# ACTIVITYDIAGRAM2PETRINET : TRANSFORMATION-BASED MODEL IN ACCORDANCE WITH THE OMG SYSML SPECIFICATIONS

Damien Foures, Vincent Albert, Jean-Claude Pascal

# ACTIVITYDIAGRAM2PETRINET : TRANSFORMATION-BASED MODEL IN ACCORDANCE WITH THE OMG SYSML SPECIFICATIONS

Damien Foures, Vincent Albert, Jean-Claude Pascal
CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse Cedex 4, France
University of Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France
{dfoures, valbert, jcp}@laas.fr

## KEYWORDS

MDE, ATL, TINA, transformation, verification, Petri Nets, SysML, OMG, VHDL-AMS, Ecore

## ABSTRACT

This study aims to automate the transformation of activity diagrams (AD) to Petri nets (PN). Based on specifications given by the Object Management Group (OMG), we have established transformation rules in ATLAS Transformation Language (ATL) to obtain a model consistent with our Petri Net meta-model (model2model). The semantic of Activity Diagram was verified with PetriNet2Tina transformation (model2text) and has allowed us to verify that was the same in the corresponding PN. This verification is done with the "model-checker" TIme petri Net Analyzer (TINA) and Linear Temporal Logic (LTL) language. The user needs only to set up the Activity Diagram from the stakeholder requirements; the transformation and verification is automatic. PetriNet formalism could enable us to provide valuable information on a Activity Diagram, to execute and simulate it.

## INTRODUCTION

### Context

The present work is based on the general context of systems engineering and integration of heterogeneous systems. These systems have software and hardware components, they generally hold high real-time constraints and disciplines (electrical, mechanical, information, hydraulic...). We seek to propose methods and tools for controlling the development cycle of such systems. The use of models and simulation is becoming dominant component in the development cycle, and we seek to improve (and eventually to automate) their use. The use of meta-modeling moves in this direction as it aims to make the modeling. Based on the instantiation of the meta-model, more clearer and less ambiguous.

### Approach

Our approach uses the concepts advocated by the OMG through Model Driven Architecture (MDA), itself based on modeling and automatic transformation of models into other models. Presently we have the TINA formalism (LAAS 2011), TINA toolbox (LAAS 2011), and a transformation procedure from PN to VHDL-AMS (Albert et al. Octobre 2005). TINA formalism allows us to verify formally that Activity Diagram properties are preserved, using its model-checking tool (selt). The transformation to VHDL-AMS (IEEE 1999) allows us to propose a simulation phase, commonly called virtual prototyping. The addition of these two approaches allows us to validate the discrete and continuous part of the activity diagrams, and hence predict functional characteristics of the system.
In this work, we begin with a method to create meta-models of AD and PN. We will see a suggestion of concept mapping and its verification, and finally a simple example of transformation.
Till date transformation from State Machine to PetriNet were made (Bernardi et al. 2002), (Campos and Merseguer 2006), but Activity Diagrams (ADs) accentuate the internal control and data flow of systems. Further work on ADs were made (Bonhomme et al. 2008), (López-Grao et al. 2004) or (Thierry-Mieg and LHillah 2008) but do not take into account many properties of ADs.

## SYSML PRESENTATION

SysML is a graphical modeling language developed by OMG and INCOSE. SysML is a UML profile adapted to systems engineering emerged in the 2000s. It can model the behavior of systems (continuous and discrete), with a hierarchical approach. OMG SysML specifications appeared only in 2007. We extracted the meta-model from (OMG 2010c) and (OMG 2010b) .

### Activity Diagram

The activity diagram is one of the four behavioral diagrams included in SysML. They are useful to describe a hierarchical behavior, delayed, or a mixed systems. The

TOPCASED framework permits to describe graphically all AD in accordance with the AD meta-model, itself in accordance with the Ecore(Budinsky et al. 2003) meta-meta-model. The Meta-Object Facility (MOF) is preconized by OMG, but Ecore is more or less aligned on Essential MOF (EMOF).

### Activity Diagram Meta-model

The AD meta-model was extracted from OMG specification as described therein, without addition. Small part of it with ControlNode meta-class is shown in figure 1. We see that *FlowFinalNode* and *ActivityFinalNode* inherit *FinalNode*. The *FinalNode* inherits to *ControlNode* as *InitialNode,* similarly *ForkNode, JoinNode* and *MergeNode*. To create the AD meta-model, it is easier to begin with basic node of AD, directly afterwards all properties, all links with other meta-class were extracted. The biggest difficulty was to know where to stop the meta-model extraction. Indeed, in (OMG 2010c) and (OMG 2010b) all classes inherit from many other classes from SysML and UML metamodels. Subsequently it uses the AD meta-model from TOPCASED [1] framework. First, we have verified compliance with OMG specification for parts we have used.

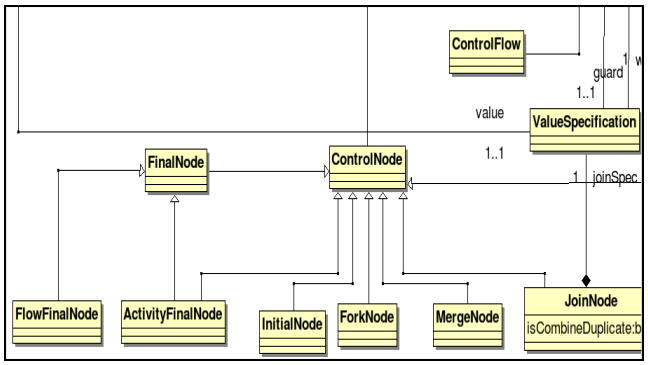


Figure 1: Part of Complete Activity Diagram in accordance with OMG specification.

## PETRINET PRESENTATION

A PetriNet is a mathematical modeling language. There are currently a lot of Petri nets classes. Gradually basic, hierarchical and differential predicate transition Petri nets, will be transformed to, the control part, the hierarchy, and finally the continuous part of the activity diagram. A Petri net is composed of places, transitions, and arcs. Arcs connect a place to a transition or a transition to place, others possibilities are forbidden. This kind of constraint must appear in the meta-model of PN.

[1] http://www.topcased.org/

### PetriNet Meta-model

The PN meta-model established in (Albert et al. Octobre 2005) was adapted to this new work. Macro-place and macro-transition were removed because they are restrictive. For example, if an ActionNode of AD is transformed into macro-place it is impossible to put new value in this macroplace during execution. During execution macro-node becomes totally independant, so we decided to work flat. Flat PN, without hierarchy, are more easy to master communication links. TINA works also on only one abstraction level.
Figure 2 shows a simplified version of PN meta-model. We can read on it: *PetriNet* is composed of *Node* and *ArcClassic*. A node can be a *Transition* or a *Place* and they are linked with *ArcClassic*. A node can have multiple *incoming* or *outgoing ArcClassic*s. An *ArcClassic* can only have one *Source Node* and one *Target Node*. This interpretation includes a description of the previous paragraph. However, constraints do not appear, they must be expressed, for example in Object COnstraint Language (OCL)(OMG 2010a).

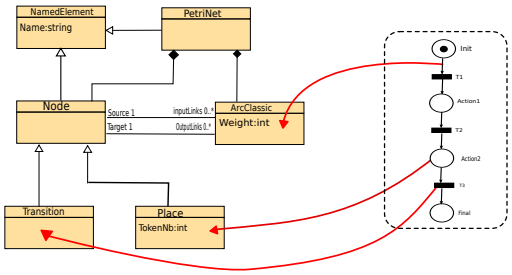


Figure 2: Petri Net model conforms to simple Petri Net meta-model.

## TRANSFORMATION WITH ATL AND ECLIPSE MODELLING FRAMEWORK (EMF)

Initially, our work was to be, totally in accordance with OMG. Tools for model transformation suggested by the OMG are still evolving, and to date we prefer to use EMF with Ecore meta meta-model and ATL language which seems to be the best choice, with a framework that has been already tried and tested. Our transformation choices are pointed out in figure 3.

### Mapping of Concepts

The original contribution of our transformation is to match an activity diagram artefact to a PN block which will preserve the AD semantic, related to this artefact as defined by OMG. Such a PN block must also handle alternatives in AD modeling, e.g. an input pin may be stereotyped "optional" and becomes useless to start the activity. Table 4 illustrates the main mapping.

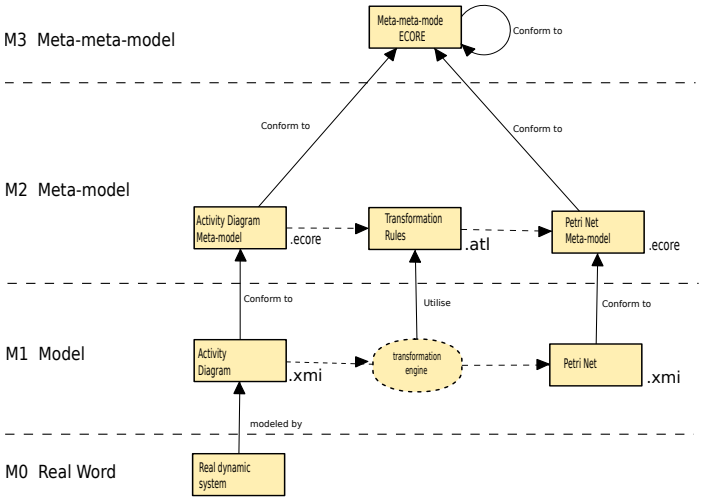


Figure 3: Meta-modelling Transformation.

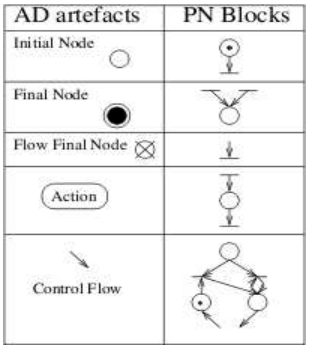


Figure 4: Basic Concepts Mapping: from AD to PN

These design choices, reflects the analysis based on the generality of blocks (SendSignalAction or CallBehaviorAction inherited from Action),on block interconnection facility but also on properties defined by (OMG 2010c) and (OMG 2010b). For example, a ControlFlow can be modelised as a single transition (Thierry-Mieg and LHillah 2008), it can be also included on nodes like in (Bonhomme et al. 2008). In figure 4 the ultimate PN block acts as a buffer to respect ControlFlow properties written in OMG specification. They define ControlFlow like an edge that starts an activity node after the previous one is finished, with this simple definition an PetriNet arc is a correct model, but OMG add many specification on ControlFlow or which influences behavior. Finally, PetriNet arc is inadequate to meet all properties. Let's look at an excerpt of the properties and define possible solutions to respect them:
Property 1 (from ControlFlow): *Tokens offered by the source node are all offered to the target node.*
Property 2 (from ActionNode): *When an action accepts the offer for control and object tokens, the tokens are removed from the original sources that offered them. If multiple control tokens are available on a single incoming control flow, they are all consumed.*

Solutions: We can do with the property 1 that the first intuition is good, PetriNet arc carry tokens too. Property 2 and many others shows that the ControlFlow has behavior of token storage like a PetriNet place. The inability to know dynamically the number of token in a place to empty correctly ControlFlow brings us to the model as a buffer. Indeed, the presence of token is important but not the token multiplicity.
The same work was done with almost every ActivityDiagram node. Many stereotypes can be applied to nodes and was not considered to date.
The transition from one column to another in figure 4 is possible at M2 level (see figure 3 ) with ATL rules and Eclipse Modeling Framework. We'll see how we built a block PN during a small example.

### Complex Petri Net

It was already seen how to build an atomic block. Building complex PN is relatively simple, in an activity diagram every or almost every node are connected to another by "ControlFlow" or "ObjectFlow". They will just have to connect each atomic block (can be viewed as:Transition-Place-Transition) to controlflow or objectflow block (can be viewed as:Arc-Place-Arc).We remind the reader that, analysis at model level should be higher than meta-model level to establish the rules in MDE context. Using the hierarchy can significantly reduce the amount of transformation rules. With AD2PN transformation, we could see that ATL cannot use easily the advantages of hierarchy. The language must be well controlled to limit significantly the coding rules.

## VERIFICATION

After establishing the rules for "control flow" part, it is important to verify formally the transformation and, thus, verify that the PN had the same behavior as the activity diagram. In other words, it must check, through PN, to find the operational semantics of an AD. Subsequently, it is possible to imagine that users adds constraints (OCL) to the model, their validity in the PN can be proved with verification.

### ResolveTemp Meta-model

Each PN block can be reduced to a sequence, Transition-Place-Transition. This meta-model defines each type of block to give essential features, but no behavior. It performs double transformation AD2PN and synchronized AD2ResolveTemp. This is to retrieve the name of input transitions (isStarted), output transitions (isFinished), running place (isRunning) and this incoming/outgoing (incomming/outgoing) (see figure 5). Sometimes attributes are added to define better LTL property (isNotRunning, optionnalIncoming,...)

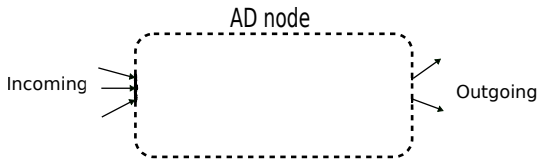


Figure 5: PN block definition

### LTL Properties and selt

Owing to lack of space, we will not present LTL language. Our approach has been to develop, properties in blocks with properties with inputs and outputs. In accordance with transitivity relationship A ⇒ B and B ⇒ C then A ⇒ C. If block satisfies this properties, and if properties with connected blocks are satisfied then entire PN is verified. This verifies formally correct construction of the Petri net. This technique shows limitations indeed to have the expected Petri net (no problem in the construction). But it does not involve checking of correct behavior of the Petri net. If the building blocks have a limited or incorrect behavior, the model will be wrong and yet the verification will be positive. The user must know the limits of model transformation used.

## EXAMPLE: ACTIVITY DIAGRAM TO PETRI NET

We can see in figure 6, at left, a simple AD. When *Activity1* is running, executes *Action1*, when this one is finished, starts parallel execution of both action (*action 2 and 3*). If *action2* is finished, *action 3* can run, but if *action 3* is the first to finished, at this moment, *Activity1* is stoped, and all action in *Activity1* are stopped. We can see figure 6, at right, the resulting petri net after AD2PN transformation. It is more complex in appearance, it takes the behavior of Activity Diagram, that part is not really readable but does not provide specific information aditional. To make this transformation, we must establish rules for each meta-class present through these instances in the model. For example: The ATL transformation rule for InitialNode meta-class.

```
rule initialnode_place{
from a:MMAD!InitialNode
to    b:MMH!Place (
   Name<-'p_Initial_'+a.name+'_'+a.activity.name,
   OutputLink<-c,
   ...
   ),
     c:MMH!ArcClassic(
   Name<-'a2_Initial_'+...
   ),
     d:MMH!Transition(
   OutputLink<-a.outgoing,
   Name<-...
   )
}
```

For each instance of InitialNode, a marked place is created, arc connects the latter to a transition. This transition is associated with meta-class instance after transformation, present in outgoing InitialNode argument ( Outputlink←a.outgoing). This graphical version under TINA is possible with a second transformation, from Petri Net to Tina (model2text). To make this transformation we have used "Query"(LINA-INRIA 2006) from ATL :

```
helper context Hiles!Transition def:genTransition():
String = 'tr ' + self.Name + ' [0,w[ '
  + self.InputLink->iterate(arc;accPlAm:String=''|
           accPlAm+arc.Source.Name +' ') + ' -> '
  + self.OutputLink->iterate(arc;accPlAv:String=''|
           accPlAv+arc.Target.Name +' ') + '\n' ;
```

On this part of "Query", it is automatically generated the "arc part" of tina text. When the equivalent PN is implemented in TINA toolbox, we use "selt" and its model-cheking tool. To generate automatically properties in LTL language, we use an other transformation: from ResolveTemp to LTL (model2text) and other "Query"(LINA-INRIA 2006):

```
helper context ResolveTemp!RTCF def:getPCF():String =
'[]('+self.isRunning+'+'+self.isNotRunning+'=1);\n'+...
```

The automated property created after this query is about ControlFlow and verifies invariant under block,the label '[]' means that this invariant must always be true to validate this property.

```
[](p2_CFlow_CF1_Activity1+p3_CFlow_CF1_Activity1= 1);
```

The OMG specification says:"*If multiple control tokens are available on a single incoming control flow, they are all consumed.*"(OMG 2010c).To respect this semantic, controlflow is modelised as seen in concepts of mapping subsection (presence or exclusively absence of tokken in ControlFlow).

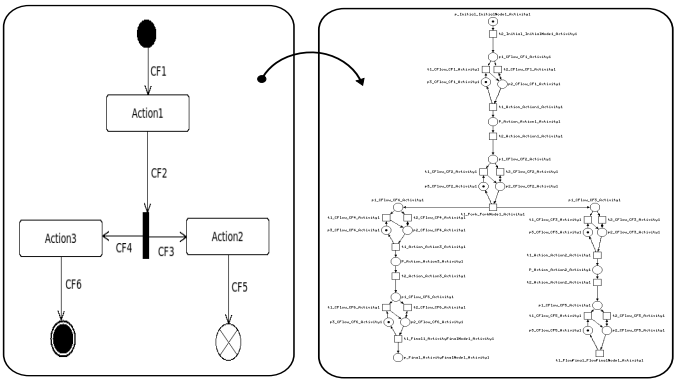


Figure 6: Simple AD2PN example

## CONCLUSION AND FUTURE WORK

This work, has already set up a complex management of AD in accordance with the OMG specifications. OMG Specifications regularly contrain to refrain some shortcuts, which could simplify development. PN provides a mathematical formalism which is, at this step of the project not really exploited. It should eventually allow to highlight the invariant properties in AD, execute activity diagram in parallel with VHDL-AMS, to cover all the possibilities offered by the activity diagrams.
The future work will use differential predicate transition PN(Genrich 1987) for dataflow management. Today this first step can say if data are presents or not, but it does not convey any information. With this new class of petri nets, the TINA formalism becomes unuseful and we'll go to validation by simulation (VHDL-AMS). During this work, it was also possible to show that management of interrupts areas is possible with basic PN.
We express concern about management of many changes which allows users (stereotype, optional attribute,...), to manage all of these cases seems to overload rules of transformation. ATL language has sometimes seemed a bit complex, it will be interesting to see the contribution of Query / View / Transformation (QVT) language. The development of the reverse chain; Petri nets to activity diagram in this work will be completed by providing the user with an AD modified or re-organized according to the invariant or error detected in PN. Verification should be transparent to the user, however, a good mastery of AD seems crucial to create a really exploitable PN for us.

## REFERENCES


Albert V.; Nketsa A.; and Pascal J.C., Octobre 2005. *Towards a metal-model based approach for hierarchical Petri net transformations to VHDL. European Simulation and Modelling Conference, Porto.*

Bernardi S.; Donatelli S.; and Merseguer J., 2002. *From UML sequence diagrams and statecharts to analysable petri net models.* In *Proceedings of the 3rd international workshop on Software and performance*. ACM, New York, NY, USA, WOSP '02. ISBN 1-58113-563-7. URL `http://doi.acm.org/10.1145/584369.584376`.

Berthomieu B.; Vernadat F.; and Ribet P.O., 2000. *Manual Reference Pages - selt (n).*

Bonhomme S.; Campo E.; Estve D.; and Guennec J., 2008. *Methodology and Tools for the Design and Verification of a Smart Management System for Home Comfort. 4th International IEEE Conference "Intelligent Systems".*

Budinsky F.; Brodsky S.A.; and Merks E., 2003. *Eclipse Modeling Framework*. Pearson Education. ISBN 0131425420.

Campos J. and Merseguer J., 2006. *On the Integration of UML and Petri Nets in Software Development.* In *Application and Theory of Petri Nets*. 19–36. doi:10.1007/11767589_2.

Genrich H.J., 1987. *Predicate/Transition Nets.* In *Lecture Notes in Computer Science*.

Gomez C.; Pascal J.C.; and Esteban P., 2010. *From Embedded Systems Requirements to Physical Representation: A Model-based Methodology in Accordance with the EIA-632 Standard.*

IEEE, 1999. *Institute of Electrical and Electronics Engineers Standard VHDL Analog and Mixed-Signal Extensions.* , no. IEEE Std 1076.1-1999.

LAAS, 2011. *http://homepages.laas.fr/bernard/tina/.*

LINA-INRIA A., 2006. *ATL:Atlas Transformation Language ATL User Manual. OMG specification*, , no. version 0.7.

López-Grao J.P.; Merseguer J.; and Campos J., 2004. *From UML activity diagrams to Stochastic Petri nets: application to software performance engineering. SIGSOFT Softw Eng Notes*, 29, 25–36. ISSN 0163-5948. doi:http://doi.acm.org/10.1145/974043.974048. URL `http://doi.acm.org/10.1145/974043.974048`.

OMG, 2010a. *OMG Object Constraint Language (OCL), Superstructure. OMG specification*, , no. 2.3, 1–256.

OMG, 2010b. *OMG Systems Modeling Language (OMG SysML). OMG specification*, , no. 1.2, 1–246.

OMG, 2010c. *OMG Unified Modeling Language(OMG UML), Superstructure. OMG specification*, , no. 2.3, 1–742.

Thierry-Mieg Y. and LHillah o.M., 2008. *UML behavioral consistency checking using instantiable Petri nets. ISSE*, 4, no. 3, 293–300.