Petriflow: Rapid language for modelling Petri nets with roles and data fields

Milan Mladoniczky[1,2], Gabriel Juhás[1,2,3], Juraj Mažári[1,2], Tomáš Gažo[2], and Martin Makáň[2]

[1] Faculty of Electrical Engineering and Information Technology
Slovak University of Technology in Bratislava,
Ilkovičova 3, 812 19 Bratislava, Slovakia,
[2] NETGRIF, s.r.o., Blumentálska 12, 811 07 Bratislava, Slovakia
netgrif@netgrif.com, Home page: http://www.netgrif.com
[3] BIREGAL s. r. o., Klincova 37/B, 821 08 Bratislava, Slovakia
biregal@biregal.sk, Home page: http://www.biregal.com

Abstract. Petri nets are the right tool for modelling of control flow of workflow processes. For more accurate reflection of reality it is necessary to extend Petri nets by more components. For this purpose, modelling language Petriflow was created, that adds roles and data variables to Petri nets and maps roles and data variables to transitions. Development of the language is deeply influenced by real requests incoming from customers or developers modelling complex processes. Every feature of the language is based on real life needs of modelling more robust and complex processes. Based on experience each property of the language was abstracted from real life models of processes. In Petriflow it is possible to model control flow of processes via place/transition Petri nets enriched by reset arcs, inhibitor arcs and read arcs. Petriflow language also introduces layer of roles and data variables into nets. The roles define who can assign an enabled transition and who can execute a transition of the net. The relation between data variables can be defined with another property of the language named Actions. In contrast with other modelling languages, Petriflow allows to set visual aspect of a modelled process, such as behaviour and style of presentation for components.

Keywords: Petri net, Petriflow, modelling, roles, data, process

1 Introduction

Petri nets are a perfect tool for modelling processes. Everybody can understand the modelled process with minor knowledge of Petri nets rules. But the situation is different in the commercial sphere. A majority of people, who model processes as their everyday job, consider Petri nets too simple or incapable to grasp the complex nature of real business processes. Petriflow language was created for purpose of taking advantage of the simple nature of Petri nets and extend them to meet any requirements of the real life business modelling. Every feature is

added to the language based on an experience and on requirements from customers. Petriflow language is developed with the rapid development in mind. If a customer requirement to model a feature of a process is out of scope of the current version of the language, the property is analysed in order to understand the nature of the requirement. If the requirement can be generalised, the resulting requirement is abstracted and implemented as a new property of the language. Petriflow language is written in XML format for the most part. Data field actions are the only exception. They are written in a domain specific language based on Groovy programming language. There are different Petri net extensions and Petri net based tools for modelling workflow processes, such as CPN [1] based on Coloured Petri nets [6], Viptool [2], [3], Yasper [4] or ProM [5], to mention just some of them. The question arises why to create another extension of Petri nets. Most of the Petri net extensions are determined to create models and some of them to analyse the models. Models are only the first step in a life-cycle of a business process management (BPM). The main advantage of BPM is that the model can be used to control the workflow process according to the designed model using a workflow engine. The problem of the most existing Petri net extensions is that they were implemented with different aims, mostly to extend the expressiveness of the formalism, or to analyse models, but they do not provide all the information about implementation details needed for the generation of a deployable application, such as resource management, manipulation with data, or behavioural aspects of the graphical user interface. Another problem is with the case generation. Usually, a model obtained via a Petri net 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 realisation of cases is often done using coloured Petri nets [6]. But in such tools, colours are used both for distinguishing cases from each other as well as for modelling the data of the cases. For the above mentioned reasons, we develop a new language for creating deployable models of workflow processes based on Petri nets. From the very beginning, along with the definition of the Petriflow language, we develop the workflow engine called Netgrif Workflow Management System, where the Petriflow models can be deployed and executed.

2 From Petri nets to Petriflows

As mentioned before, Petriflow language extends Petri nets with other components. As the underlying model, we use place/transition nets enriched by reset arcs, inhibitor arcs and read arcs. The read arcs appear quite necessary in order to model unbounded number of concurrent reading of data in a case. To meet modern business modelling requirements other layers were brought to the language on top of Petri nets. Roles are the first layer to extend Petri nets. Roles layer defines who can fire transitions to which they are bound. Data variables were added as the second layer on top of modelled processes. Data variables rep-

resent all properties of an instance of a process during its life-cycle. To have more control over process instance data, data field actions were added to Petriflow. Actions can define relations or dependencies between data fields in the model of a process or generate values based on a process instance state. All extensions and layers create the right tool for modelling complex, yet simple to understand models of any process that comes to mind.

2.1 Transitions as tasks

In classical Petri nets, firing a transition is one atomic event. During this event a transition consumes tokens from connected input places and produce tokens to output places. The number of consumed and produced tokens is according to the firing rule in original Petri nets. In Petriflow language execution of a transition represent an activity executed by an actor (a user or a system). In the current version of Petriflow language transitions of the underlying net can either represent immediate events (event transitions) or transitions can represent tasks (task transitions) consisting of the assign event, an action which correspond to the state, during which the activity is executed, a cancel event, an optional delegate event and a finish event:

1. Assign event - consumes tokens from input places of the transition in the underlying net.
2. Action - is the state where an actor, who executes the activity, does his job defined by the activity.
3. Cancel event - cancels the task, i.e. produces the consumed tokens back to the input places of the transition in the underlying net.
4. Optional Delegate event - delegates the task to an another actor.
5. Finish event - produces tokens in the output places of the transition in the underlying net.

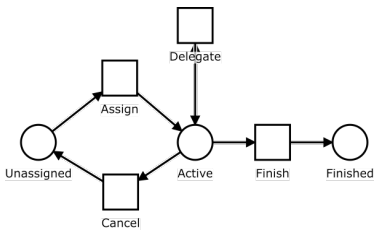

Fig. 1. A subnet modelling a transition representing a task

To illustrate the detailed behaviour of a task transition in the Petriflow language, consider that in the underlying net a task transition has an input place *Unassigned* connected with the task transition by an arc with weight one and an output place *Finished* connected with the task transition by an arc with weight

one. Detailed behaviour of the task transition occurrence representing the task execution can be expressed by a subnet of the underlying Petri net in Figure 1.

3 Role layer in Petriflow

Roles can be defined for Petriflow processes. If a role is associated with an event transition, then it specifies that an actor associated with the role can fire the event transition. In the case of a task transition Petriflow language enables to specify for the role associated with the task transition, whether the actor can fire the assign event, whether the actor can perform the action and fire the cancel event and the finish event or whether the actor (optionally) can delegate the event to be performed by other actor. Assigned or delegated actor has to have a role that authorise to perform the action and to finish the task.

4 Data layer in Petriflow

In Petriflow language the data layer consists of data variables and their binding to transitions of the model of the process. In Petriflow language a data variable is primarily defined by its type, unique id and title. The type attribute of data variable determines the data structure of the variable. For example, if the data type of variable is set to number, the value is stored as a number of the type double. Petriflow language allows to use of primitive data types like number, boolean, as well as standard data types such as date, text as well as more complex types like enumeration, multi-choice, file and table. The language also introduces domain specific types user and case reference. The user type stores the reference to the specific user of the system, who has assigned a process role. The case reference data type is used when it is required to create connection between two different cases. Visual attributes can be also set in the data variable object to more precisely define the representation and behaviour of the data variable in graphical user interface. For example, the attribute placeholder sets a text value of a data field element when no user defined default value is present.

4.1 Binding data fields to task transitions

It is not a rule that every data variable has to be bounded to some task transition. A data variable should be bounded to a transition if it is desired to display its values to the user performing the task transition. A data variable associated to a task transition is called data field of the task transition. Obviously, a data variable can be bounded to several transitions. A data field, i.e. a connection between a data variable and a task transition is implemented as a reference inside of the task transition. The referenced data variable is identified in the reference task transition by its unique id set in the data variable object. In addition to data variable id, reference object has attributes to set data field behaviour and logic inside of the transition. Behaviour attribute defines relation of the data

field to the transition. The attribute values can be hidden, visible, editable, and required. When the data field in a transition is visible user can see the value of the data field but cannot modify it. When one or more data fields in a transition are set as required, this transition cannot be finished until every required data field is filled. Values in the logic attribute are functions. In Petriflow language they are named as Actions. They are executed when the value is changed inside of the referenced data field.

5 Petriflow Actions

Actions are functions executed every time when a value inside of a data field is changed. Actions can be placed into a data variable definition or into a data field logic attribute (i.e. into a data variable reference inside of a task transition). Each action contains two parts. Action variables are defined first. They reference to a data variable or a transition in the process model. Second part consist of keywords that define a desired expression. Values of data variables referenced by action variables are changed according to the evaluated expression. Actions are not written in XML format. Actions implementation is based on Groovy DSL meta language. Groovy allows to define own semantics for domain specific language and then compiles it to an executable code. All actions are compiled after successful import of a model to Netgrif WMS and then saved into a database. Actions are a big advantage of Petriflow language, because they define relations between data fields across whole process model. They modify information stored inside of each case of the model in real time as transitions are fired.

Algorithm 1 Usage of action defined in DSL

```
<data type="enumeration">
    <title>PERIODICITY</title>
    <id>108001</id>
    <values>yearly</values>
    <values>quarterly</values>
    <init>quarterly</init>
    <action trigger="set">
        field: f.this, amount: f.308004, payment: f.308006;
        change payment about {
            if (field.value == "yearly")
                return 0.95*amount as Double;
            if (field.value == "quarterly")
                return amount/4 as Double;
        }
    </action>
</data>
```

For better illustration, consider the data variable object with the action defined in Algorithm 1. The data variable object is an enumeration with two choices, namely *yearly* and *quarterly*, with the initial value *quarterly*. The action works with three action variables, namely *field* which refers to the data variable itself, action variable *amount*, which refers to the data variable with id 308004 and action variable *payment*, which refers to the data variable with id 308006. Consider, that in the data variable referred by action variable *amount* is stored the previously computed amount of an insurance. The action works as follows, if the value *yearly* is set, then 5% discount is given and the value of the data variable referred by action variable *payment* is set to the 95 % of the insurance. If the value *quarterly* is set, then no discount is given and the value of the data variable referred by action variable *payment* is set to the one quarter of the insurance.

6 Conclusion

We have briefly introduced Petriflow language, which is an extension of Petri nets using roles and data variables associated with transitions and functions over data variables called actions. The Petriflow language is suitable for creating deployable models of workflow processes. The further step is to enrich the Petriflow language by the possibility to define process scope (global) variables and by the communication with data passing between cases of different processes either via task transitions or via a constructor.

References

1. M. Beaudouin-Lafon, W. E. Mackay, M. Jensen, P. Andersen, P. Janecek, M. Lassen, K. Lund, K. Mortensen, S. Munck, A. Ratzer, K. Ravn, S. Christensen, K. Jensen: CPN/Tools: A Tool for Editing and Simulating Coloured Petri Nets In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS 2031, pp. 574577, Springer-Verlag, 2001.
2. 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. 381390, Springer-Verlag, 2006.
3. J. Desel, G. Juhás, R. Lorenz and C. Neumair: Modelling and Validation with VipTool. In: BPM 2003, LNCS 2678, pp. 380389, SpringerVerlag, 2003.
4. K.M. van Hee, J. Keiren, R. Post, N. Sidorova, J.M. van der Werf: Designing case handling systems. In Transactions on Petri Nets and Other Models of Concurrency I, LNCS 5100, pp. 119133, Springer, Berlin. 2008.
5. 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 Application and Theory of Petri Nets 2005, LNCS 3536, pp. 444454. Springer-Verlag, Berlin, 2005.
6. 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, 2005, Department of Computer Science, University of Aarhus, PB-576, 211230.