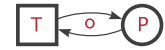


Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany



Lehrstuhl Theorie der Programmierung
Group Theory of Programming



Translating UML2 Activity Diagrams to Petri Nets

for analyzing IBM WebSphere Business Modeler process models

Dirk Fahland
fahland@informatik.hu-berlin.de

October 16, 2008

We present a formal semantics for a variant of UML2 Activity Diagrams that is used in the IBM WebSphere Business Modeler for modeling business processes. Business process models created in the IBM WebSphere Business Modeler or with other UML2 modeling tools often constitutes one of the key specification artifacts for building an information system that implements or supports the specified processes. As UML2 Activity Diagrams lack a universally agreed semantics, the step from specification to implementation usually faces a semantical impedance caused by different interpretation of the same diagram. A well-defined formal semantics for the specification language determines the interpretation of a diagram and allows for reasoning about as well as validating and verifying a given specification on common (formal) grounds.

We adapt approaches for formalizing semantics of UML2 Activity Diagrams and apply them to the core features of the IBM WebSphere Business Modeler language for purpose of formal verification. We provide a parameterized Petri net pattern for each language concepts. A diagram is translated by instantiating a pattern for each use of a concept; merging the resulting Petri net fragments according to the structure of the original diagram yields a Petri net that specifies the behavioral semantics of the diagram. The resulting Petri net can be verified for control-flow errors using the model checker LoLA. The semantics has been implemented in a tool that is available at <http://www.service-technology.org/uml2owfn/>.

Contents

1	Introduction	4
2	Basic Concepts and Patterns	6
2.1	Nodes, Pins and Edges	6
2.2	Task	7
2.3	Edges	8
2.4	Gateway Nodes	9
2.4.1	Decision	9
2.4.2	Merge	9
2.4.3	Fork	11
2.4.4	Join	11
2.5	Process	12
2.5.1	Formal Semantics of UML2-AD Processes	13
3	Advanced Concepts and Patterns	15
3.1	Pinsets	15
3.1.1	Processes with Pinsets	15
3.1.2	Proper Termination of Processes with Pinsets	17
3.1.3	Tasks with Pinsets	17
3.2	Optional Pins	18
4	Variants of Patterns	19
4.1	Edges by Merging Places	19
4.2	Call to Local Processes	20
5	Conclusion	21

1 Introduction

Diagram-based models are a widely used means for documenting and specifying business processes and workflows. Usually, such diagrams follow standards like event-driven process chains, the business process modeling notation, or, increasingly, UML2 Activity Diagrams (UML2-AD) [2] which are supported various tools. Unlike programming languages these graphical modeling languages do not come with a unique, universally agreed behavioral semantics. Thus, for a given graphical business process model, there exist many different interpretations.

Such business process models, often created by domain experts with few or non computer science background, abstractly denote the logics of actual work procedures and business processes. Their interpretation usually emerges from an intuitive understanding of how arrows represent causality or flow of information and goods. This intuitive interpretation reaches its limits if more complex concepts for process model are involved, as they are provided by all present-day languages; for instance consider parameterized enabling of activities, multiple instantiations, or implicit OR-joins. Whenever a business process model shall serve as a blue print for the architecture and implementation of an information system, a commonly agreed interpretation of a given diagram is important.

Providing a priori a formal behavioral semantics for a graphical modeling language avoids negotiations and discussions about the interpretation of a diagram. With such a semantics, any model can be simulated, validated, and even verified during the modeling process.

In the following, we consider a variant of UML2-AD that is implemented in the IBM WebSphere Business Modeler. The IBM WebSphere Business Modeler allows creating libraries consisting of several hundred, inter-related processes together with specifications of resources and data. Each process is modeled in a graphical editor using the mentioned UML2-AD variant and can be simulated based on a proprietary implementation of the UML2-AD semantics. These semantics not necessarily follow the UML2-AD standard [2] and cannot be used to transform a given process model into another format for the purpose of formal verification, implementation, or code generation.

Subsequently, we address the issue of providing a formal semantics for the core of UML2-AD as they are used in the IBM WebSphere Business Modeler for the purpose of verifying a library of process models. We have chosen Petri nets as the semantical domain of our interpretation, because a number of structural and behavioral analysis techniques are available for this formalism [4, 8].

Briefly sketched, our semantics is a model transformation from graphs denoting UML2-AD into graphs that denote Petri nets. The former graphs are constructed by instantiating UML2-AD concepts into a set of (complex) nodes which are connected with links. Our semantics provides a parameterized Petri net pattern for each UML2-AD concept

(or rather, a subset of the concepts we are interested in). The model transformation thus re-constructs the process model using Petri net patterns:

1. For each node of the original UML2-AD model, we create a new instance of the corresponding Petri net pattern yielding a set of place-bordered Petri net fragments.
2. Each border place either represents the source or the target of an edge in the UML2-AD model.
3. Adding a connecting transition, that consumes from the source and produces on the target place, for all corresponding places of all generated Petri net fragments yields the resulting Petri net.

This approach follows existing works on formal semantics for UML2-AD [5, 6, 7].

In the following, we assume the reader to be familiar with Petri nets; an introduction to the formalism and its application can be found in [3]. The remainder of this paper is organized as follows. We introduce and explain those UML2-AD concepts used in the IBM WebSphere Business Modeler which we consider for translation as we present the corresponding Petri net patterns. We begin with the basic concepts in Section 2. This section is supplemented with more advanced patterns in Section 3 to capture variants and extensions of the basic concepts. We conclude our formalization in Section 5.

2 Basic Concepts and Patterns

This section introduces the basic UML2-AD concepts and corresponding Petri net patterns.

2.1 Nodes, Pins and Edges

Each UML2-AD diagram consists of a set of (complex) *nodes* that are connected via *edges*. Each UML2-AD node has a set of *input pins* and *output pins* which may be partitioned further. A UML2-AD edge begins at one output pin and ends at one input pin, connecting the two pins, and hence the two nodes. This is the basic UML2-AD diagram construction and composition principle.

The basic UML2-AD semantics uses tokens: a state of a UML2-AD diagram assigns a (non-negative) number of tokens to the pins of its nodes. Depending on the type of the node, a node is *enabled* if one, several, or all of its input pins carry a token. An enabled node may *execute* which consumes all tokens (that caused the enabling of the node) and produces a number of nodes on one, several, or all of the nodes output pins. A token on an output pin is propagated along the adjacent edges to the next input pin.

In our semantics, we provide a place-bordered *Petri net pattern* for each UML2-AD concept. The pins of UML2-AD nodes are translated as *interface* places. An input pin becomes an input place of the pattern (the pattern will only consume from this place), and an output pin becomes an output place of the pattern (the pattern will only produce on this place). Figure 2.1 depicts the general pattern of this formalization. *Instantiating* the pattern will fix a number of input and output places, and its interior consisting of further places and transitions. This yields a place-bordered *Petri net fragment*.

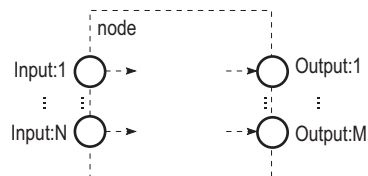


Figure 2.1: General pattern for translating UML2-AD nodes to Petri nets.

To formalize an UML2-AD edge that connects two pins (of two UML2-AD nodes), we add a connecting transition between the corresponding input and output places of the corresponding Petri net fragment. Figure 2.2 depicts an example of this translation: Some UML2-AD node X (with 1 input pin and 2 output pins) and some UML2-AD node Y (with 2 input and 2 output pins) are translated to corresponding Petri net fragments.

A UML2-AD edge from the second output pin of X to the first input pin of Y yields a transition that consumes from place X.output.2 and produces on place Y.input.1.

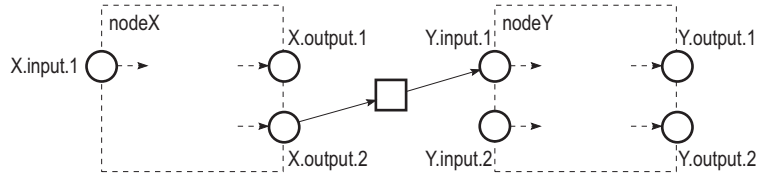


Figure 2.2: General pattern for translating UML2-AD edges to Petri nets.

This formalization faithfully captures the token-based semantics of UML2 Activity diagrams. Subsequently, we will specify for each UML2-AD concept the interior of the corresponding Petri net pattern.

2.2 Task

A UML2-AD *task* specifies a working step. The pins of a task (and all subsequent nodes) are partitioned into *control flow* pins and *data-flow* pins. This distinction will play a role when we consider more elaborate concepts for specifying UML2-AD.

In the standard case, a task is enabled iff each input pin has one token. It executes by consuming the enabling tokens from its input pins and producing one token on each output pin. Figure 2.3 depicts this case; the left hand side shows the graphical representation of a task in the IBM WebSphere Business Modeler, the right hand side shows the corresponding Petri net pattern.

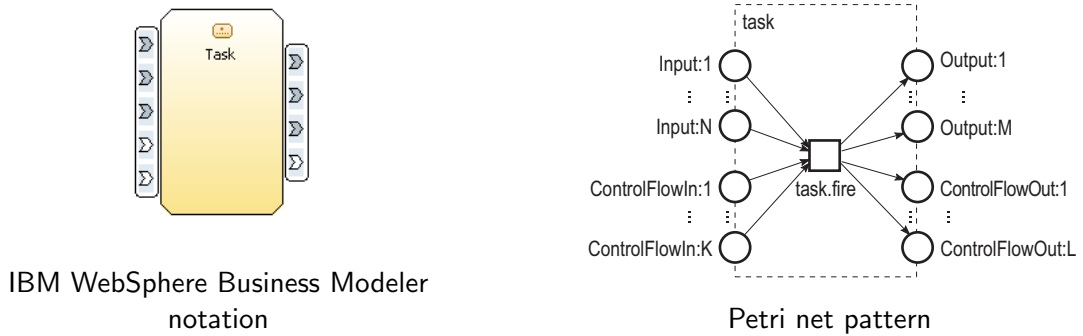


Figure 2.3: Activity with multiple data and control flow inputs and outputs

Figure 2.4 displays the special case of a task with exactly one control-flow input pin and exactly one control-flow output pin.

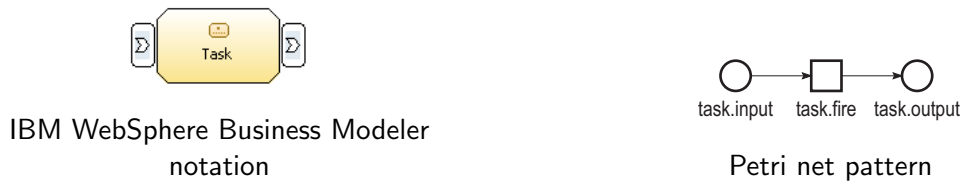


Figure 2.4: Activity

2.3 Edges

As already explained in Sect. 2.1, a UML2-AD edge can directly be translated as a transition consuming from the place that represents the starting point of the edge and producing on the place that represents the end point of the edge.

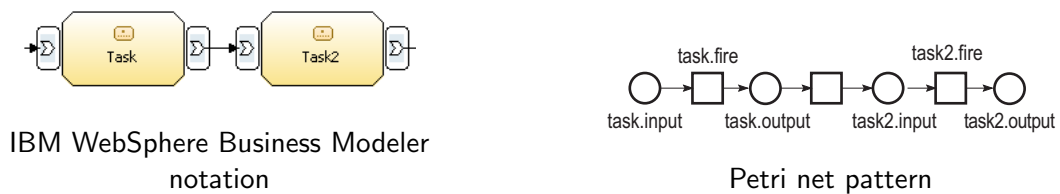


Figure 2.5: A UML2-AD edge is translated into a transition

A special case arises when in the original UML2-AD, a pin has no incoming or no outgoing edge. The first case would prevent the enabling of a node, the second case would leave a token in the process that cannot be propagated further. A strict interpretation would consider these cases as modeling errors. We also propose a more liberal translation: In case an input or output pin of a node is not connected to another pin, we omit the corresponding place in the translation.

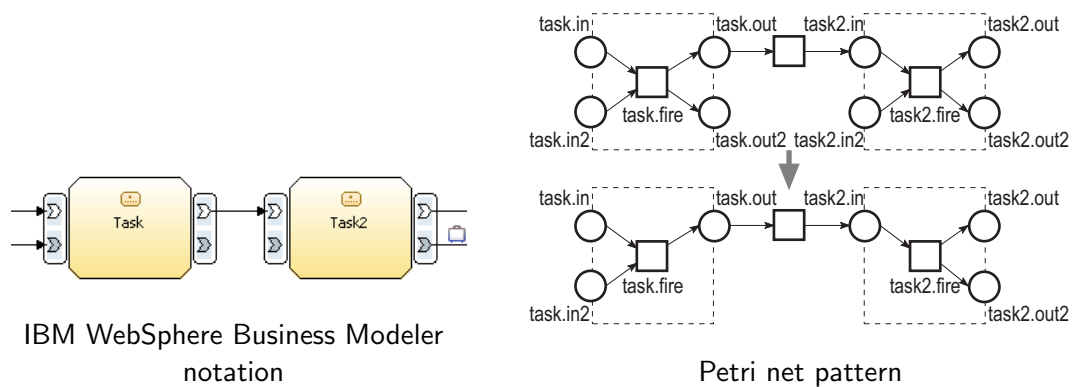


Figure 2.6: Unconnected pins are removed in the translation

An alternative pattern that merges places is shown in Sect. 4.1.

2.4 Gateway Nodes

UML2-AD allow to control the flow of tokens by means of *gateway* nodes. The standard gateway nodes are the *decision* (also XOR-split or OR-split), *merge* (also XOR-join), *fork* (also AND-split) and *join* (also AND-join). We explain semantics and implementing Petri net patterns for each gateway.

2.4.1 Decision

A UML2-AD decision node partitions its output pins into *output branches*. A decision is enabled if all input pins have a token. It executes by consuming the enabling tokens, choosing one output branch, and producing a token on each output pin of the output branch. Figure 2.7 shows the graphical representation of a UML2-AD decision in the IBM WebSphere Business Modeler, Fig. 2.8 depicts the corresponding Petri net pattern.

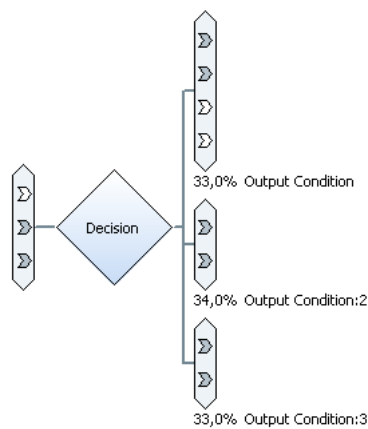


Figure 2.7: Decision in IBM WebSphere Business Modeler notation

The IBM WebSphere Business Modeler allows (or rather requires) to specify the probability of choosing one output branch. This information is not translated to the Petri net pattern, thus we abstract from probabilities and make the choice of the output branch non-deterministic. This abstraction preserves all runs (except for the implausible case of probability 0 for choosing one output branch; in this case, the branch in the Petri net would be omitted upon instantiation).

2.4.2 Merge

A UML2-AD *merge* is the counter-part of a UML2-AD decision. A merge node partitions its input pins into *input branches*. A merge is enabled if all input pins of *one* input branch have a token. It executes by consuming the enabling tokens and producing a token on each of its output pins. Figure 2.9 shows the graphical representation of a UML2-AD merge in the IBM WebSphere Business Modeler, Fig. 2.10 depicts the corresponding Petri net pattern.

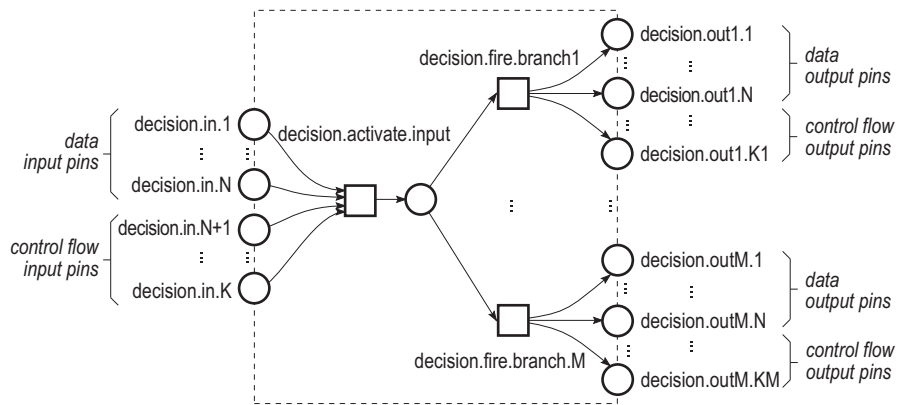


Figure 2.8: Decision Petri net pattern, probabilities are not translated

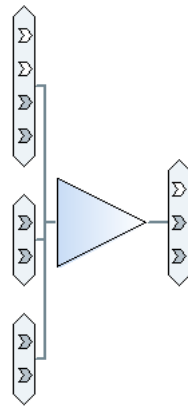


Figure 2.9: Merge in IBM WebSphere Business Modeler notation

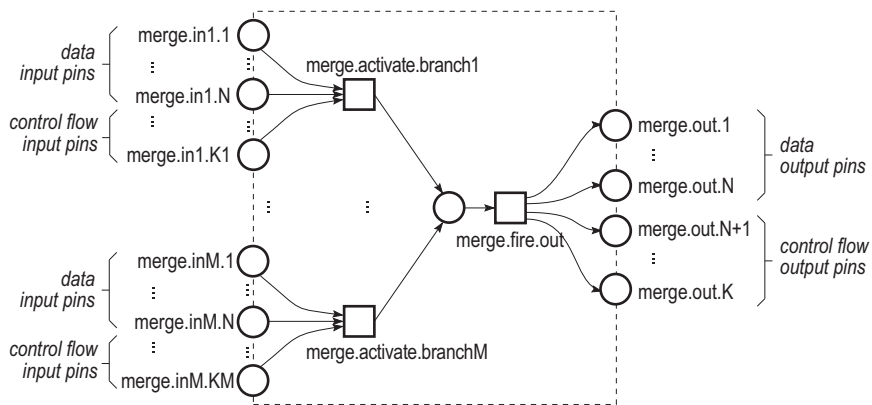


Figure 2.10: Merge petri net pattern

2.4.3 Fork

A UML2-AD *fork* is similar to a split, but it puts tokens on all of its output branches: A fork partitions its output pins into output branches. A fork is enabled if all input pins have one token. It executes by consuming the enabling tokens and producing a token on each output pin of each output branch. Figure 2.11 shows the graphical representation of a UML2-AD fork in the IBM WebSphere Business Modeler, Fig. 2.12 depicts the corresponding Petri net pattern.

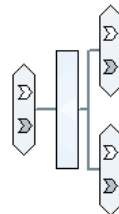


Figure 2.11: Fork in IBM WebSphere Business Modeler notation

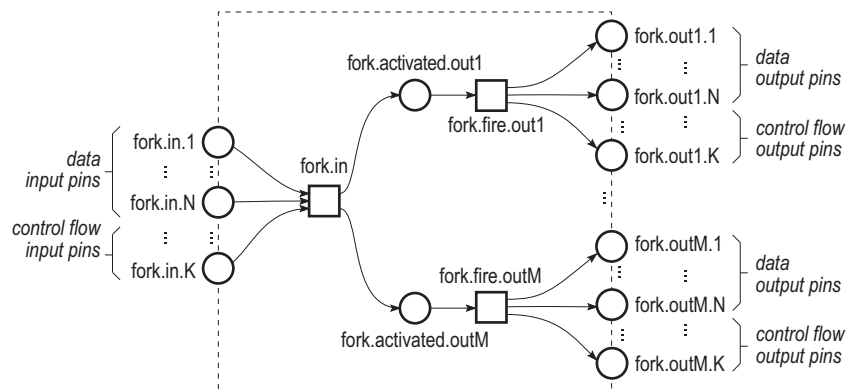


Figure 2.12: Fork Petri net pattern

A pattern that directly relates inputs and outputs without an intermediate place is shown in Sect. ??.

2.4.4 Join

A UML2-AD *join* is the counter-part of a UML2-AD fork. A join node partitions its input pins into *input branches*. A join is enabled if all input pins of *all* input branches have a token. It executes by consuming the enabling tokens and producing a token on each of its output pins. Figure 2.13 shows the graphical representation of a UML2-AD merge in the IBM WebSphere Business Modeler, Fig. 2.14 depicts the corresponding Petri net pattern.

This concludes the basic patterns for standard nodes occurring in IBM WebSphere Business Modeler diagrams.

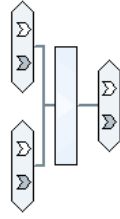


Figure 2.13: Join in IBM WebSphere Business Modeler notation

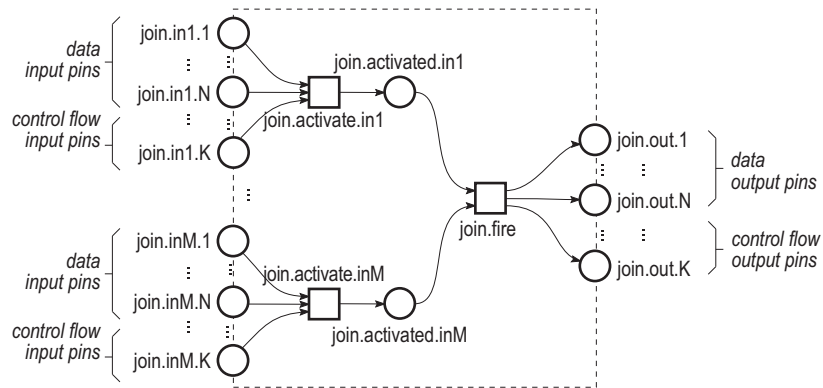


Figure 2.14: Join Petri net pattern

2.5 Process

A UML2-AD process itself is not only a set of nodes, but it provides a “shell” and process instantiation and termination semantics. We explain these semantics and its formalization in Petri nets. Figure 2.15 depicts the IBM WebSphere Business Modeler syntax of a process.

A UML2-AD process has a (possibly empty) set of input pins and a (possibly empty) set of output pins. It further has a set of *start nodes* as well as a set of *end nodes* and *stop nodes*. Start, stop, and end nodes can be conceived as single pins.

Start nodes are starting points for control-flow edges, input pins are starting points for data-flow edges. End nodes and stop nodes are ending points for control-flow edges and output pins are ending points for data-flow edges.

A UML2-AD process is enabled if all input pins have a token. A process execution is not atomic, but stateful (determined by the behavior of the nodes and edges it contains). A process begins its execution by putting a token on each start node (which is then propagated along its outgoing edge). A process terminates if it contains no token anymore (except on its output pins). A token that reaches an end node simply vanishes, a token that reaches a stop node causes the immediate removal of all remaining tokens in the process. This termination is *proper* if upon termination, each output pin of the process has a token.

Termination semantics are rather complex. It would be possible to provide a Petri



IBM WebSphere Business Modeler notation

Figure 2.15: A single process

net translation that represents exactly the behavior described above (including removal of all remaining tokens upon putting a token on a stop node). But as the aim of this translation is the analysis of faults, such complicated cases can be handled later in the analysis. We therefore provide a rather straight forward Petri net pattern that formalizes the process execution and consider termination semantics during the analysis.

The structure of the process is translated canonically: Each data input pin of the process is an input place of the net, each data output pin of the process is an output place of the net. Each start node of the process becomes an *initial control flow* place `process.start.i`, stop nodes are translated to corresponding places `process.stop.s` while each end node is translated to a corresponding place `process.end.r` which has a dedicated transition only consuming tokens from `process.end.r` but not producing any token.

Each `process.start.i` place gets a token only if all input data is available. To handle this issue, we provide a “shell” around the process: We replicate process input and output places and add a transition that consumes from all replicated (outer) input places and produces on all original (inner) input places (for process output places correspondingly in the opposite direction). The input-replication transition also produces the control-flow tokens on all `process.start.i` places at the moment when the data-flow tokens are available. Figure 2.16 depicts this pattern.

2.5.1 Formal Semantics of UML2-AD Processes

To translate a UML2-AD process into Petri nets,

1. instantiate the process pattern,
2. instantiate a corresponding pattern for each UML2-AD node that is contained in the process, which results in a set of Petri net fragments with input and output places,
3. connect the input and output places of the fragments according to the UML2-AD edges between corresponding UML2-AD input and output pins, and

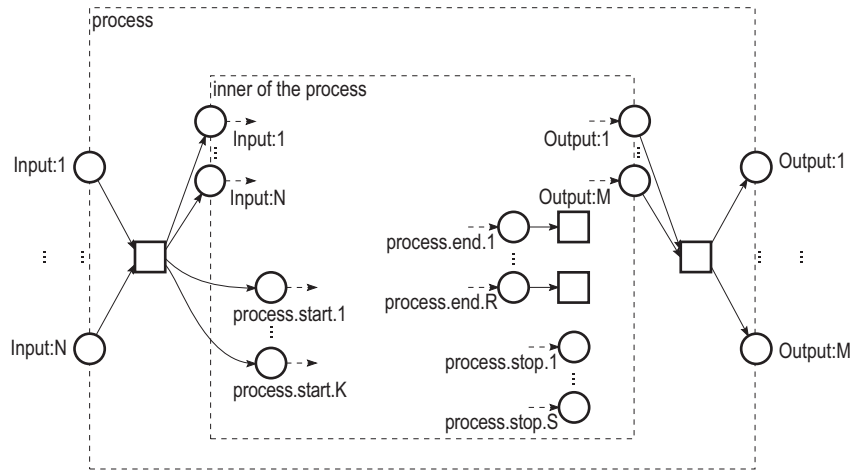


Figure 2.16: Petri net pattern for a process

4. remove all unconnected input and output places as sketched in Sect. 2.3

It is reasonable to visualize this process translation by putting the instantiated Petri net patterns for each node of the process inside the inner box of the process pattern of Fig. 2.16.

To instantiate the process in terms of the Petri net semantics, put one token on each (process) input place. Then firing enabled Petri net transitions yields the possible executions of the process. An execution terminates properly if it reaches a marking m where one of the following conditions holds:

1. If m puts a token on one of the stop-places **process.stop.s**, then each process output place has one token.
2. Marking m puts a token on each of the process output places, and all other places of the net have no token.

These concepts, the translation algorithm and the criterion for proper termination conclude the basic formal semantics for UML2-AD models.

3 Advanced Concepts and Patterns

In the previous section, we provided the basic formal semantics for UML2-AD models. The basic semantics contains the minimal core of UML2-AD and already allows for specifying fairly complex processes. However, UML2-AD allow extensions of these core concepts and provides further concepts, which are used in practice and supported by the IBM WebSphere Business Modeler. In this section, we present extensions of the core concepts, mainly constituted by the concept of *pinsets*. We do not consider new node types like broadcast communication or repositories.

3.1 Pinsets

The input pins and the output pins of processes and of tasks are covered by *pinsets*, called *input pinsets* and *output pinsets*. Each input pinset constitutes a subset of the task's (or process') input pins. Only the pins of one input pinset must have a token in order to enable a task (process). In the standard case considered in Sect. 2, each task (process) has exactly one input pinset containing all input pins, and one output pinset containing all output pins.

Upon execution, the task (process) only consumes the tokens that caused its enabling, i.e. only the tokens from the enabling pinset. Likewise, the result of the execution of a task (process) only puts tokens in one of the output pinsets while the other pins remain empty.

Further, input pinsets and output pinsets can be related to each other to specify input/output behavior of a task/process at an abstract level. Figure 3.1 displays an example specification of two input pinsets and two output pinsets in the IBM WebSphere Business Modeler. Enabling via the first input pinset causes the task (process) to terminate with tokens on the first output pinset.

3.1.1 Processes with Pinsets

In order to formalize the enabling and execution of tasks with pinsets, we have to adapt the pattern for processes (see Fig. 2.15). Instead of one transition that replicates the data input of the process, we create one transition for each input pinset that produces a token on the corresponding start places `process.start.i`.

We further remember which input pinset, e.g. `input.setX`, did enable the process by a dedicated place, e.g. `setX.fired`. This place is used later to control which output pinsets may fire.

Likewise, we add a transition for each output pinset, e.g. `output.setY`, that replicates the output tokens of the process. The firing of `output.setY` is made dependent on place

Name	Input	Input:2	Input:3	Criterion
Input Criterion	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	In1 AND In2
OR Input Criterion:2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	In2 AND In3

Name	Prob...	Out1	Out2	Out3	Criterion
Output Criterion		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Out1 AND Out2
OR Output Criterion:2		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Out2 AND Out3

IBM WebSphere Business Modeler notation, input and output pinsets

Name	Criterion
Output Criterion	Out1 AND Out2
Output Criterion:2	Out2 AND Out3

Details

Name
Output Criterion

Regular
 Exceptional output
 Streaming (can send output while th

Associated input criteria

Input Criterion
 Input Criterion:2

IBM WebSphere Business Modeler notation relating input and output pinsets

Figure 3.1: Pin sets example specification

setX.fired iff the pinset specification describes that enabling the process via setX leads to termination via setY. Figure 3.2 depicts the pattern that implements the pinset specification of Fig. 3.1 for a process with one start and one stop place, we omit the complete parameterized pattern for the sake of understandability.

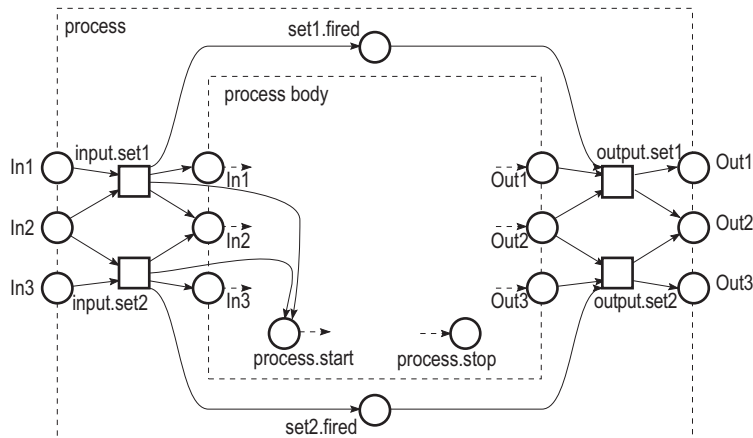


Figure 3.2: A wrapper for a process that has overlapping pin sets as specified in Fig. ??

This pattern can easily be extended to incorporate the feature that an input pinset may enable only some start nodes (instead of all) while a specific stop node requires termination via a specific output pinset. This is achieved by omitting corresponding arcs from input pinset transitions to start nodes and introducing corresponding arcs from stop nodes to output pinset transitions.

Observe that the pattern is correct wrt. the UML2-AD semantics only if there is no input pinset that strictly contains another pinset, because the UML2-AD semantics

requires that only the largest pinset fires. This priority of transitions is not expressible in the standard Petri net semantics. Using inhibitor arcs that prevents enabling of the smaller pinset if the larger pinset is enabled, would solve the problem [1].

3.1.2 Proper Termination of Processes with Pinsets

The use of (output) pinsets requires to refine the notion of a proper termination of a process. An execution of a process terminates properly if it reaches a marking m where one of the following conditions holds:

1. If m puts a token on one of the stop-places `process.stop.s`, then the output places of exactly one output pinset have a token.
2. Marking m puts a token on each output places of exactly one output pinset, and all other places of the net have no token.

The first condition can be refined further to require that a specific output pinset is marked in case a corresponding relation between stop nodes and output pinsets has been specified.

3.1.3 Tasks with Pinsets

In order to formalize the enabling and execution of tasks with pinsets, we adapt the pattern for processes by omitting all internal behavior and keeping only its “shell”. Figure 3.3 depicts the resulting Petri net pattern.

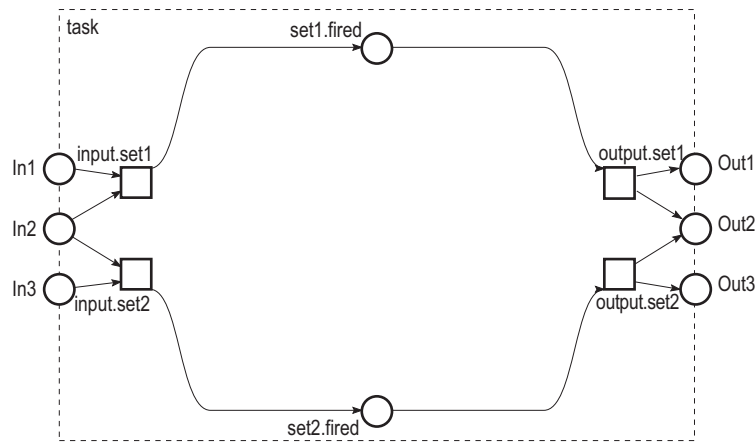


Figure 3.3: A task that has overlapping pin sets as specified in Fig. 3.1

For the purpose of analyzing UML2-AD processes, this abstraction is sound as it preserves the flow of tokens wrt. incoming and outgoing tokens. In case a task has some specific semantics associated to it, the pattern could be extended to represent this behavior that is enabled by any of the `input.setX` transitions and terminates via any of the `output.setY` transitions.

3.2 Optional Pins

UML2-AD allow to make pins of tasks and processes variable in an execution by specifying minimum and maximum values of tokens to be produced/consumed. Here, we consider the interesting case of $min = 0$ and $max = 1$.

An input pin with $min = 0$ and $max = 1$ means that a task is enabled if the pin carries one or zero tokens. In case the task is enabled, it must consume all available tokens on all pins of the enabling pinset. Conversely, an output pin with $min = 0$ and $max = 1$ means that the execution yields one or zero tokens on the output pin (if its output pinset was chosen).

In order to express optional input pins of an activity, we need inhibitor arcs (or complementary places) to detect whether an input pin has *no* token. Figure 3.4 depicts an example of a task with one optional input pin and one optional output pin. Again, we omit the general parameterized pattern for the sake of understandability.

In Fig. 3.4, the arcs from Input2 to task.3 and task.4 are inhibitor arcs specifying that task.3 and task.4 are enabled only if Input2 has no token. The pattern of Fig. ?? is combinatorial to relate all feasible input markings (which enable the activity) with all feasible output markings (which are produced by the activity).

Name	Associated data	Minimum	Maximum	Input source
Input				
Input:2	String	0	1	Flow

Name	Associated data	Minimum	Maximum	Input source
Input				
Input:2	String	0	1	Flow

IBM WebSphere Business Modeler
notation, optional input and output pins
of a task

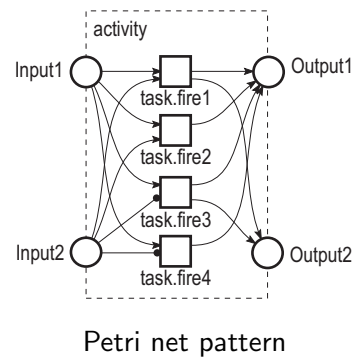


Figure 3.4: Task with an optional input pin and an optional output pin

This pattern can be combined with the pattern for input pinset and output pinsets. In this case, we need for each enabling variant of each input pinset a unique transition, and for each variant of each output pinset (wrt. pin multiplicities) a unique transition. Input transition variants and output transition variants have to be related to each other according to the pinset pattern of Fig. 3.3.

To cover more elaborate pin multiplicities like $min = 3$ and $max = 17$ we would also need arc weights and reset arcs (or complementary places) to faithfully capture the enabling and termination of tasks and processes [1]. For the moment, we do not translate such processes.

4 Variants of Patterns

In this section, we present some variants of the patterns that we have introduced in the preceding sections.

4.1 Edges by Merging Places

Previously, we translated UML2-AD edges by adding dedicated transitions. An alternative pattern is to merge places: If an output pin and an input pin are connected by an edge, the corresponding output place and input place of the Petri net fragments are merged. In case an input or output pin of a node is not connected to another pin, we omit the corresponding place in the translation. Figure 4.1 and Figure 4.1 illustrate the matter.

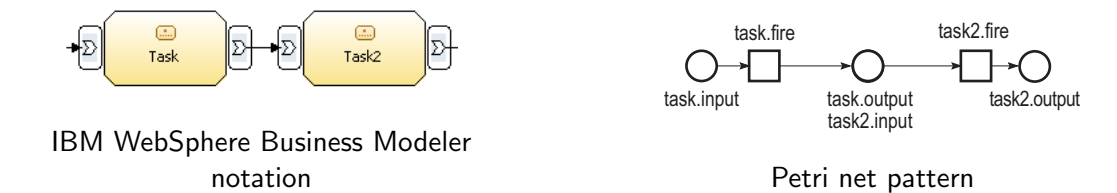


Figure 4.1: An edge is translated by merging places

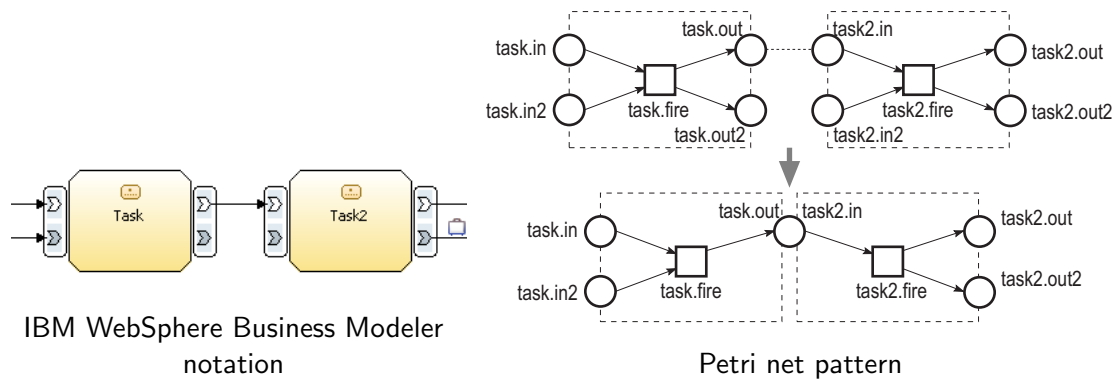
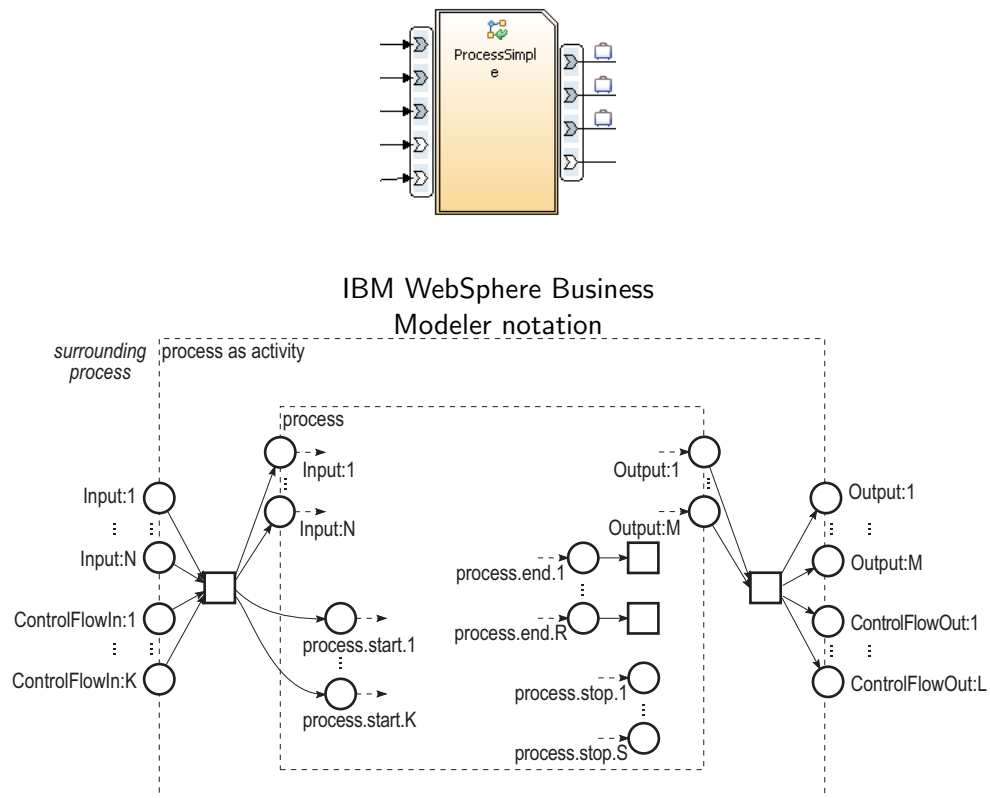


Figure 4.2: Unconnected pins are removed in the translation

4.2 Call to Local Processes

A process can call another (sub-)process during its execution by a dedicated task. In the IBM WebSphere Business Modeler, the called process can be a global process that can be called from any other process, or it is a local subprocess, only to be called by its parent. Thus a local subprocess is a means to hierarchically structure complex processes by grouping tasks in a subprocess. For the purpose of analysis, this hierarchy can be flattened into a single, large Petri net.

The Petri net pattern that formalizes the call and flattens the hierarchy is an adaptation of the process pattern, also allowing for control-flow input and output places, see Fig. 4.3.



Petri net pattern modeling a process that is called as an activity within another process

Figure 4.3: Call to a local process

5 Conclusion

In this document, we presented a pattern-based Petri net semantics for the core features of a variant of UML2 Activity Diagrams that is implemented in the IBM WebSphere Business Modeler. We presented basic patterns for tasks, control-flow nodes and processes, and the basic composition principle. We showed extensions of the patterns that incorporate the more involved features of pinsets and pin multiplicities. Our approach thereby followed existing works for formalizing UML2-AD, [5, 6, 7].

The aim of our formalization is to analyze IBM WebSphere Business Modeler models for control-flow errors. To this end, we provided structurally simple patterns which we supplemented with a specification of properly terminating behavior. We implemented a UML2-AD-to-Petri nets compiler that allows us to verify IBM WebSphere Business Modeler models with model-checking techniques as follows: A IBM WebSphere Business Modeler process library can be exported into an XML format. The compiler translates the entire library into a set of Petri nets using the patterns specified above; the standard output of our compiler is the input format of the model checker LoLA [8].

The compiler also generates a CTL formula that formalizes the proper termination: The specification of proper termination as described in Sect. 3.1.2 can be formalized as a state predicate ψ over the places of the constructed Petri net. A process is free of control-flow errors iff this state predicate is always reachable, i.e. if the CTL formula $AG EF \psi$ holds in the net. This can be checked using LoLA.

The compiler that translates IBM WebSphere Business Modeler XML export to Petri nets is available at <http://www.service-technology.org/uml2owfn/>.

Bibliography

- [1] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *ICALP'98*, volume 1443 of *LNCS*, pages 103–115, 1998.
- [2] Object Management Group. UML superstructure, v2.1.2, formal/07-11-02. Standard, 2007.
- [3] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, 1985.
- [4] P. Starke. *Analyse von Petri-Netz Modellen*. Teubner, Stuttgart, 1990.
- [5] Harald Störrle. Semantics of control-flow in uml 2.0 activities. In *VL/HCC*, pages 235–242, 2004.
- [6] Harald Störrle. Semantics and verification of data flow in UML 2.0 activities. *Electr. Notes Theor. Comput. Sci.*, 127(4):35–52, 2005.
- [7] Harald Störrle and Jan Hendrik Hausmann. Towards a formal semantics of uml 2.0 activities. In *Software Engineering*, pages 117–128, 2005.
- [8] Karsten Wolf. Generating Petri net state spaces. In *ICATPN*, volume 4546 of *LNCS*, pages 29–42. Springer, 2007.