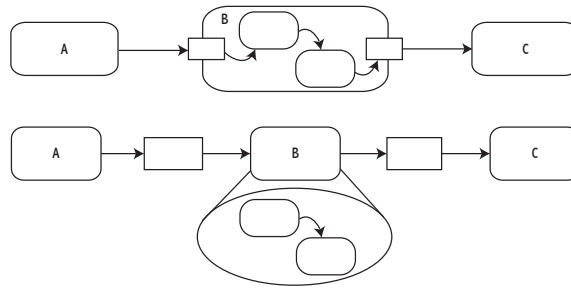


**Fig. 3.** Concrete syntax for data-flows: UML 1.5 notation (left), and in alternative equivalent UML 2.0 notations (all others), including “attached data-flow”-, and “pin”-notations (third and fourth).

Second, there is a simplified version that allows to attach a data-flow item to a control-flow edge, speaking in terms of visual representation. This notation is particularly convenient from a practical point of view since with this notation, it is very easy to first specify control-flows and then later to selectively add data-flows, leaving the control-flow untouched. In fact, there are similar procedures for many other parts of the UML, one can view this as give rise to a kind of incremental methodology.

It is not entirely clear though, what this notation really means. Two interpretations are possible. First, “attaching” an ObjectNode to an existing ActivityEdge could be interpreted as introducing the ObjectNode and two ObjectFlows to and from it (the ObjectFlows are not represented visually). Second, it could be interpreted as introducing the ObjectNode and *replace* the one ActivityEdge by two ObjectFlows. The first interpretation seems to be closer to the intuition, particularly to the incremental method of creating Activity Diagrams hinted at above. The second interpretation, on the other hand, would avoid “invisible” arcs. Either interpretation, however, only affects the transition from concrete to abstract syntax, but not the one from abstract syntax to semantics.

Third, data-flows may be specified using Pins, cf. Figure 3 (right). Pins are a subclass of ObjectNode, and act as as a kind of “parameter” of Activities (that may be called by the Actions that are executed in lieu of the ActivityNodes that have the Pins). The purpose of this notation becomes more understandable in the context of procedure calling, see Figure 4.



**Fig. 4.** Pins are are ObjectNodes for refinement.

The standard is not very explicit about the inscription language. In fact, there is not a single example of the concrete syntax, and references to the “action semantics” are scarce. So, we more or less had to make up an inscription language. It is fairly straightforward, and close to Standard ML [25] for practical reasons. The details are explained in the section on data-flow. The standard also does not specify timing annotations.

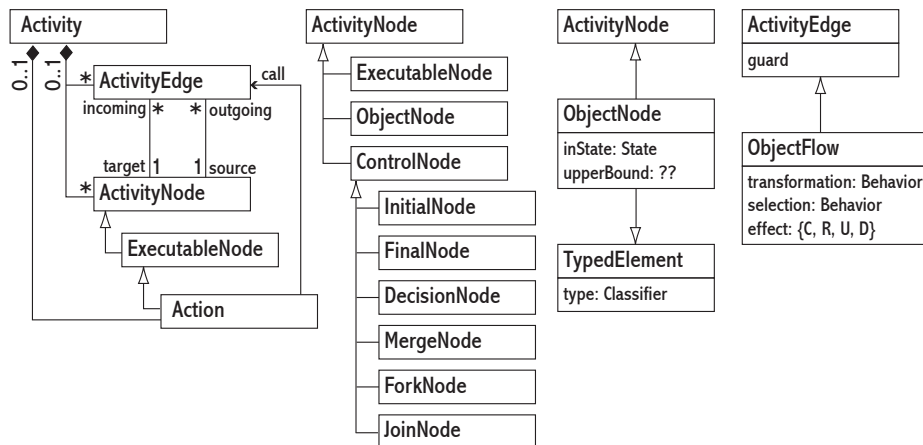
## 2.2 Abstract syntax

The metamodel for Activities has been redesigned from scratch in UML 2.0. The main concept underlying Activity Diagrams is now called Activity and “replaces Activity-Graph in UML 1.5.” (cf. [24, p. 292]). Activity is not a subclass of StateMachine any more, but is “redesigned to use a Petri-like semantics instead of state machines.” (cf. [24, p. 292]). The metamodel defines six levels increasing expressiveness. The first level (“BasicActivities”) already includes control flow and procedurally calling of subordinate Activities by ActivityNodes that are in fact Actions (see Figure 5), the second level (“IntermediateActivities”) introduces data flow. This paper is restricted to BasicActivities.

The basic two entities are Actions and Activities. While an Action “is the fundamental unit of executable functionality” (cf. [24, p. 280]), an activity provides “the coordinated sequencing of subordinate units whose individual elements are actions” (cf. [24, p. 280]). This coordination is captured as a graph of ActivityNodes connected by ActivityEdges (see figure 5). Data-flow is represented using ObjectNodes and ObjectFlows, which are subclasses of ActivityNodes and ActivityEdges, respectively. See Figure 5 for the portion of the metamodel relevant for Activity Diagrams. For all instances of metaclasses, the usual dot-notation is used to access the fields of the instances, i.e., to extract the state of a given ObjectNode  $o$ , we write  $o.inState$  and so on.

For convenience, we assume that an Activity is presented as a graph in the mathematical sense, i.e., in the form  $\langle ActivityNodes, ActivityEdges \rangle$ , where the *ActivityNodes* and *ActivityEdges* are again partitioned into the respective metaclasses. That is, *ActivityNodes* is really a tuple  $\langle EN, iN, fN, BN, CN, ON \rangle$  again, where:

$EN$  the set of ExecutableNodes (i.e. elementary Actions);



**Fig. 5.** A small portion of the UML 2.0 metamodel: Activities either have Actions or a graph of ActivityNodes and ActivityEdges (left); kinds of nodes and edges (right).

- $iN, fN$  the InitialNodes and FinalNodes (of which there may be only one);
- $BN$  the set of branch nodes, including both MergeNodes and DecisionNodes;
- $CN$  the set of concurrency nodes, subsuming ForkNodes, JoinNodes and ForkJoinNodes;
- $ON$  the set of ObjectNodes;

and *ActivityEdges* is a pair  $\langle AE, OF \rangle$ , where:

- $AE$  the set of plain ActivityEdges between ExecutableNodes and ControlNodes;
- $OF$  the set of ObjectFlows between ExecutableNodes and ControlNodes on the one hand, and ObjectNodes on the other.

So, all in all, Activities have the form  $\langle \langle EN, iN, fN, BN, CN, ON \rangle, \langle AE, OF \rangle \rangle$ .

### 2.3 Intuitive semantics

The semantics has changed even more than the abstract syntax: Activities now “use a Petri-like semantics instead of state machines.” (cf. [24, p. 263]). Unfortunately, the standard does not elaborate on this promising statement. So, the semantics defined in this paper is also an attempt to fill in the blanks.

## 3 Semantics of control-flow

In this section the formal semantics of basic control flow is defined, that is, sequencing, branching, and concurrency. In order to keep this semantics simple, we impose some restrictions on the concrete syntax. So, it is assumed that merging control flows is always

properly modeled by a MergeNode (see the DecisionNodes of the Activity Diagram in Figure 7). Procedure calling is ignored here—the treatment in [31] is orthogonal, and may thus be added ad lib. Also, connectors, and send, receive, and time events are omitted. We demand, that there are unique initial and final nodes in Activities. Finally, we require that all elements are named with globally unique names.

### 3.1 Semantic domain

The standard declares that “Activities are redesigned to use a Petri-like semantic” (cf. [24, p. 281]). Thus, we define the domain PN of Petri net as a tuple  $\langle P, T, A \rangle$  where  $P$ ,  $T$ , and  $A$  have the usual meanings (places, transitions, and flow arcs, see [23]).

### 3.2 Semantic mapping

For basic control-flow of Activities, the mapping is rather simple. Intuitively, ExecutableNodes become net transitions, ControlNodes become net places or small net fragments, and ActivityEdges become net arcs, possibly with auxiliary transitions or places. See Figure 6 for an intuitive account of the translation.

The formal semantics is also straightforward. Recall that the abstract syntax representation of an Activity has the structure  $\langle \langle EN, iN, fN, BN, CN, ON \rangle, \langle AE, OF \rangle \rangle$ . The translation for basic Activity Diagram thus is  $\llbracket \langle Nodes, Edges \rangle \rrbracket_{CF} = \langle P, T, A \rangle$  where

$$\begin{aligned} P &= \{iN, fN\} \cup BN \cup \{p_a \mid a \in Edges, \{a_1, a_2\} \cap (EN \cup CN) \neq \emptyset\}, \\ T &= EN \cup CN \cup \{t_a \mid a \in Edges, \{a_1, a_2\} \subseteq BN \cup \{iN, fN\}\}, \\ A &= \{\langle x_{\langle from, to \rangle}, to \rangle, \langle from, x_{\langle from, to \rangle} \rangle \mid \langle from, to \rangle \in Edges\}, \end{aligned}$$

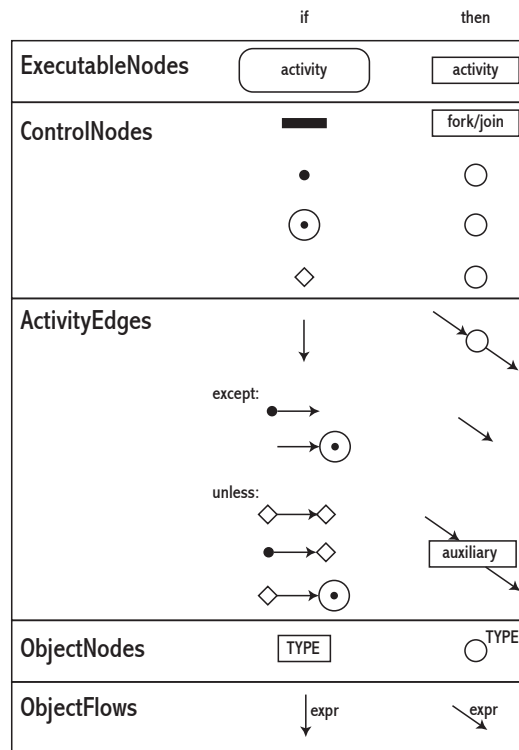
similar to the definition presented in [31]. Observe that the semantic function is indexed  $CF$  to indicate that this is the control-flow translation.

We use the shorthand  $a_1$  and  $a_2$  to denote the first and second element of a pair. Observe that ActivityEdges are used as indexes for names for some PN places—this might seem awkward at first, but it simplifies the definition. For the resulting Petri net, the tuple is treated as an atomic symbol, and just serves as an index to create a unique name.

For an example, reconsider the Activity from Figure 7 and its translation into a colored Petri-net shown in Figure 8. For the time being, ignore the inscription, and consider just the net-structure. A more detailed definition of the control-flow semantics with additional aspects is presented in [31].

## 4 Semantics of data-flow

As for the mapping for control-flow, there are also restrictions we impose for data-flow. First, we use one notation for data-flow only, namely the one with ObjectFlows attached to ActivityEdges (third variant in Figure 3). All other notations are considered as syntactic sugaring. For simplicity, we assume that there are no data-flows ActivityEdges from InitialNodes and to FinalNodes, though this would translate nicely into Petri-nets.



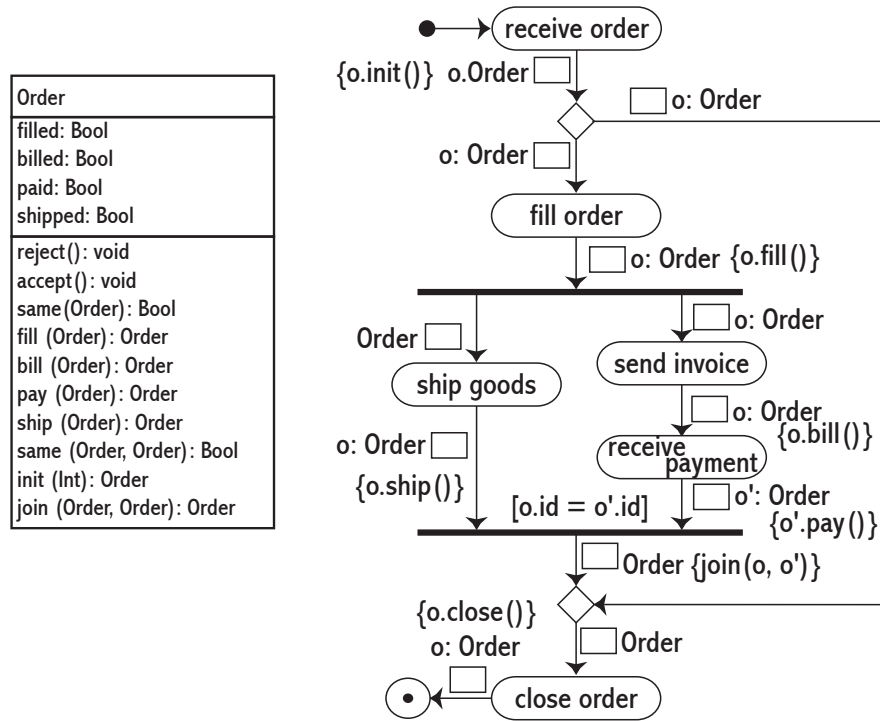
**Fig. 6.** The intuition of the semantic mapping for control- and data-flow of Activities.

Then, only those ObjectNodes and ObjectFlows are translated, that are actually present in an Activity. Implicit ObjectNodes and ObjectFlows that a human observer might add in his mind are not translated. In practical settings, one would rarely fill in all the details of an Activity Diagram, but expect a human to understand the intuition. Figure 2 shows an example for this: a human reader would simply gloss over the incomplete and sloppy annotation. For a formal semantics, this can not be tolerated, however. So, for the purpose of this paper, we assume that no elements are elided. The complete and correct example would thus be as shown in Figure 7.

Here, we have also added a minuscule class diagram that specifies a signature which for a concrete executable model will be turned into something like a Sigma-algebra for the inscription language later on.

#### 4.1 Semantic domain

The data-flow facilities of Activities may be represented by any dialect of high-level Petri nets [20, 19, 16]. For pragmatic reasons—availability of good tool support to name



**Fig. 7.** A more diligently specified version of the Activity Diagram presented in Figure 2.

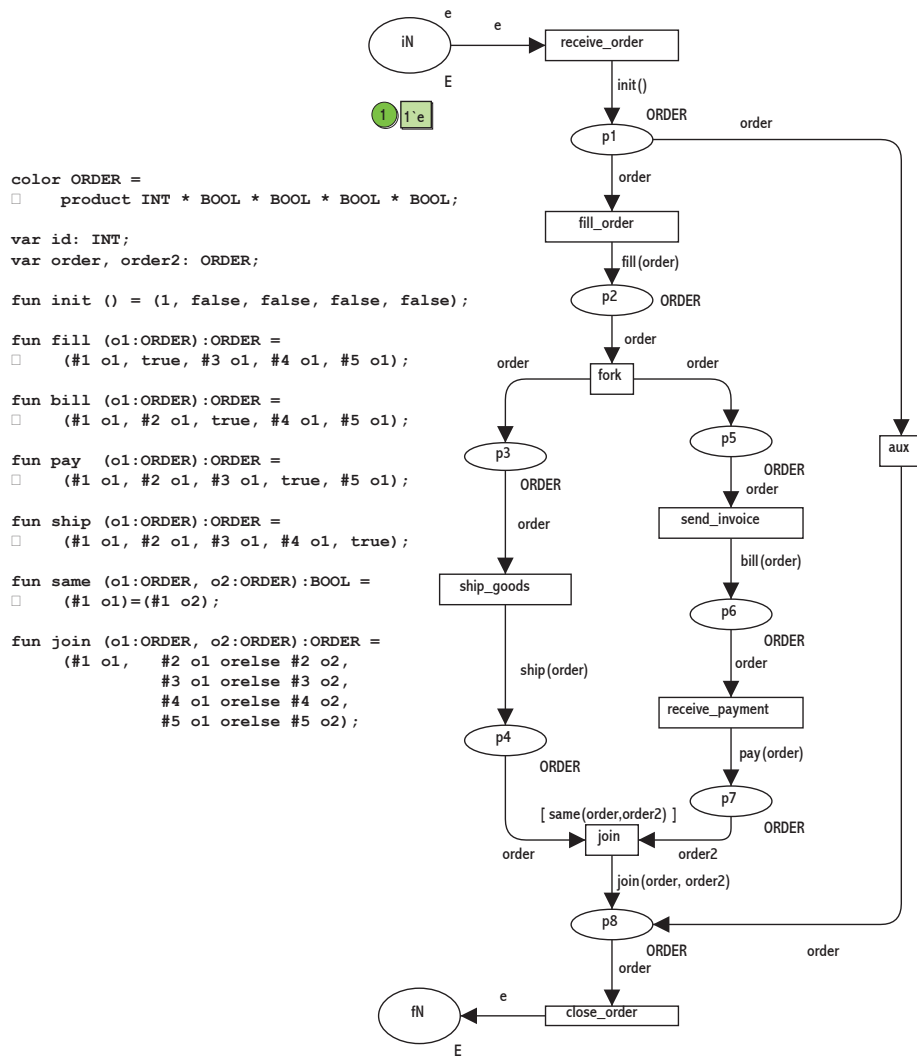
but one—colored Petri-nets (CPNs) are chosen as the semantic domain here (see Figure 8 for an example).

**Definition 41 (structure of colored Petri-nets)**

A tuple  $\langle N, SigAlg, color, guard, effect \rangle$  is a colored Petri-net (CPN), iff

- $N$  is a Petri net  $\langle P, T, F \rangle$  of places, transitions, and flow arcs;
- $SigAlg$  is a  $\Sigma$ -algebra  $\langle \Sigma, Op \rangle$  of sorts and operations;
- $color$  is a total function  $P \mapsto \Sigma$  assigning a type (“color”) to each place;
- $guard$  is a total function  $T \mapsto Expr$  assigning a boolean expression to each transition;
- $effect$  is a total function  $A \mapsto Expr$  assigning a expression to each arc, its type being the color of the place of the arc.

For convenience,  $color$ ,  $guard$ , and  $effect$  may be specified partially, with black-dot tokens as the default. That is, if  $color(p)$  is undefined, then  $color(p) = \text{TOKEN}$  is intended, and analogously for  $guard$  and  $effect$ . □



**Fig. 8.** The Petri net representing the Activity of the Activity Diagram in Figure 7 (right).

The definition of the behavior of CPNs is a little more complicated, as we now need to take into account the values of tokens and the meanings of operations on them. A marking of a CPN is multiset (or word) over  $\{\langle p, v \rangle \mid p \in P, v \in \text{color}(p)\}$ . As we lack the space for a complete definition, we have to make do with an example: consider Figure 9 for a sample run of the net of Figure 8, representing the Activity Diagram of Figure 7.

Formally, the Class Diagram shown in Figure 7 (left) maps into a signature  $\Sigma$ , and that is all we know about the inscriptions. At some point during refinement of the initial Activity Diagram, however, the modeler must provide more details. For simplicity, we assume that this is done in terms of code fragments that define the meaning of the sort- and operation-symbols of  $\Sigma$ . See the text to the upper left of the net in the example shown in Figure 8. Since we use the CPN Toolset, the programming language of these code fragments is Standard ML [25]. Observe that by convention, E is type of the traditional black-dot token. The net in Figure 8 is fully operational.

Observe, that the net-elements for CPNs are orthogonal to those of procedural petri nets. Thus, the semantics for data-flow defined here may be combined with procedure call semantics defined in [31].

## 4.2 Semantic mapping

Intuitively, the semantic mapping creates a net place for an ObjectNode, their type (a Classifier) becoming the place's color. ObjectFlows are simply net arcs, their guards are moved up- or downstream to the next net transition. The effects, selections and transformations remain at the resulting net arc. See Figure 6 for an intuitive account of the translation.

Now we may supplement the definition of  $\llbracket - \rrbracket_{CF}$  from above by the mapping  $\llbracket - \rrbracket_{DF}$  for translating the data-flow. It is defined as follows.

$$\llbracket \langle Nodes, Edges \rangle \rrbracket_{DF} = \langle N, SigAlg, color, guard, effect \rangle$$

where  $N_{CF} = \langle P_{CF}, T_{CF}, A_{CF} \rangle = \llbracket \langle Nodes, Edges \rangle \rrbracket_{CF}$  in

$$\begin{aligned} N &= \langle P_{CF} \cup ON, T_{CF}, A_{CF} \cup OF \rangle \\ SigAlg &= \langle \{o.type \mid o \in ON\}, \{a.transformation \mid a \in OF\} \rangle \\ color &= \{o \mapsto o.type \mid o \in ON\} \\ effect &= \{a \mapsto a.transformation \mid a \in OF\} \\ guard &= \{t \mapsto \bigwedge_{\langle a,t \rangle \in OF} a.selection \wedge \bigwedge_{\langle t,a \rangle \in OF} a.selection \mid a \in OF\}. \end{aligned}$$

Recall that  $Nodes = \langle EN, iN, fN, BN, CN, ON \rangle$  and  $Edges = \langle AE, OF \rangle$ .

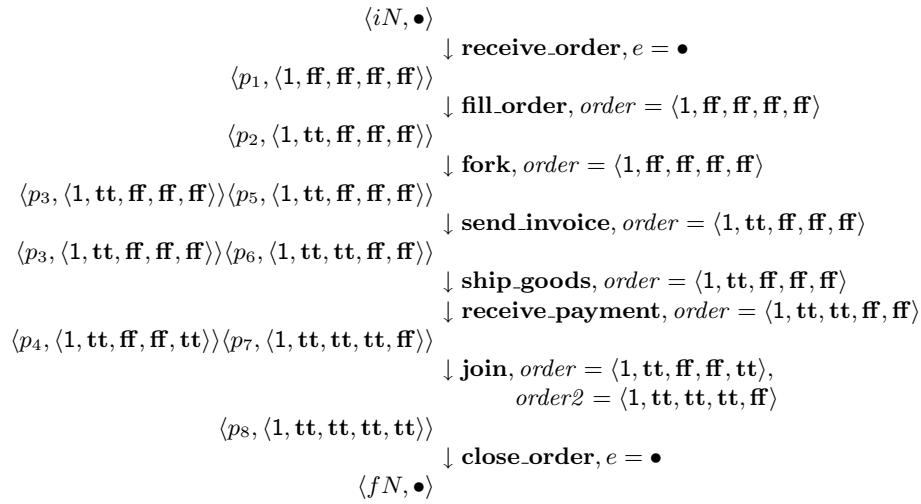
At this point, it is time to have a closer look at the inscription language. Keep in mind, that the standard does not specify a concrete syntax for the inscriptions. In Figure 7, there are basically three kinds of inscriptions. First, there are type declarations on ObjectNodes, some of which declare a variable (in the example simply called o) representing instances of the type that may reside in the ObjectNode. The types correspond directly to the colors of the respective net places, and the variables of the ObjectFlows adjacent to an ActivityNode constitute the name space for the Action the ActivityNode executes (assuming it is an ExecutableNode).

Second, there are effect functions in curly braces on ActivityEdges. All we can reasonably know about them is that they work on a given name space, changing the state of some of the objects in it, possibly augmenting or reducing the name space in doing so. The modeler must fill in the exact meaning of these functions, the effect expressions are simply handed down through the translation and mapped into effect expressions attached to the arcs going out of the transition representing the ActivityEdge.

Third, there are guard functions in square brackets on ActivityEdges. They simply access the name space, and may read states of the objects in it. Again, the exact meaning is up to the modeler, and the expressions are turned into guards (boolean expressions) over the variables defined by the arcs adjacent to the transition representing the ActivityEdge.

If  $o.upperBound$  is defined for an an ObjectNode  $o$ , it translates into a place capacity of the place representing  $o$  (canonical construction).

Initial and final markings for the CPN resulting from a translation could be constructed by restricting the corresponding places to color TOKEN, i.e.  $\langle iN, \bullet \rangle$  and  $\langle fN, \bullet \rangle$ , see Figures 8. A sample run of this net is shown in Figure 9.



**Fig. 9.** A run of the net in Figure 8, representing the Activity Diagram of Figure 7.

From a Petri-net point of view, the net in Figure 8 is very dull indeed, as it has only a single net process, and only three (terminal) traces. It would be very easy to increase the initial marking, thus allowing several Orders to flow through the system concurrently. The way the net is modeled, different Orders would be isolated by their identifiers (the first element in the tuple denoting the object state of instances of Order). In fact, the way they are defined now, Activities are closer to workflow instances than to workflow types.

Having larger initial markings would also allow to apply formal analysis and verification techniques on Activities (cf. [9, 10, 13]) in a natural way. In particular, time analysis (see e.g. [21]) and stochastic analysis, but the standard does not treat these issues. It is not obvious, how the concrete and abstract syntax are best extended to include such information in a satisfactory way.

The `JoinNode.joinSpec` defaults to “and”, and so `JoinNodes` are translated as regular transitions. Any exceptions from this rule (see e.g. [24, Fig. 264, p. 341]) must be specified explicitly, and may be translated into guards of the transition representing the `JoinNode`. Interestingly, there is no corresponding construct for `ForkNodes`, i.e., for splitting up sets of values to concurrent streams of tokens. This might be interesting in conjunction with streaming/buffering.

## 5 Conclusions

In this paper, a denotational semantics of Activities in UML 2.0 is defined based on colored Petri-nets. The semantics covers flat control-flow and data-flow, but not procedure call, exception handling and other advanced features. Since CPNs are orthogonal to procedural Petri-nets, the semantics for procedure calls of Activities as defined in [31] can be easily combined with the data-flow semantics presented here.

There have been several proposals for semantics of Activity Diagrams in UML 1.x so far, but none for UML 2.0. Some of proposals however interpreted the old standard rather liberally, and therefore still fit the new standard to some degree. These proposals, however, generally do not cover data-flow.

There are some questions raised by the results in this paper. First, behavioral descriptions must be aligned with static and functional descriptions, i.e. use cases, parts/ports (“architecture diagrams”), and classes. The same applies for inscriptions in various parts of the UML.

Second, the standard defines three fundamentally different, but tightly integrated views of behavior: `StateMachines`, `Activities`, and `Interactions`. Their formal semantics should also be compatible, but it is currently unclear, how this is best achieved. Some of the problems are:

- Each of the views has its own “natural” semantic domain, e.g. sequences for `Interactions`, Petri nets for `Activities`, and state machines for `StateMachines`. Bringing them together in a way that is meaningful for intuition and useful for practical purposes raises a number of issues.
- While most of the simpler parts of UML 2.0 `Interactions` are fairly well understood by now (cf. [33, 34]), there are definitely open questions both for `Interactions` (cf. [32]) and `Activities`. Also, even if at first sight, `StateMachines` don’t seem to have much changed from UML 1.5 to 2.0, this must be examined, and possible repercussions to the other parts of the behavioral semantics must be assessed.

Third, we have seen that it is quite straightforward to define a “*Petri-like semantics*” for simple `Activities` with only control-flow. But already for data-flow, some problems popped up (workflow type vs. instance, inscription semantics, initial markings/data-flows on initial edges). For the other, more powerful constructs proposed in the standard, it is entirely unclear whether a semantics may be defined based on Petri-nets—think of exception handling (`InterruptibleActivityRegion`), isolated concurrency (`ExpansionRegion`) and transactionality (`StructuredActivityNode`, `LoopNode`) in their various flavours. There might be a way of uniformly treating these by net unfoldings and/or procedure calling, but this definitely requires further work.

## References

1. Thomas Allweyer and Peter Loos. Process Orientation in UML through Integration of Event-Driven Process Chains. In Pierre-Alain Muller and Jean Bézivin, editors, *International Workshop «UML»'98: Beyond the Notation*, pages 183–193. Ecole Supérieure des Sciences Appliquées pour l'Ingénieur—Mulhouse, Université de Haute-Alsace, 1998.
2. L. Apvrille, P. de Saqui-Sannes, C. Lohr, P. Sénac, and J.-P. Courtiat. A New UML Profile for Real-Time System Formal Design and Validation. In Gogolla and Kobryn [17], pages 287–301.
3. João P. Barros and Luís Gomes. Actions as Activities as Petri nets. In Weber et al. [36], pages 129–135.
4. Christie Bolton and Jim Davies. Activity graphs and processes. In W. Griesskamp, T. Santen, and W. Stoddart, editors, *Proc. Intl. Conf. Integrated Formal Methods (IFM'00)*. Springer Verlag, 2000. LNCS.
5. Christie Bolton and Jim Davies. On giving a behavioural semantics to activity graphs. In Reggio et al. [29], pages 17–22.
6. Egon Börger, A. Cavarra, and E. Riccobene. An ASM Semantics for UML Activity Diagrams. In Teodor Rus, editor, *Proc. 8<sup>th</sup> Intl. Conf. Algebraic Methodology and Software Technology (AMAST 2000)*, pages 293–308. Springer Verlag, May 2000. LNCS 1816.
7. Ray J. A. Buhr. Use Case Maps as Architectural Entities for Complex Systems. *IEEE Transactions on Software Engineering*, 24(12):1131–1155, December 1998.
8. Marlon Dumas and Arthur H.M. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In Gogolla and Kobryn [17], pages 76–90.
9. Henrik Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, CTIT, U. Twente, 2002. Authors first name sometimes appears “Rik”.
10. Rik Eshuis and Roel Wieringa. A formal semantics for UML Activity Diagrams - Formalising workflow models. Technical Report CTIT-01-04, U. Twente, Dept. of Computer Science, 2001.
11. Rik Eshuis and Roel Wieringa. A Real-Time Execution Semantics for UML Activity Diagrams. In Heinrich Hussmann, editor, *Proc. 4<sup>th</sup> Intl. Conf. Fundamental approaches to software engineering (FASE'01)*, number 2029 in LNCS, pages 76–90. Springer Verlag, 2001. Also available as [wwwhome.cs.utwente.nl/~tcm/fase.pdf](http://wwwhome.cs.utwente.nl/~tcm/fase.pdf).
12. Rik Eshuis and Roel Wieringa. An Execution Algorithm for UML Activity Graphs. In Gogolla and Kobryn [17], pages 47–61.
13. Rik Eshuis and Roel Wieringa. Verification support for workflow design with UML activity graphs. In *Proc. 24<sup>th</sup> Intl. Conf. on Software Engineering (ICSE'02)*, pages 166–176. IEEE, 2002.
14. Rik Eshuis and Roel Wieringa. Comparing Petri Net and Activity Diagram Variants for Workflow Modelling - A Quest for Reactive Petri Nets. In Weber et al. [36], pages 321–351.
15. Thomas Gehrke, Ursula Goltz, and Heike Wehrheim. The Dynamic Models of UML: Towards a Semantics and its Application in the Development Process. Technical Report 11/98, Institut für Informatik, Universität Hildesheim, 1998.
16. H. Genrich and K. Lautenbach. *Predicate/Transition Nets*. In [20], 1991.
17. Martin Gogolla and Chris Kobryn, editors. *Proc. 4<sup>th</sup> Intl. Conf. on the Unified Modeling Language («UML» 2001)*, number 2185 in LNCS. Springer Verlag, 2001.
18. John Hosking and Philip Cox, editors. *Human Centric Computing Languages and Environments*. IEEE Computer Society, 2003.
19. Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. I*. EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1992.

20. Kurt Jensen and Grzegorz Rozenberg. *High-Level Petri Nets. Theory and Application*. Springer Verlag, 1991.
21. Xuandong Li, Meng Cui, Yu Pei, Zhao Jianhua, and Zheng Guoliang. Timing Analysis of UML Activity Diagrams. In Gogolla and Kobryn [17], pages 62–75.
22. Esperanza Marcos, Valeria de Castro, and Belen Vela. Representing Web Services with UML: A Case Study. pages 17–27.
23. Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proc. IEEE*, 77:541–580, April 1989.
24. OMG Unified Modeling Language: Superstructure (final adopted spec, version 2.0). Technical report, Object Management Group, November 2003. Available at [www.omg.org](http://www.omg.org), downloaded at November 11<sup>th</sup>, 2003, 11<sup>30</sup>.
25. Lawrence C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 1991.
26. Cesare Pautasso and Gustavo Alonso. Visual Composition of Web Services. In Hosking and Cox [18], pages 92–99.
27. Dorina C. Petriu and Yimei Sun. Consistent Behaviour Representation in Activity and Sequence Diagrams. In Bran Selic, Stuart Kent, and Andy Evans, editors, *Proc. 3<sup>rd</sup> Intl. Conf. <<UML>> 2000—Advancing the Standard*, number 1939 in LNCS, pages 369–382. Springer Verlag, October 2000.
28. Paulo Pinheiro da Silva. A proposal for a LOTOS-based semantics for UML. Technical Report UMCS-01-06-1, Dept. of Computer Science, U. Manchester, 2001.
29. Gianna Reggio, Alexander Knapp, Bernhard Rumpe, Bran Selic, and Roel Wieringa, editors. *Dynamic Behavior in UML Models: Semantic Questions. Workshop Proceedings*, Oktober 2000.
30. Roberto W.S. Rodrigues. Formalising UML Activity Diagrams using Finite State Processes. In Reggio et al. [29], pages 92–98.
31. Harald Störrle. Semantics of Activities in UML 2.0. submitted for review at VLFM'04, March, 20<sup>th</sup>, 2004.
32. Harald Störrle. Assert, Negate and Refinement in UML-2 Interactions. In Jan Jürjens, Bernhard Rumpe, Robert France, and Eduardo B. Fernandez, editors, *Critical Systems Development with UML - Proceedings of the UML'03 Workshop*, pages 79–94, 2003.
33. Harald Störrle. Semantics of Interactions in UML 2.0. In Hosking and Cox [18], pages 129–136.
34. Harald Störrle. Trace Semantics of Interactions in UML 2.0. *J. Visual Languages and Computing*, t.b.d. 2003. submitted for review, February, 13<sup>th</sup>, 2004.
35. Sebastian Thöne, Ralph Depke, and Gregor Engels. Process-Oriented, Flexible Composition of Web Services with UML. In *Proc. Intl. Ws. Conceptual Modeling Approaches for e-Business: A Web Service Perspective (eCOMO 2002)*, number 2784 in LNCS. Springer Verlag, 2002.
36. M. Weber, Hartmut Ehrig, and Wolfgang Reisig, editors. *Petri Net Technology for Communication-Based Systems*. DFG Research Group “Petri Net Technology”, 2003.