PetriFlow: A Petri Net Based Framework for Modelling and Control of Workflow Processes

Martin Riesz, Milan Baláž and Gabriel Juhás

Faculty of Electrical Engineering and Information Technology
Slovak University of Technology
Ilkovičova 3, 812 19 Bratislava, Slovak Republic

Abstract:

The paper presents an architecture of a Petri net based framework for modelling and control of workflow processes. It focuses on the PNEditor module and briefly discusses workflow engine based on the models designed using the PNEditor.

1. Introduction

There are many different Petri net tools and Petri net based tools for modelling workflow processes, such as CPN ([GA05]), Viptool ([BDJL06], [DJLN03]) or ProM ([DMVWA05]), to mention just some of them. The question arises why to implement another one. Most of them are determined to create models and some of them to analyze the models. However, models are only the first step in bussines process management systems (BPMS). The main advantage of BPMS is that the models can be used to control the workflow process according to the designed model using a workflow engine. The problem of the most existing editors (let us just mention Viptool as a prominent example), is that they were imlemented with different aims, mostly to analyse the model, but they do not provide all the information needed for the generation of a deployable application, such as resource management. Another problem is with the case generation. Usually, a model obtained via a Petri net modelling tool can be understood as a general definition of a model of a process, while the single cases can be understood as instances of the process. Using an analogy with object-oriented programming, a model can be understood as a class, while single cases can be understood as objects of that class. In Petri net based modelling tools, the realization of cases is often done using coloured Petri nets ([GA05]). But in such tools, colours are used both for distinguishing cases from each other as well as for modelling the data of the cases. Another imortant feature for succesfull application of models is hierarchy, which enables not only model on different level of abstraction but also to deploy reusable components. This is important for practical use, as the bussiness consultants, which design models often do not offer models of sufficient level of details. Many tools offer different kinds of hierachical nets, but because they mostly implement also a semantical framework with aim to preserve some properties, they are mostly too complex and/or too restrictive to generate a deployable code.

2. PetriFlow: A brief introduction

For the above mentioned reason, we develop a new framework for modelling and control workflow processes based on Petri nets. Presently, it consists of two modules, PNEditor and PNEngine.

PNEditor enables to design a model, which is basically a place/transition nets enriched with the feature for distinguishing between static and dynamic places, where static places correspond to static variables (they exists once for a process), while dynamic places are constructed for each case. Moreover PNEditor enables modelling with subnets, where subnets represents only a visual tool for designer, i.e. on semantical level, PNEditor works with a flat place/transition nets. Another important feature of PNEditor is that it provide a resource management on a very simple and intuitive way. It enables to create roles, where a role is

basically a subset of the set of transitions, determining which transitions (which tasks) can be performed by users having the role.

The development version of the PetriFlow PNEditor can be downloaded via: http://matmas.e2net.org/petriflow/pneditor/

PNEngine is a light version of a workflow engine based on the models provided by PNEditor, but also by place/transition nets imported in PNML format. In present states it does not include users management, but only enables to control processes modelled by standard p/t nets. It includes a parser from PNML, which produces an SQL script for a given net and inserts it into an SQL database. Further, it enables to create new cases for a given process and to control the cases processing according to the business logic given by the Petri net modelling the process. The business logic layer is implemented in J2EE, the connection with the database and persistance of the cases is realized using Hibernate. User interface is realized via Java Server Pages. The PNEngine is running on a Tomcat server. It enables the users to perform an activity from the tasklist for actually processed cased, i.e. to fire enabled transitions of the correspondent copy of the net, via an internet browser. After a user perform an activity from the tasklist for a given case, the corresponding transition is fired, the new marking is computed and the takslist is updated.

3. PNEditor

PNEditor offers usual features of a graphical editor for design of place/transition nets. It enables to draw place/transition nets, i.e. labeled places, transitions, weighted arcs and markings with multiple tokens per place. The further functionality is saving the net to a file, definition of roles, definition of subnets (nested nets), saving of predefined subnets to files and their reuse as reusable components, replacement of subnets, definition of static places, which exists once per process, and other standard features, such as unlimited undo-redo actions.

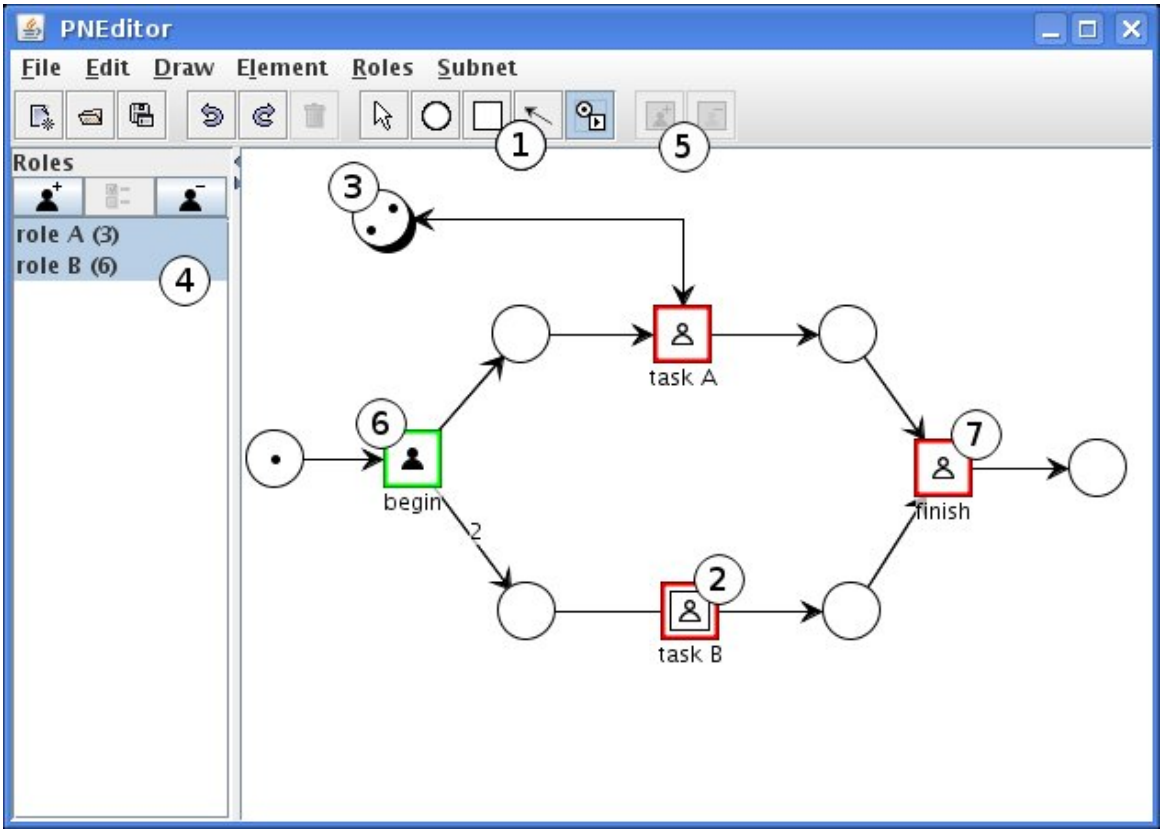

Figure 1: Illustration of the key features of the PetriFlow PNEditor

For better illustration, Figure 1 gives an owerview of the functionality of the PetriFlow PNEditor module:
1) Drawing tool selection: from left: object selection tool, inserting places, inserting transitions, arc drawing, adding/removing tokens/transition firing
2) Square with double border represents subnet
3) Place with shadow represents static place
4) Panel with roles: buttons from left: add a role, edit role properties, delete a role.
- role A contains set of transitions: begin, task A, finish (total of 3)
- role B contains set of transitions: begin and all nested transitions in subnet task B (total of 6)
5) Buttons for adding or removing transitions from currently selected roles
6) Both roles are selected so the information icons are displayed on top of the transitions in diagram: black person icon on the transition describes situation in which all selected roles contain this transition
7) white person icon on the transition describes situation in which only some selected roles contain this transition

4. Subnets in PetriFlow PNEditor

Usually, business modeller models task as atomic transitions. On a more detailed level, typically a task can be started, finished, paused, be continued or cancelled. Each of such tasks can be illustrated with part of Petri net given in Figure 2, which can be understood as a subnet on a more detailed level and shoud be used as a reusable component.

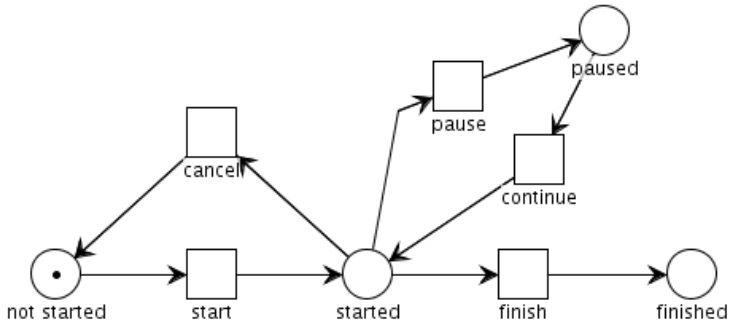

Figure 2

Place "not started" in figure 2 describes a state, in which the task has not yet been started and "finished" describes the state reached after the task is finished.

It would be very practical to model workflow process using such reusable components and in general to give modeller an option to desing and use his own reusable components possibly without any semantical restricition.

Anoher situation where reusable components represents a desirable feature is in case of complex prcesses assembled from relatively independent units with exactly defined inputs and outputs. PNEditor supports subnets allowing each subnet to have more nested subnets so that the whole hierarchy can be build up in a place/transition net.

In the case the user creates a workflow model where the tasks will be represented by transitions, PNEditor gives a choice of replacing a transition by a subnet. The subnets can be replaced by existing stored subnets. In this way, individual transitions can be converted to custom subnets representing reusable components.

In order to explain the concept of subnets, we have to recall some basic definitions of place/transition nets (for more details see e.g. [Rei86]). Given a place/transition net $N = (P,T,F,W)$, where P is a finite set of places, T is a finite number of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs (i.e. the flow relation) and $W: F \rightarrow N$ is the weight function (N

denoting nonnegative integers), we say that a subnet of N is any net N‘ = (P‘, T‘, F‘, W‘) where P‘ $\subseteq$ P, T $\subseteq$ T, the flow relation F‘ = F $\cap$ ((P‘ $\times$ T‘) $\cup$ (T‘ $\times$ P‘)) and W‘ = W|F‘. Moreover, we consider only proper subnets, i.e. subnets satisfying the following condition for each p $\in$ P and each t $\in$ T‘: ((p,t) $\in$ F $\vee$ (p,t) $\in$ F) $\Rightarrow$ p $\in$ P‘. For clarity of the text let us define the interface of a proper subnet as the set of places p $\in$ P‘ which are connected with a transition which does not belong to T‘. In PNEditor, interface places are graphically expressed using dashed places. On one abstraction level, a subnet is visualized via interface places connected with the squere with double border via reference edges. These edges can have one of three appearances:

1) dashed edge - the interface place is not connected with any transition in the subnet.
2) In the case that the interface place is connected in the subnet with exactly one transition using an arc, the reference edge takes direction of the arc.
3) In the case that the interface place is connected in the subnet to more than one transition, reference edge is undirected, i.e. it is displayed by a reference edge without arrow.

Neighbourhood of a place p $\in$ P w.r.t. the net N is a subnet N_p = ({p}, T_p, F_p, W_p) of N with the set of places formed by the place itself and the set of transitions T_p = {t $\in$ T | (t,p) $\in$ F $\vee$ (p,F) $\in$ F} formed by the the union of the preset and the postset of the place p in the net N, i.e. by surrounding transitions of p in net N.

Recall that two place/transition nets are isomorphic, when there exists a bijective mapping between the sets of places and a bijective mapping between the sets of transitions, which preserve arcs and their weights. We say that a a place p in a place/transition net is said unique place of the net, if there is no place p’ in the net with the isomorphic neighbourhood.

In PNEditor, identities of the interface places are not saved when storing a subnet to make it a reusable component, i.e. a subnet is stored just as an ordinary net with an additional information which places forms its interface. When replacing one subnet by another stored subnet, the interface places of the replaced subnet are identified with the interface places of the stored replacing net according to the following rules:

1) In the first place, only a unique interface places are identified: A unique interface place p of the replaced subnet is considered to be equal to a unique interface place p‘ of the stored replacing subnet, if the neighbourhood of p w.r.t. the replaced subnet is isomorphic to the neighbourhood of p‘ w.r.t. the replacing stored net.

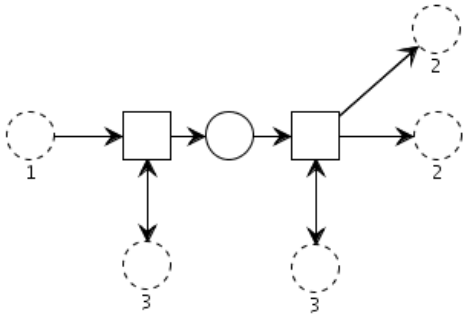

Figure 3: Interface places labelled with the same number have isomorphic neighbourhood. Only interface place labelled with number 1 is unique.

2) In the second step, if there exists exactly one unique interface place of the replaced subnet and exactly one unique interface place of the replacing stored net satisfying that their neigberhoods w.r.t. the respective subnents are not isomorphic, then these interface places are considered to be equal. This correspond to a predicate, that if it is unambiguously possible, then interface places should be identified.
3) Remaining interface places of replacing subnet replacing stored net are changed to ordinary

places and remaining interface places of replaced net becomes interface places of replacing net. It means that it is left to the user to identify manually by further editing which remaining interface places of the replaced subnet equal to remaining interface places of replaicing net.

An example of the use of a subnet concept in PNEditor is illustrated in Figure 4:

1) The transition is created and converted to subnet
2) Visualtization of the inside of the subnet
3) Subnet is modified and saved to file
4) New subnet is created, selected command for replacing subnet
5) Result of 2 identical subnets

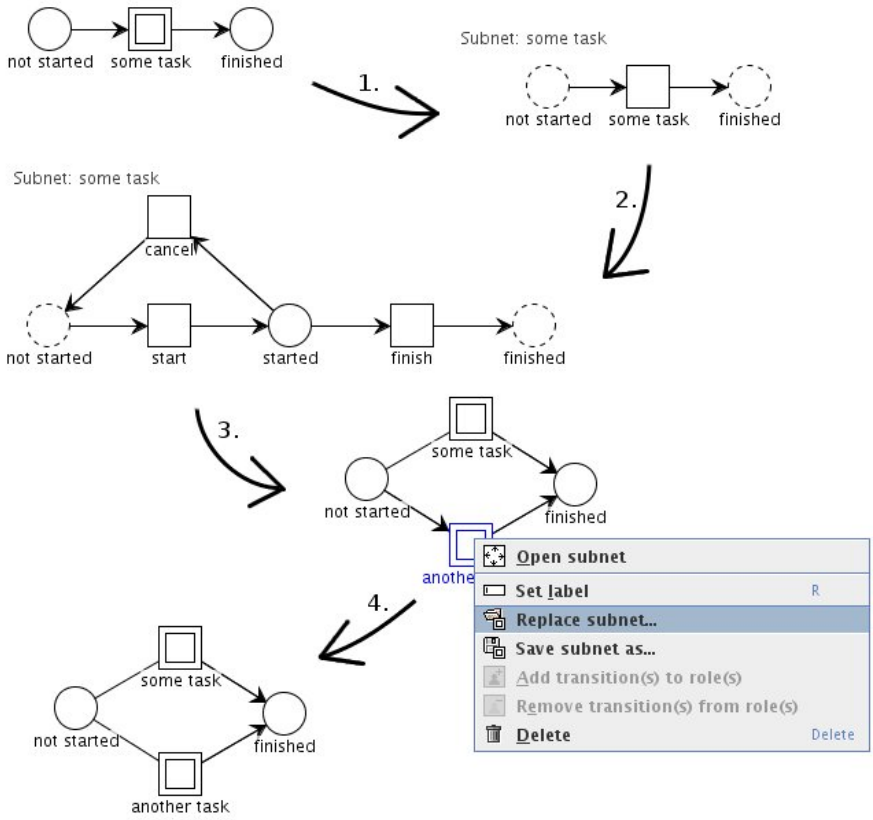

Figure 4: Illustration of the subnet concept in PetriFlow PNEditor

Thus, behind the visualization of a hierarchical process model in PNEditor using the subnet concept is a single flat place/transition net.

References

[BDJL06] R. Bergenthum, J. Desel, G. Juhás, R. Lorenz, : Can I Execute my Scenario in Your Net? VipTool tells you! In *Application and Theory of Petri Nets and Other Models of Concurrency.* LNCS 4024, pp. 381-390, Springer-Verlag, 2006.

[DJLN03] J. Desel, G. Juhás, R. Lorenz and C. Neumair: Modelling and Validation with VipTool. In: *Business Process Management 2003*, LNCS 2678, pp. 380-389, Springer-Verlag, 2003.

[DMVWA05] B. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444-454. Springer-Verlag, Berlin, 2005.

[GA05] C.W. Gunther, W.M.P. van der Aalst: *Modeling the Case Handling Principles with Colored Petri Nets.* Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, October 2005, Department of Computer Science, University of Aarhus, PB-576, 211-230.

[Rei86] W. Reisig: Petri nets: An introduction. Springer, 1985.