Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 127 (2005) 35–52
www.elsevier.com/locate/entcs

Semantics and Verification of Data Flow in UML 2.0 Activities

Harald Störrle[1]

Institut für Informatik, Ludwig-Maximilians-Universität München, GERMANY

Abstract

One of the major changes in going from UML 1.5 to UML 2.0 is the reengineering of activity diagrams. This paper examines activity diagramies as described in the current version of the UML 2.0 standard by defining a denotational semantics. It covers basic control flow and data flow, but excludes hierarchy, expansion nodes, and exception handling (see [18,19,20] for these issues). The paper shows, where the constructs proposed in the standard are not so easily formalized, and how the formalisation may be used for formal analysis.

Keywords: UML 2.0, activity diagrams, data flow, modeling of web services, workflows, and service oriented architectures

1 Introduction

1.1 Motivation and goal

Modeling of 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 have been introduced into the UML rather late. They have since been considered mainly as a workflow definition language, but it is also the natural choice when it comes to modeling web services, and plays an important role in specifying system level behaviors. The UML has become the "*lingua franca of software engineering*", and it has recently undergone a major revision (advancing from version 1.5

[1] Email: stoerrle@pst.ifi.lmu.de

doi:10.1016/j.entcs.2004.08.046

to version 2.0), including a complete redefinition of Activities. Unfortunately, the standard has yet again failed to define a formal semantics, as would be necessary to take full advantage of the UML, e.g., in automated tools.

Compared to UML 1.5, the concrete syntax of activity diagrams has remained mostly the same concerning control flow. 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 (ActivityGraph used to be a subclass of StateMachine in the Metamodel), there is now no such connection between the two: "*Activity replaces ActivityGraph in UML 1.5.*" (cf. [15, p. 292]). The standard claims that "*Activities are redesigned to use a Petri-like semantics instead of state machines*" (cf. [15, p. 292]). This paper tries to find out to which degree Petri-nets actually can be used to this purpose.

1.2 Approach

In order to find out, we examine the standard and try to define a formal semantics in terms of Petri-nets. Traditional P/T-nets, however, are not suitable for this task since activity diagrams also feature procedure call and data flow facilities. While the issue of procedure calling has been addressed in [18] using the notion of procedural Petri-net systems (cf. [12]), this paper turns towards the data flow issue. There are several extension to the basic Petri-net model that are capable of modeling data flow. These are generally subsumed under the title of "higher order nets" (cf. [11]), with [10,9] probably being the most well known.

1.3 Related work

As of writing this, the work on the UML 2.0 has reached the finalization stage. The technical work is said to have ended, and voting should be completed late this year, leading to an official endorsement of the new UML version probably early 2005. Thus, the currently available specification [15] are already pretty close to what will become the UML 2.0 later this year (or early next year).

In UML 2.0, there are numerous substantial improvements over UML 1.5. The metamodel has been totally reengineered, and is now much cleaner, more complete and orthogonal than it used to be. Many details have been improved, and the UML 2.0 is now object-oriented again in some meaningful way.

On the outside, some new notational elements have been added, but the most important innovation is, that notational elements may now be combined with much less restrictions than before.

Concerning the semantics, however, the UML is still not satisfactory. While many issues have been removed or resolved, and the descriptions are much cleaner for some aspects, there still is no formal semantics. In many cases, there are even no adequate examples, e.g., exceptions in Activities, and negation in Interactions, to name but two.

Truning to the scientific world, it seems that so far, very little work has been published on either data flow in activity diagrams (of whatever version) and activity diagrams in UML 2. Concerning data-flow in UML 1.x Activites, [1] seems to be the only reference.

Concerning UML 2 activity diagrams, [2] examines the semantics of pins very briefly. Also, there is a series of articles by C. Bock on UML 2 Activities one of which deals with data flow [3]. While being far more explicit and complete than the standard, these articles lack formal precision. Given that the author has participated in the OMG's standardization process of the UML 2 Activities, the interpretations provided should be considered, even if they are mere individual opinions and not part of the standard proper.

A more formal approach on UML 2 Activities is found in my previous publications: [18] deals with control flow and procedure calling, [19] deals with exceptions, and [20] deals with expansion regions and structured nodes. These papers also include extensive surveys of the literature.

There are also only few contributions dealing with formal analysis procedures for activity diagrams (without data flow, of course). Li et al. [13] show how duration constraints may be specified for Activities. They also provide a verification algorithm based on linear programming. Rodrigues [17] proposes to use finite state processes and seems to have some tool support for their formal analysis, without providing any details. Finally, [6,7,8] provide a mapping from a restricted class of activity diagrams to labeled transition systems and elaborate on the verification of temporal logic (CTL*) formulas and static structural reduction rules as known for Petri-nets.

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. Activity diagrams are best explained by means of an example (see Figure 1, adapted from [15, 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 2.2 for more on contexts).

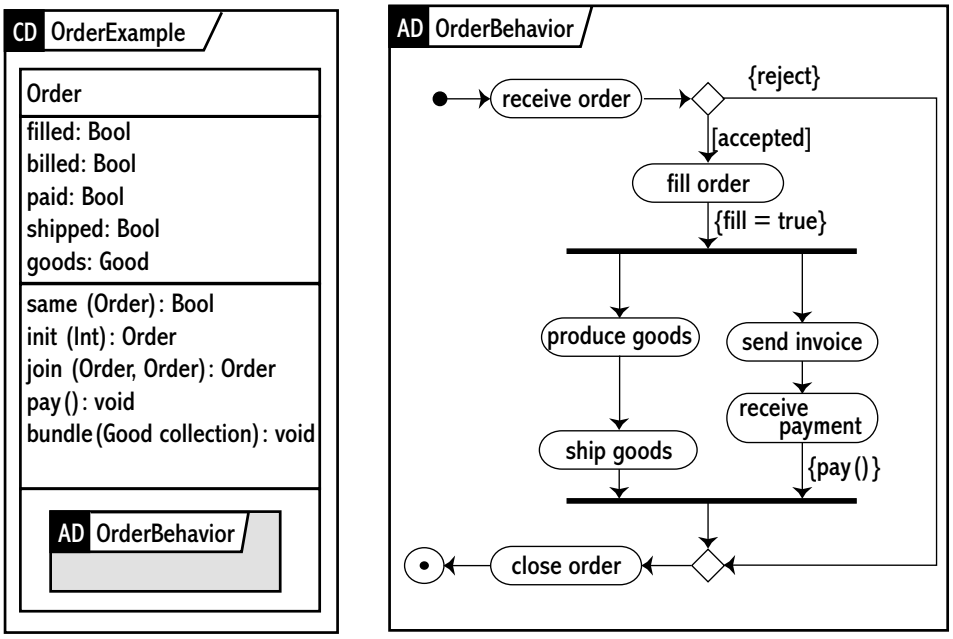

Fig. 1. An introductory specification consisting of a class diagram (left), and an activity diagram (right). The icon in the third compartment of class Order is a reference to the Activity specified in diagram OrderBehavior.

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

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

is underspecified. 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 digram 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.

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) difference 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 in UML 2.0 (cf. [4]). It does not influence the behavior of an Activity, however, and may thus be ignored here. SubactivityStates have vanished, and nesting is now accomplished by calling subordinate Activities from the Actions that define the behavior of superordinate Activities. Data flow is now denoted by ObjectNodes and ObjectFlows.

The standard allows three different notations for data flows (cf. Figure 2). 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.

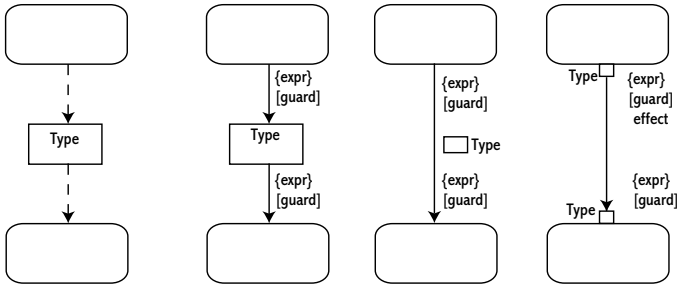

Fig. 2. Concrete syntax for data flows: UML 1.5 notation (left), and in alternative equivalent UML 2.0 notations (all others), including “attached data flow”, and “pin” notations (third and fourth).

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, and one can even view this as the foundation of an 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 to introduce 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.

We have to demand, though, that diagrams are complete, that is, there are no omissions. In particular, only those ObjectNodes and ObjectFlows are translated, that are actually present in an Activity. In practical settings, one would rarely fill in all the details of an activity diagram, but expect a human to understand the intuition. Figure 1 is 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. Also, for simplicity, we assume that the operations used in the inscriptions of the activitiy diagram are specified as methods of a class. The complete and correct example would thus be as shown in Figure 3. There, the effect reject is omitted, as it represents human decision, whose effects we chose not to implement in our model: we will be able to make this decision when simulating the net. For the same reason, the guard accepted has gone.

The standard does not specify a concrete language for inscriptions, including guards, effects on ObjectNodes and so on. Also, timing annotations are not defined. 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 SML. The details are explained in the section on data flow.

There is also a third notation for data flow proposed in the standard (see again Figure 2), which specifies Pins (a subclass of ObjectNode) as the "parameters" of the Activities (that may be called by the Actions that are executed in lieu of the ActivityNodes that have the Pins). The purpose of this

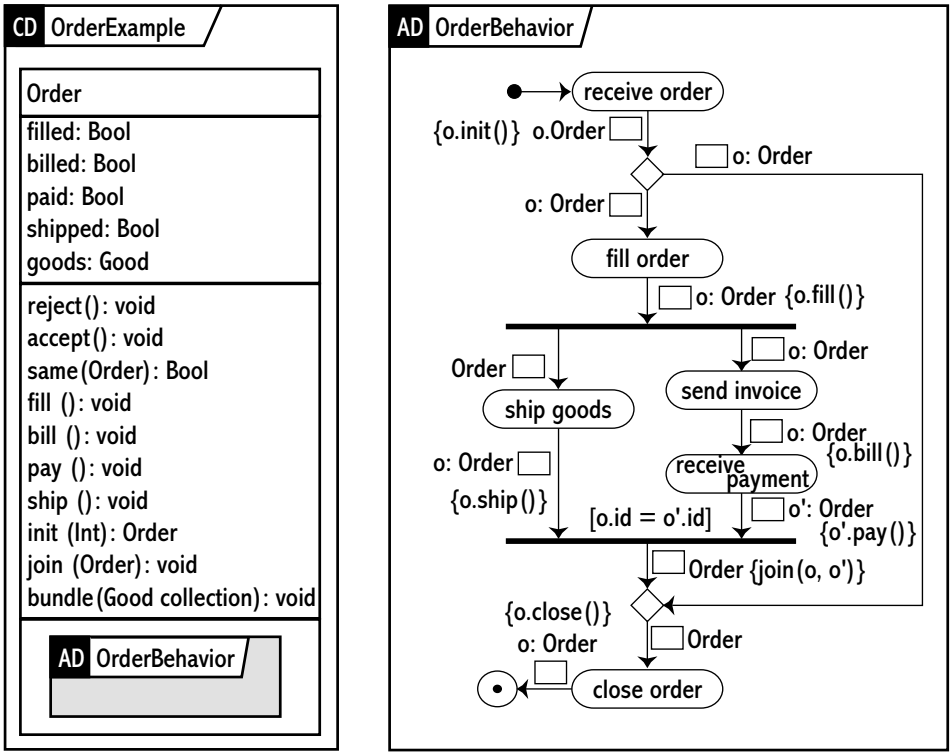

Fig. 3. A more diligently specified version of the activity diagram of Figure 1.

notation becomes more understandable in the context of procedure calling, see Figure 4.

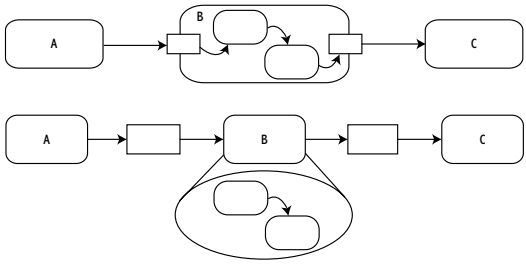

Fig. 4. Pins are ObjectNodes for refinement.

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 ActivityGraph in UML 1.5.*" (cf. [15, 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. [15, p. 292]). The metamodel defines six levels of 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.

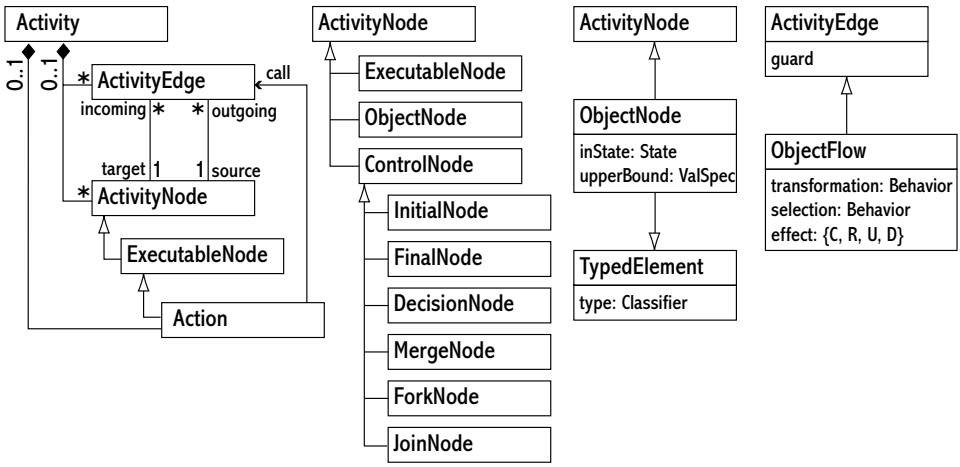

Fig. 5. A small portion of the UML 2.0 metamodel: Activities either have Actions or a graph of ActivityNodes and ActivityEdges (left); kinds of nodes and edges (right). The ObjectFlow.effect is an element of the ObjectFlowEffectKind-Enumeration (create, read, update, delete).

The basic two entities are Actions and Activities. While an Action "*is the fundamental unit of executable functionality*" (cf. [15, p. 280]), an Activity provides "*the coordinated sequencing of subordinate units whose individual elements are actions*" (cf. [15, 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);

iN, fN the InitialNodes and FinalNodes (of which there may be only one each);

BN the set of branch nodes, including both MergeNodes and DecisionNodes;

CN the set of concurrency nodes, subsuming ForkNodes and JoinNodes;

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.

From now on, we will use this abstract syntax representation of an Activity.

3 Semantics of data flow

In the new UML version, Activities now "*use a Petri-like semantics instead of state machines.*" (cf. [15, p. 263]). We will now try to check this by defining the formal semantics for Activities. 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 Figure 3). Procedure calling is ignored here—the treatment in [18] 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. We require that all elements are named with globally unique names.

Concerning data flow, only one notation is used (the one with ObjectFlows attached to ActivityEdges, see the third variant shown in Figure 2). All other notations are considered as syntactic sugaring. Observe, however, that for initial and final nodes, the traditional notation is necessary, and that an arc with an attached ObjectNode symbol really represents two ObjectFlows and an ObjectNode.

Then, only those ObjectNodes and ObjectFlows are translated, that are actually present in an Activity (cf. Figure 3). Implicit elements that a human observer might add in his mind are not translated.

3.1 Semantic domain

The data flow facilities of Activities may be represented by high level Petri-nets (cf. [11,10,9]). For pragmatic reasons—availability of good tool support to

name but one—colored Petri-nets (CPNs) are chosen as the semantic domain here (see Figure 7 for an example).

Definition 3.1 (structure of colored Petri-nets) *A colored Petri-net (CPN) is a tuple* $\langle N, SigAlg, color, guard, effect \rangle$ *with*

N	*is a Petri-net* $\langle P, T, A \rangle$ *of places, transitions, and arcs;*
$SigAlg$	*is a* Σ*-algebra* $\langle \Sigma, Op \rangle$ *of sorts and operations;*
$color$	*is a total function* $P \to \Sigma$ *assigning a sort ("color") to each place;*
$guard$	*is a total function* $T \to Expr$ *assigning a boolean expression to each transition, the default is the constant* tt*;*
$effect$	*is a total function* $A \to Expr$ *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 $color(p) = \mathsf{TOKEN}$ is intended, the defaults for *guard* and *effect* are true and skip, respectively.

The definition of the behavior of net systems 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 a multiset (or word) over $\{\langle p, v \rangle \mid p \in P, v \in color(p)\}$. As we lack the space for a complete definition, we can only provide an example here: consider Figure 8 for a sample run of the net of Figure 7, representing the activity diagram of Figure 3.

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 [18].

3.2 Semantic mapping

In this section, we first sketch the intuition behind the semantic mapping, and then provide a precise definition. With respect to control flow, ExecutableNodes become net transitions, ControlNodes become net places or small net fragments, and ActivityEdges become net arcs, possibly with auxiliary transitions or places. With respect to data flow, ObjectNodes become net places, and ObjectFlows become net arcs. See Figure 6 for an intuitive account of the translation.

At this point, we also need to deal with the inscriptions on ObjectNodes and ObjectFlows. The UML standard is rather unspecific concerning their language. We now try to clarify the requirements on the inscription language by examining the example provided by the UML standard (cf. Figure 3).

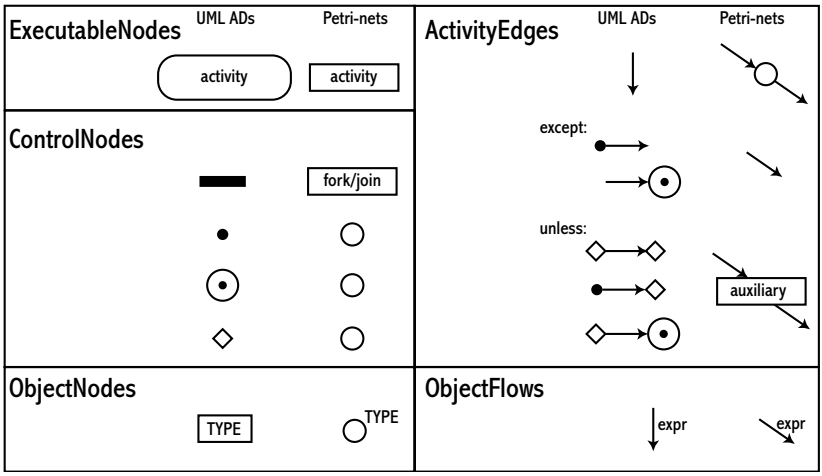

Fig. 6. The intuition of the semantic mapping for control and data flow of Activities.

In this example, there are basically four kinds of inscriptions. First, there are type declarations on ObjectNodes (they become colors of the respective CPN places in our interpretation). Some ObjectNodes also declare a variable (called o in the example) representing instances of the type residing in the ObjectNode. The set of of variables declared on ObjectFlows adjacent to an ActivityNode constitute the name space for the Action that the ActivityNode executes (assuming it is an ExecutableNode).

Second, there are effect, selection and transformation 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. The modeler must fill in the details, 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. In our interpretation, these functions remain at the respective net arc.

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. In our interpretation, they are moved up- or downstream to the next net transition. Again, the exact meaning is left for the modeler, and the expressions are turned into guards (closed boolean expressions) over the variables defined by the arcs adjacent to the transition representing the ActivityEdge.

Taking these requirements into consideration, we adopt the inscription language used in the CPN Toolset (see [5]), a dialect of Standard ML [16]. This way, tool support for case studies and analysis techniques is ensured. As an example, reconsider the Activity from Figure 3 and its translation into a

CPN shown in Figure 7 (this Figure is printed from the CPN Toolset). The text to the upper left of the net in Figure 7 is Standard ML code used in CPN Toolset defining the inscriptions: E is the type of the traditional black dot token, and ORDER is a custom defined type carrying the state of an order. The net is created manually using the CPN Toolset and is fully operational (see the screenshots of Figure 9).

The formal semantics is also rather straightforward. We have to map the abstract syntax representation of an Activity $\langle \mathit{ActivityNodes}, \mathit{ActivityEdges} \rangle$ into a CPN $\langle N, \mathit{SigAlg}, \mathit{color}, \mathit{guard}, \mathit{effect} \rangle$ by a function $[\![\_]\!]$, where $N = \langle P, T, A \rangle$ and *ActivityNodes* and *ActivityEdges* are partitioned into the various kinds of nodes and edges as explained above. The semantic function $[\![\_]\!]$ is defined by:

$$
\begin{aligned}
P &= \{iN, fN\} \cup BN \cup ON \\
&\quad \cup \{p_e \,|\, e \in AE, \{e.source, e.target\} \subseteq EN \cup CN\}, \\
T &= EN \cup CN \cup \{t_e \,|\, e \in AE, \{e.target, e.source\} \subseteq BN \cup \{iN, fN\}\}, \\
A &= \{\langle e.source, x_e \rangle, \langle x_e, e.target \rangle \,|\, e \in AE, x_e \in P \cup T\} \\
&\quad \cup \{\langle x, y \rangle \,|\, \langle x, y \rangle \in AE \wedge (x \in P, y \in T) \vee (x \in T, y \in P)\} \\
\mathit{SigAlg} &= \langle \{o.type \,|\, o \in ON\}, \{a.transformation \,|\, a \in OF\} \rangle \\
\mathit{color} &= \{o \mapsto o.type \,|\, o \in ON\} \\
\mathit{effect} &= \{a \mapsto a.transformation \,|\, a \in OF\} \\
\mathit{guard} &= \{t \mapsto \textstyle\bigwedge_{\langle a,t \rangle \in OF} a.selection \wedge \bigwedge_{\langle t,a \rangle \in OF} a.selection \,|\, a \in OF\}.
\end{aligned}
$$

Observe the use of the dot-notation to access the nodes adjacent to edges (cf. Figure 5). If o.upperBound is defined for an ObjectNode o, this may be interpreted as a capacity of the respective place using the canonical construction.

4 Analysis of data flow

With the semantics defined in the previous section, we may now transfer all the standard techniques for validation and verification of Petri-nets to UML 2.0 activity diagrams. In the remainder of this section, we assume that A is an Activity that is mapped to the CPN N by the semantics above (i.e. $[\![A]\!] = N$). We assume that there are sensible initial and final markings of N that correspond to initial and final states of A.

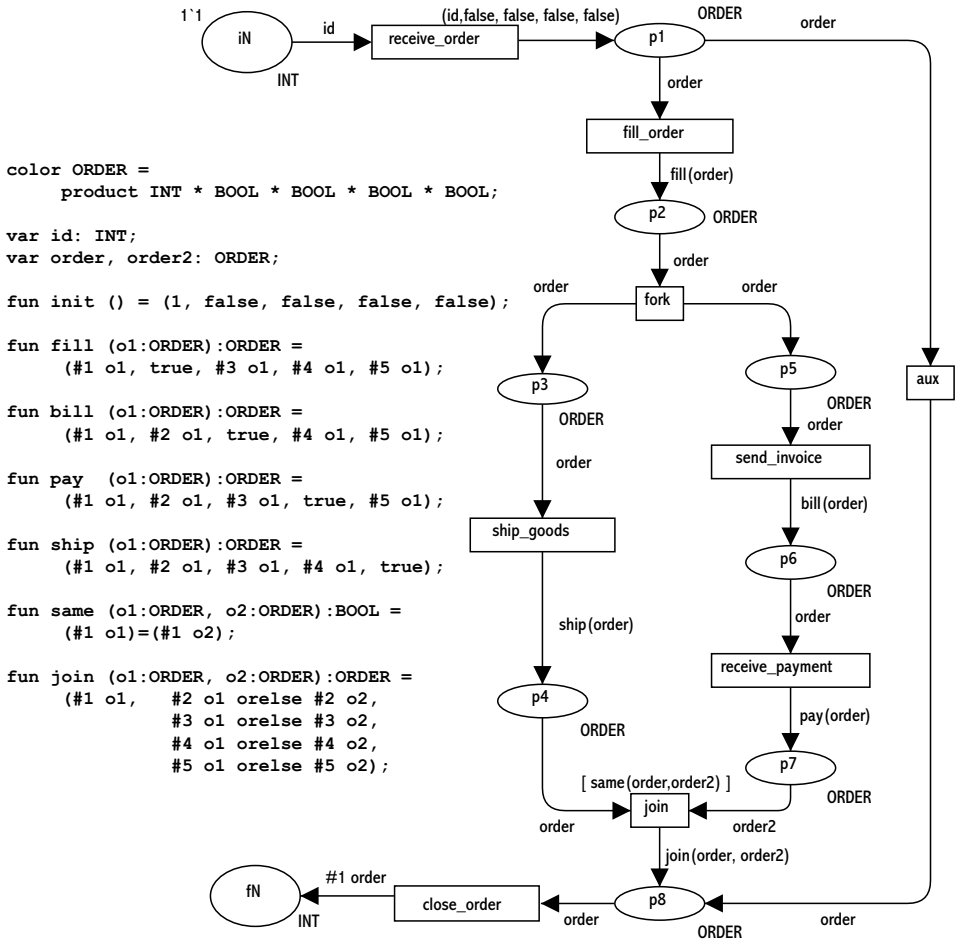

Fig. 7. The Petri-net representing the Activity and the type declared in Figure 3.

4.1 Validation

Traces can be a great help during testing or inspecting a design. With the CPN-semantics provided above, both methods are now available for UML 2.0 Activities. A trace of N can be turned into a trace of A by abstracting away the "internal" actions aux, fork, and join. Reconsider the Activity shown in Figure 3 that had been translated into the CPN of Figure 7.

Assuming an initial marking for the place representing the InitialNode, we can generate a trace either by a manual simulation ("the token game"), tool supported interactive simulation, or batch computation. Mapping the CPN trace back to the Activity is trivial. Since we chose CPNs with Standard ML inscriptions as our semantic domain, we may use the CPN Toolset as a

simulator. See Figure 9 for a screenshot of the CPN Toolset executing the net in Figure 7.[3] It would even be possible to use it as a basis for a visual simulation of the activity diagram.[4]

Generating a trace, however, requires an initial marking, something that is not provided in the UML standard. For simplicity, we take only one token with value 1 as the initial marking. In the inscription language used by CPN Toolset, this initial marking is written as `1‘1`.

marking	**transitions in a step from this marking**	**variable bindings for this step**
$\langle iN, \mathtt{1‘1}\rangle$	**receive_order**	$id = 1$
$\langle p_1, \langle \mathbf{1}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}\rangle\rangle$	**fill_order**	$order = \langle \mathbf{1}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}\rangle$
$\langle p_2, \langle \mathbf{1}, \mathbf{tt}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}\rangle\rangle$	**fork**	$order = \langle \mathbf{1}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}\rangle$
$\langle p_3, \langle \mathbf{1}, \mathbf{tt}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}\rangle\rangle$, $\langle p_5, \langle \mathbf{1}, \mathbf{tt}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}\rangle\rangle$	**send_invoice** **ship_goods**	$order = \langle \mathbf{1}, \mathbf{tt}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}\rangle$ $order = \langle \mathbf{1}, \mathbf{tt}, \mathbf{ff}, \mathbf{ff}, \mathbf{ff}\rangle$
$\langle p_4, \langle \mathbf{1}, \mathbf{tt}, \mathbf{ff}, \mathbf{ff}, \mathbf{tt}\rangle\rangle$, $\langle p_6, \langle \mathbf{1}, \mathbf{tt}, \mathbf{tt}, \mathbf{ff}, \mathbf{ff}\rangle\rangle$	**receive_payment**	$order = \langle \mathbf{1}, \mathbf{tt}, \mathbf{tt}, \mathbf{ff}, \mathbf{ff}\rangle$
$\langle p_4, \langle \mathbf{1}, \mathbf{tt}, \mathbf{ff}, \mathbf{ff}, \mathbf{tt}\rangle\rangle$, $\langle p_7, \langle \mathbf{1}, \mathbf{tt}, \mathbf{tt}, \mathbf{tt}, \mathbf{ff}\rangle\rangle$	**join**	$order = \langle \mathbf{1}, \mathbf{tt}, \mathbf{ff}, \mathbf{ff}, \mathbf{tt}\rangle$ $order2 = \langle \mathbf{1}, \mathbf{tt}, \mathbf{tt}, \mathbf{tt}, \mathbf{ff}\rangle$
$\langle p_8, \langle \mathbf{1}, \mathbf{tt}, \mathbf{tt}, \mathbf{tt}, \mathbf{tt}\rangle\rangle$	**close_order**	$id = 1$
$\langle fN, \mathtt{1‘1}\rangle$		

Fig. 8. A run of the CPN in Fig. 7, representing the Activity specified in Fig. 3 (each row is a step). Here, we use the simplified initial marking `1‘1` at place iN.

4.2 Standard properties

Often it is of great practical value to determine whether certain states may or may not be reached. For instance, the question "will the terminal state of A be reached under all circumstances?" may be formalized as

$$\forall_{m \in \mathsf{R}_N(\overline{m})} : \underline{m} \in \mathsf{R}_N(m),$$

where $\mathsf{R}_N(m)$ denotes the set of markings of N reachable from m, and $\overline{m}$ and $\underline{m}$ are the initial and final markings of N, respectively. Similarly, the absence of deadlocks may be verified by ensuring that

$$\forall_{m \in \mathsf{R}_N(\overline{m})} : \exists_{t \in T_N} : m \xrightarrow{t},$$

[3] For this example, the translation from Activity to CPN has still been done manually, but I am working on automating the process.

[4] The predecessor of CPN Toolset has originally been designed as a simulator for IDEF.

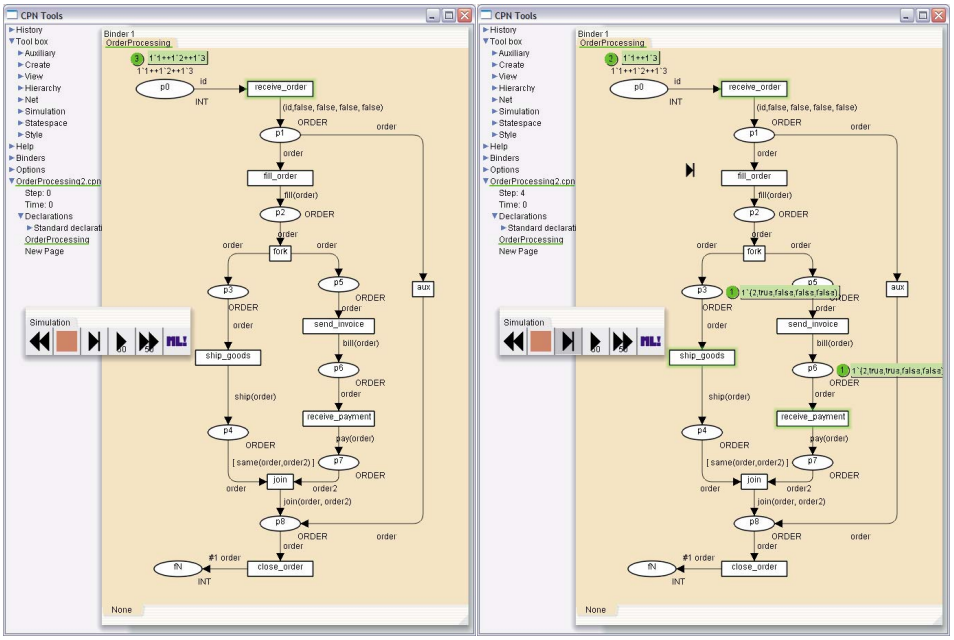

Fig. 9. Two screenshots of a simulation run of the net shown in Figure 7 using Design Toolset: the initial state (left) and the state after sending the first invoice but before shipping goods for it (right).

where $m \xrightarrow{t}$ is the Petri-net notation to express that transition t is activated in m. Finally, properties like "if an order is filled, will the respective goods be shipped eventually?" may expressed in a similar way:

$$\forall_{m \in \mathsf{R}_N(\overline{m})} : m \xrightarrow{\textbf{fill\_order}} \Longrightarrow \exists_{m' \in \mathsf{R}_N(m)} : m' \xrightarrow{\textbf{ship\_goods}\, order}$$

These properties are structurally rather similar, so that translating intuitive properties into Petri-net terminology and interpreting their results for the UML Activity is quite easy.

4.3 Quantitative analysis

An important reason for using Petri-net based tools in the design and analysis of business processes, logistics and manufacturing problems and so on has always been the possibility to do simulations and apply quantitative analysis techniques. In these cases, it is not sufficient to verify that ordered goods are shipped *eventually*, but we need to ensure certain time bounds.

Also, for many applications, it is quite useful to determine the number of orders that reside in the system at any given time or on average, and to

determine minimum required capacity for the system to cope with a given load. These kinds of questions are often attacked with stochastic Petri-nets, and there is a large body of work dealing with performance-related questions for UML 1.x activity diagrams—not for UML 2.0, though.

Unfortunately, the standard does not provide the kinds of inscriptions necessary to apply the typical analysis techniques for Petri-nets, much less the conceptual framework required. Thus, we would need syntactic extensions to UML activity diagrams to cover aspects like duration, frequency, probability, and latency of Actions. While there have been a number of approaches to provide and exploit such extensions for UML 1.x (e.g. [13,21,14]), it is not clear, if and how these can be transferred to the new version of the UML.

5 Conclusion

In this paper, a formal semantics of activity diagrams in UML 2.0 is defined. It is based on colored Petri-nets and covers control flow, concurrency and data flow, but not procedure call, exception handling, and expansion regions (see [18,19,20] for these aspects). By a carefully choice of net formalism and inscription language, the standard analysis techniques and tools for CPNs are made accessible for the verification and validation of UML Activities.

Since the semantics presented here preserves the structure of the original activity diagram in the resulting Petri-net, it is very easy to map between Activities and corresponding Petri-nets, opening the road to easy mapping between an Activity and corresponding CPN, and even visualisation of executions of Activities. This is very difficult if not impossible for semantics based on a non-graphical formalism, like process algebras. Finally, defining the semantics also helped identify a number shortcomings in the standard.

A tool implementation to support industrial case-studies is currently under way, together with extensions and automation of the analysis techniques. This effort is impeded, however, by the current lack of true UML 2.0 tools: despite the marketing promises by many vendors, there are currently no such tools available. With the official adoption of the UML 2.0 standard, we are hoping that this unpleasant situation will disappear.

The next step is to extend this semantics to also cover quantitative aspects like processing time and amount of processed data. Also, processing of streaming data is an unsolved problem. It would be interesting to see, whether these extensions allow quantitative analysis of system architectures at an early stage during development. In the Petri-net world, analysis techniques based on Markov-chains have a long tradition, so it seems feasible to turn the semantics presented in this paper into one that creates Generalised Stochastic

Petri-nets instead of Colored Petri-nets.

References

[1] Allweyer, T. and P. Loos, *Process Orientation in UML through Integration of Event-Driven Process Chains*, in: P.-A. Muller and J. Bézivin, editors, *International Workshop ≪UML≫'98: Beyond the Notation* (1998), pp. 183–193.

[2] Barros, J. P. and L. Gomes, *Actions as Activities as Petri nets*, in: J. Jürjens, B. Rumpe, R. France and E. B. Fernandey, editors, *Proc. Ws. Critical Systems Development with UML*, 2003, pp. 129–135.

[3] Bock, C., *UML 2 Activity and Action Models: Object Nodes*, J. Object Technology **3** (2004), pp. 27–41, available at www.jot.fm.

[4] Buhr, R. J. A., *Use Case Maps as Architectural Entities for Complex Systems*, IEEE Transactions on Software Engineering **24** (1998), pp. 1131–1155.

[5] CPN Tools Team, *CPN Tools Manual*, Technical report, Univ. of Aarhus (2004), available at http://wiki.daimi.au.dk/cpntools/cpntools.wiki.

[6] Eshuis, H., "Semantics and Verification of UML Activity Diagrams for Workflow Modelling," Ph.D. thesis, CTIT, U. Twente (2002), Author's first name sometimes appears as "Rik".

[7] Eshuis, R. and R. Wieringa, *A formal semantics for UML Activity Diagrams - Formalising workflow models*, Technical Report CTIT-01-04, U. Twente, Dept. of Computer Science (2001).

[8] Eshuis, R. and R. Wieringa, *Verification support for workflow design with UML activity graphs*, in: *Proc. 24th Intl. Conf. on Software Engineering (ICSE)* (2002), pp. 166–176.

[9] Genrich, H. and K. Lautenbach, "Predicate/Transition Nets," in: [11] .

[10] Jensen, K., "Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. I," EATCS Monographs on Theoretical Computer Science, Springer Verlag, 1992.

[11] Jensen, K. and G. Rozenberg, "High-Level Petri Nets. Theory and Application," Springer Verlag, 1991.

[12] Kiehn, A., "A Structuring Mechanism for Petri Nets," Dissertation, TU München (1989), appeared as Technical Report TUM-I8902 of the TU München in March, 1989.

[13] Li, X., M. Cui, Y. Pei, Z. Jianhua and Z. Guoliang, *Timing Analysis of UML Activity Diagrams*, in: M. Gogolla and C. Kobryn, editors, *Proc. 4th Intl. Conf. on the Unified Modeling Language (≪UML≫ 2001)*, number 2185 in LNCS (2001), pp. 62–75.

[14] Merseguer, J. and J. Campos, "Software Performance Modelling Using UML and Petri Nets," LNCS **2965**, Springer Verlag, 2004 pp. 265–289.

[15] OMG, *OMG Unified Modeling Language: Superstructure (final adopted spec, version 2.0, 2003-08-02)*, Technical report, Object Management Group (2003), available at www.omg.org, downloaded at November 11th, 2003.

[16] Paulson, L. C., "ML for the Working Programmer," Cambridge University Press, 1991.

[17] Rodrigues, R. W., *Formalising UML Activity Diagrams using Finite State Processes*, in: G. Reggio, A. Knapp, B. Rumpe, B. Selic and R. Wieringa, editors, *Proc. Intl. Ws. Dynamic Behavior in UML Models: Semantic Questions. Technical Report No. 0006 of the Ludwig-Maximilians-Universität, München, Inst. f. Informatik*, 2000, pp. 92–98.

[18] Störrle, H., *Semantics of Control-Flow in UML 2.0 Activities*, in: P. Bottoni, C. Hundhausen, S. Levialdi and G. Tortora, editors, *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2004), pp. 235–242.

[19] Störrle, H., *Semantics of Exceptions in UML 2.0 Activities* (2004), submitted to Journal of Software and Systems Modeling, May, 9^{th}, available at www.pst.informatik.uni-muenchen.de/~stoerrle.

[20] Störrle, H., *Semantics of Expansion Nodes in UML 2.0 Activities*, in: I. Porres, editor, *Proc.* 2^{nd} *Nordic Ws. on UML, Modeling, Methods and Tools (NWUML'04)*, 2004.

[21] Xu, J., M. Woodside and D. Petriu, *Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time*, in: *Proc.* 13^{th} *Intl. Conf. Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS'03)*, number 2794 in LNCS (2003), pp. 291–310.