UDC 65.01

Petri Nets on the Semantic Web Guidelines and Infrastructure

Dragan Gašević

GOOD OLD AI research group, FON – School of Business Administration
University of Belgrade, Serbia and Montenegro
http://goodoldai.org.yu
Department of Computer Engineering and Informatics, Military Academy
Belgrade, Serbia and Montenegro
dgasevic@acm.org

Abstract. This paper gives the Petri net ontology as the most important element in providing Petri net support for the Semantic Web. Available Petri net formal descriptions are: metamodels, UML profiles, ontologies and syntax. Metamodels are useful, but their main purpose is for Petri net tools. Although the current Petri-net community effort Petri Net Markup Language (PNML) is XML-based, it lacks a precise definition of semantics. Existing Petri net ontologies are partial solutions specialized for a specific problem. In order to show current Petri net model sharing features we use P3 tool that uses PNML/XSLT-based approach for model sharing. This paper suggests developing the Petri net ontology to represent semantics appropriately. This Petri net ontology is described using UML, Resource Description Framework (Schema) – RDF(S) and the Web Ontology Language – OWL.

1 Introduction

The main idea of this paper is to provide suitable way for Petri nets [1] to be used on the Semantic Web. That means, full semantic interoperability of Petri net models. Currently, Petri net interoperability is assumed as a syntax for model sharing. This is firstly introduced in the paper [2] where the authors said that it would be very useful if Petri net researchers could share their Petri net model descriptions. In this way more software solutions could be used for observing the same model.

Accordingly, different software tools [3] implement different syntax for model sharing. Previously, most of them implemented regular text-based formats (e.g. DaNAMiCS). Recent Petri net tools implement XML-based formats. In this way, they achieve interoperability on the Web, since the XML is W3C's recommendation. Especially important advantage of this approach is that XML documents can be easily transformed using

eXtensible Stylesheet Language Transformations (XSLT) into other formats (that should not necessarily be XML-defined). For instance, Renew Petri net tool was one of the first tools that used XML support. Nowadays, within Petri net community there are efforts to formulate universal Petri net transfer format using XML. This initiative is called *Petri Net Markup Language (PNML)* and tends to be a part of future ISO/IEC High-level Petri net standard [4].

However, all these attempts are syntactically oriented, i.e. they introduce some constraints that enable validation of documents against their definition (e.g. when we want to validate whether an arc connects two nodes of different types, i.e. a place and a transition). In this paper, we are aware of the fact that Petri net syntax is very important for Petri net model distribution, but we need something else to define and validate the semantics. Current solutions that give semantic Petri net descriptions are: metamodels, UML profiles and ontologies. But, metamodel-based solutions are mainly intended for developing Petri net tools and do not have support for the Semantic Web languages. On the other hand, Petri net ontologies only attempt to solve concrete issues using a specific Petri net dialect (i.e. Time Petri nets). Therefore, we propose a Petri net ontology [5] as a semantic description of the Petri net concepts and their relationships. In this way, we put the Petri net usage in the context of the Semantic Web and enable Petri nets to be described using Semantic Web languages [6]. Also, we can incorporate Petri net description into other non-Petri net XML-based formats (e.g. *Scalable Vector Graphics - SVG*) and in that way we would be able to reconstruct Petri net model using metadata and annotations according to the Petri net ontology.

The next section describes existing Petri net formal descriptions: metamodels, UML profiles, ontologies and syntax. We give main focus on Petri net syntax because most work has been done on this problem. Especially, we discuss current Petri net syntax standard proposal – Petri Net Markup Language. Section three depicts P3 – a Petri net tool that uses PNML and contains collection of XSLTs for transforming the PNML document into formats of DaNAMiCS, Renew, and Petri Net Kernel (uses PNML as well). Then, section four enumerates advantages of the Petri net ontology. Section five outlines a Petri net ontology development – its initial realization in UML, Protégé (RDFS-based), and an UML profile (OWL-based). In section six we show how P3 tool supports the proposed Petri net ontology using RDF and mappings between RDF and PNML based on XSLT. This work is a part of GOOD OLD AI (http://goodoldai.org.yu) effort for developing AIR - a platform for intelligent systems.

2 Previous work on Petri net sharing

This section discusses previous work in developing Petri net formal description that can be used in different software solutions for Petri net: model sharing, software implementation, model validation, etc. Therefore, we analyze present Petri net: metamodels, UML profiles, ontologies and syntax.

2.1 Petri net metamodels

Breton and Bézivin in their paper [7] define a Petri net metamodel in the context of the OMG's Model Driven Architecture (MDA) (http://www.omg.org/mda) initiative since they use the Meta-Object Facility (MOF) for metamodel definition. In this paper we assume that metamodel concept is closely related to ontology concept. They start from the definition that a metamodel defines a set of concepts and relations, i.e. the terminology and a set of additional constraints, i.e. the assertions. Also, they say that each model encompasses both a static part and a dynamic part. Accordingly, they define Petri net metamodel consisting of three parts:

1. *Petri Nets Definition metamodel* that defines the static part of Petri nets (i.e. Petri net basic structure concepts: Petri net itself, Place, Transition, Arc, and their mutual relations). Additionally, the Object Constraint Language (OCL) is used here to define an arc's source and target nodes.
2. *Petri Nets Situation metamodel* that defines a particular situation of Petri nets. In order to represent particular a Petri net situation they introduce Marking and Token concepts in the Petri net metamodel
3. *Petri nets Execution metamodel* that defines a sequence of particular situations. This metamodel contains Petri net concepts (i.e. Move) needed for Petri net execution since Petri net execution consist of a sequence of transition firings with regard to place marking.

Note that this proposal is very important for development Petri net tools. Although this approach gives a useful classification of Petri net concepts in three different parts it has few shortcomings because it does not: consider existence of different Petri net dialects and Petri net structuring mechanisms (e.g. pages); show how Petri nets can be used on the Semantic Web with non-Petri net tool (i.e. annotation), and hence how Petri nets are mapped into Semantic Web languages (e.g. RDF(S)); suggest which general MOF-based tools can be used for validating models against their metamodels.

Hansen proposes a Petri net UML Profile in the paper [8]. Defining a Petri net UML Profile provides similar solution to the metamodel-based

one as UML Profiles extend the UML metamodel by introducing stereotypes, tagged values and constraints. The main intent of an UML Profile is to provide the usage of standard UML tools for different purposes. Thus, Hansen extends the UML metamodel with Coloured Petri net concepts – stereotypes for Petri net nodes, places, transition, arcs, and declarations. Additionally, this UML Profile has tagged values attached to stereotypes for: places (i.e. initial marking, and colour set), transitions (i.e. guard), and arcs (expression). Also, the OCL is used in order to define more precisely semantics for the UML Profile. Finally, this solution has software implementation as a practical support that extends an existing UML tool (the Knight/Ideogramic UML tool) with artifacts from the Petri net UML Profile. Although, this solution is metamodel-based it is fairly awkward since it is based on the UML metamodel. That means all UML concepts are introduced in the Petri net metamodel, but most of them are needless for Petri net semantic. Also, this approach has the same limitation as the previous one regarding support for: Petri net dialects, Semantic Web use, and Petri net structuring mechanisms.

2.2 Petri net ontologies

Perleg and her colleagues propose modeling of biological processes using workflows [9] since it has ability to represent process knowledge. On the other hand, workflows can be mapped to Petri nets, which allows verification of formal properties and qualitative simulation (i.e. reachability analysis). We develop a Petri net ontology using Protégé tool as a software support. They acquire a knowledge of a Petri net through Graphical User Interface (GUI) that extends the standard GUI of the Protégé's tool [10]. Actually, this GUI provides graphical tools for all Petri net concepts (Places, Transitions, and Arc). In addition, the Petri net ontology is represented in RDFS, and concrete Petri net models are represented in RDF. This solution gives a solid starting point for defining the Petri net ontology. However, this solution has some limitation: it considers only Time Petri nets regardless of other Petri nets, it does not define Petri net structuring mechanisms, and it does not provide precise constraints (e.g. types of an arc's source and target nodes that can be done using Protégé Axiom Language (PAL) constraints). Additionally, we want to provide other ontology languages for Petri net ontology (e.g. DAML or OWL).

2.3 Petri net syntax – tool specific formats

Abstract Petri Net Notation (APNN) is presented in the paper [11]. It has ability to describe different Petri net dialects. To increase the readability

of this notation, the key words are similar to LATEX commands. This notation should satisfy the following requests:

- net descriptions should be easily exchanged in electronic form;
- extensibility – it should be used by different Petri net dialects. Simple Petri net dialects could be extended in order to describe high-level dialects;
- modularity and hierarchy – the Petri net description in the file should be reusable
- readability – text notation should be easily transformable into a human-readable format as well as suitable for printing.

This notation does not keep information on Petri net graphical elements (place position, transition, place name, etc.). The abstract notation for each Petri net class is defined in BNF. The convention used for writing grammar productions includes the following: terminals are written in lower-case and non-terminal in upper case. This paper contains a short review of APNN, using an example of P/T nets that is shown in Figure 1.

Tool specific text-based format. The DaNAMiCS software solution can be downloaded from http://www.cs.uct.ac.za/Research/DNA/DaNAMiCS where it is available for free. It is implemented in Java and supports High-level Petri nets and Stochastic Petri nets. The main advantage of this software is a rich set of tools for analysis, such as place and transition matrix invariants, structural analysis as well as a few both simple and advanced performance analyses. For model recording DaNAMiCS uses a file format with the *bim* extension. Format with the *bim* extension has many internal marks whose documentation is not publicly. The second file format, that DaNAMiCS uses, has the *wam* extension and these files are used for model import (menu *File*, option *Import net*). However, it is a textual format with a structure that evidently corresponds to certain Petri net object. It has been analyzed and compared with the models obtained by importing files of this format into the DaNAMiCS. The meaning of every format element has been obtained as well. One example of a Petri net description in the format with the *wam* extension is given in Figure 2a whereas the graph of that net is given in Figure 2b.

```
\beginnet, \endnet, {, }, \place, \transition, \like, \arc, \name, \init,
\from, \to, \capacity, \weight
```

a)

```
NET, ELEMENT, PLACE, TRANSITION, ARC, ID, NAME, INIT, WEIGHT, CAP,
STRING, INTEGER
```

b)

```
    NET           ::= \beginnet{ID} ELEMENT \endnet
    ELEMENT       ::= empty
                    | PLACE ELEMENT
                    | TRANSITION ELEMENT
                    | ARC ELEMENT
    ID            ::= STRING
    PLACE         ::= \place{ID}{ NAME INIT CAP }|\place{ID}{ \like{ID} }
    NAME          ::= empty | \name{ STRING }
    INIT          ::= empty | \init{ INTEGER }
    CAP           ::= empty | \capacity{ INTEGER }
    TRANSITION    ::= \transition{ID}{ NAME }
    ARC           ::= \arc{ID}{ \from{ID} \to{ID} WEIGHT }
    WEIGHT        ::= empty | \weight{ INTEGER }
    Start-symbol: NET
```

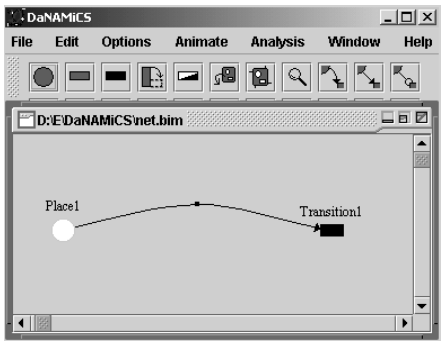
c)

Fig. 1. Abstract Petri Net Notation: a) set of terminal symbols, b) set of non-terminal symbols, c) set of grammar productions

Tool specific XML-based format. Renew Petri net software solution can be downloaded from http://www.renew.de, it is available for free, and is Java implemented. In order to overcome the problem of model exchange with other Petri net software tools, Renew uses XML. It supports the following Petri net dialects: object oriented Petri nets, High-level Petri nets, P/T nets, and Time Petri nets. Advantages of Renew software solution are: support of synchronized channels as an advanced communication mechanism; support of the modeling object oriented concepts; support of numerous arcs types; a rich graphical environment. The XML document model description is defined using *Document Type Definition (DTD)* [12] [13]. The assumptions, included in the formulation of this DTD, are the same as the PNML assumptions since they use the same elements for the description of: net (XML tag net), place (place), transition (transition) and arc (arc). These elements in previously mentioned two formats also have similar content model. Each element in the Renew's XML format can have graphical information and arbitrary number of annotations.

```
PiPs [
]
TiPs [
]
Places [
1 'Place1' (82,104) 0 0 0 0
]
TimedTrans [
]
ImmedTrans [
3 'Transition1' (201,102) 0 0 0
]
Subnets [
]
Edges [
'Place1' to'Transition1' 1 [( 139,78)
]
]
```

a)



b)

Fig. 2. Format for Petri net description, which can be imported into DaNAMiCS: a) example of a net described in this format, b) graphical presentation

2.4 Petri Net Markup Language

The Petri net community has already been working on development of the Petri Net Markup Language for three years [14] [15] that might be a part of the future High-level Petri nets ISO/IEC standard [4]. PNML is a proposal that is based on XML. The design of PNML was governed by the principles of [16]:

- *flexibility* - PNML should be able to represent any kind of Petri nets with their specific extensions and features
- *unambiguity* - Ambiguity is removed from the format by ensuring that the original Petri net and its particular type can be uniquely determined from its PNML representation. Accordingly, PNML supports the definition of different Petri net types through the use of the Petri net type definition (PNTD), which determines legal labels for a particular Petri net type
- *compatibility* - unlimited exchange of information between different types of Petri nets should be provided. PNML comes with conventions on how to define a label with a particular meaning. The Conventions Document predefines for all kinds of extensions both their syntax and intended meaning. When defining a new Petri net type, the labels can be chosen from this Conventions Document.

The PNML specification is based on the PNML technology metamodel that formulates a PNML document structure. Actually, this metamodel defines basic Petri net concepts (places, transitions, arcs) as well as their relations that can be presented in a PNML document. Currently, PNML is in version 1.3, and it is defined using RELAX NG – an XML grammar definition mechanism. One should notice that PNML can also be described

using W3C's XML Schema definition, and previous PNML versions were define by this mechanism as well. The full PNML definition as well as a few examples of PNTD can be found at the PNML home page: http://www.informatik.hu-berlin.de/top/pnml/about.html. The next section describes how the present Petri net tools support PNML, as well as how one using XSLT can share models with tools that do not support PNML.

3 PNML – current tool support

PNML, being more matured, is currently supported (or will be supported) by many Petri net software tools, for instance: Petri Net Kernel (PNK), CPN Tools, Worflan, PIPE, PEP, VIPtool, P3 etc. There are also Petri net tools that do not primarily use PNML syntax, but do use considerably similar formats to the PNML (e.g. Renew). In this paper we emphasize PNK – a tool that is closely related to the PNML technology. The PNK is not just a Petri net tool, but also an infrastructure for building Petri net tools [17]. It is not limited to one Petri net dialect; on the contrary it can be used for each Petri net dialect, supporting specific features of each one. Thus, it provides methods to manage Petri nets of different types. PNK implements a data model for Petri nets that is similar to that of PNML. Each place, transition, arc, or even net may contain several labels according to the Petri net type.
Secondly, we make overview of P3 tool – a tool that has been developed for Petri net teaching [18]. P3 contains collection of XSLTs from the PNML to other Petri net formats. On the example of P3 tool we depict a possible PNML/XSLT-based syntax schema for model sharing.

3.1 P3 – Petri net tool

Being based on the PNML concepts, P3 achieves compatibility with the PNML. The P3 tool supports P/T nets, and Upgraded Petri nets. Main parts of the P3's architecture are following [19]:

- *Petri net structure* – The central part of the structure is a Petri net that consists of the Petri net basic concepts: places, transitions, and arcs [20]. Important part of the Petri nets that pertains their structure is *marking*, and *initial marking*, although these concepts are not real part of the Petri net structure
- *Petri net graph* – is closely related to the Petri net structure. It can be said that Petri net graph is a graphical notation for Petri net structure
- *Petri net simulation* – implements two different modes of simulation: by parallel execution of all enabled transitions with a previous conflict resolution; and by single execution of an enabled transition

- *Petri net analysis tools* - there are many well-known Petri net analysis tools [1] (e.g. *Reachability Tree, Matrix Equations*), but also we introduced new analysis tools appropriate for teaching purposes (e.g. *Fireability Tree, Firing graph*)
- *Petri net model sharing* – P3's sharing is format based on the PNML as PNML is extensible.

The P3's architecture is shown in Figure 3. The organization of Petri net classes is shown on the left in the figure, whereas the supported formats are on the right side.

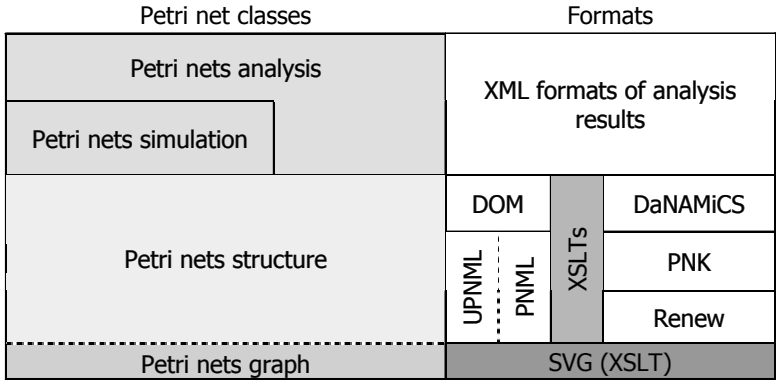


Fig. 3. P3 architecture: class organization and supported XML formats

P3 uses a collection of XSLTs to convert PNML format into the formats of Renew, DaNAMiCS, and PNK tools. These XSLTs are omitted here due to their length, but are available at: http://www15.brinkster.com/p3net. The transformation principle that was used is illustrated in Figure 4: a P3-generated PNML-based model is the input to the XSLT processor, and is converted using the XSLT corresponding to the target tool. Also, using the same principle P3 supports transformation to the SVG – a W3C's format for 2D vector graphics.

DaNAMiCS is selected in order to analyze sharing a PNML-based model developed in P3 with a tool that uses an ordinary text format. Renew was suitable for analyzing the exchange of P3 models (PNML-based) with a tool that uses another XML-based format, and PNK enabled analysis of model exchange between two different PNML-based tools.

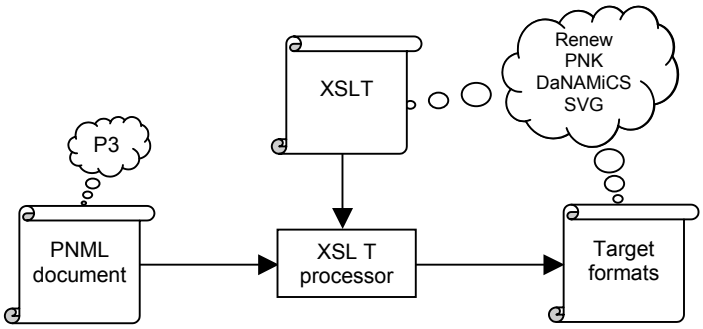


Fig. 4. The principle of XSLT-based model conversion from P3 to another format

Model sharing between P3 and DaNAMiCS is possible, but impeded by the lack of publicly available specification of DaNAMiCS format. It is not possible to use XML/XSLT-based approach to convert the models developed in DaNAMiCS, since such models are described in text-based files. The major difficulty in sharing models between P3 and Renew is Renew's syntax for identifiers, which is not defined in the XML specification; it is only hard-coded in the Renew source code. Another problem related to Renew's XML format is DTD mechanism used to define this format. This format has very little support for description of semantics (e.g. inheritance). As it was expected, for the P3-PNK pair, model sharing is the most convenient approach, since PNK uses PNML. Still, some difficulties exist because P3 and PNK interpret PNML slightly different, and hence usage of an XSLT was inevitable.

4 Do we need a Petri net ontology?

As we have seen so far, Petri net formats use different concepts for defining its syntax. Some of these syntactic-based approaches actually have problems with syntax validation. For instance, it is very difficult to validate some text-based (i.e. DaNAMiCS) document if we do not develop specialized software for checking this format. A slightly better solution is to use DTD for XML definition as the Renew's format uses. But, DTD has well-known drawbacks: it does not support inheritance (generalization/specialization), it does not have datatype checking (for the primary semantics checking), it does not support defining specific formats, and what is more a DTD document has non-XML structure.

W3C's XML Schema overcomes most of these problems, since it has: a rich set of datatypes, constructs to define inheritance of complex as well as simple types, and document structure that is in the form of a well-formed XML document. But, XML Schema has not full support for describing semantics [21]. In fact, XML Schema is only a way for defining syntax. For

example, it is emphasized that current PNML definition does not have an ability to validate whether an arc connects a place and a transition, or two transitions or two places. Also, directly using some standard XML validators cannot validate whether a reference place has a reference to a place or other reference place [16]. In order to perform this kind of validation one must use some specific tools (e.g. for PNML is proposed using *Jing* validator), but these tools are not widely known in the XML community. Further, in the case we want to share Petri net models, not only with Petri net specialized tools, we must have a formal way for representing Petri net semantics since we can not expect that non-Petri net tool perform semantic validation.

Accordingly, we believe that the concept of ontology can be used for formal description of Petri net semantics. In this paper domain ontology is understood as formal way for representing shared conceptualization in some domain [5]. Ontology has formal mechanisms to represent concepts, concept properties, and relations between concepts in the domain of discourse. Having a Petri net ontology we would be able to overcome validation problems that we have so far noticed. However, a Petri net ontology does not exclude current Petri net formats (especially PNML). Ontology is closely related to syntax in the meaning that syntax should enable ontological knowledge sharing [22]. When we have a Petri net ontology, we could use ontological tools for validation of Petri nets models (e.g. Protégé – a tool for ontological development). Also, having a Petri net ontology, one can use Semantic Web languages for representing Petri net models (RDF, RDF Schema – RDF, DAML+OIL, OWL, etc) [6]. Accordingly, we show how PNML can be used as a guideline for the Petri net ontology.

5 Petri net ontology – initial realization

There are many different ways to develop an ontology and for ontological engineering one can use different tools. For development of the Petri net ontology, we firstly decided to use UML [23]. The use of the UML is suitable because it is generally accepted and standardized approach for analysis and modeling in software engineering. We can also employ existing UML-based Petri net descriptions that are made within PNML definition [16]. However, neither UML tools nor UML itself are intended to be used for ontology development. Thus, in order to achieve more precise Petri net definition than an UML model provides, it is necessary to use some ontology development tools. We decided to use Protégé 2000 [10] since it is the leading tool for ontology development. Protégé 2000 is also suitable because it is able to import UML models. This is enabled by Protégé's UML backend – that imports UML models (in UML's *XML*

Metadata Interchange – *XMI* – format) into Protégé ontology. Next, this section describes design details of the Petri net ontology. In order to have a precise UML-based ontology definition, we use Ontology (OWL-based) UML profile that we transformed into OWL using a converter developed in XSLT [30].

5.1 Starting solution – UML model

Hierarchy of core concepts of the Petri net ontology is shown in Figure 5. In design of the Petri net ontology, we have one root element, and we call it – *ModelElement*. This element is parent for all elements of Petri net structure. The name of this class is *ModelElement* because the UML's metamodel uses the same name for its root class [24]. A Petri net (*Net* class) can contain many different *ModelElement*s. *ModelElement* and *Net* have ID attribute (unique identifier) of String type, and *Net* has also an attribute that describes a type of Petri net. It is in accordance with PNML. Basically, there are three main Petri net concepts are: place, transition, and arc. These concepts constitute a Petri net structure, and they are shown in Figure 5 with classes that have the same names (i.e. *Place*, *Transition*, *and Arc*). Places and transitions are kind of nodes (class *Node*), and an arc connects two nodes of different kind. This constraint can be represented using Object Constraint Language (OCL), with the following statements:

```
context Arc
  inv: self.to.oclIsTypeOf(Transition) and self.from.oclIsTypeOf(Place) or
       self.to.oclIsTypeOf(Place) and self.from.oclIsTypeOf(Transition)
```

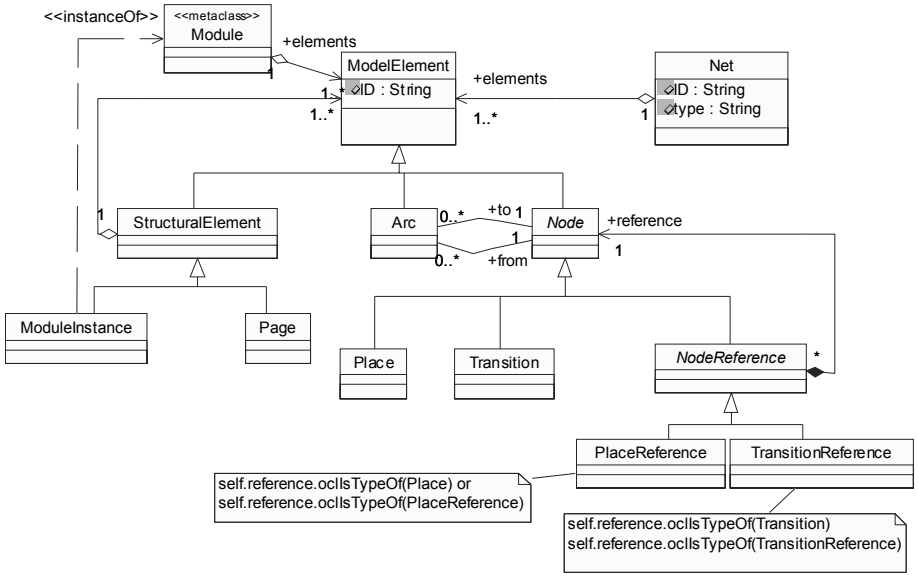


Fig. 5. Petri net ontology – Hierarchy of core Petri net concepts

Class *Node* is introduced into the ontology in order to have a common way to reference both places and transitions. In order to make easily maintainable Petri net models, different concepts for structuring can be used. In the Petri net ontology, we have the class *StructuralElement*. This class is inherited from *ModelElement*, and we inherit from this class all classes that represent structuring mechanisms. We have decided to support two common mechanisms: pages (class *Page*), and modules (class *Module*). A *Page* may consist of other Petri net *ModelElement*s – it may even consist of other pages. A *NodeReference*, which can be either a *TransitionReference* or a *PlaceReference,* represents an appearance of a node. *Decorator* design pattern [25] was used to represent referencing of a *NodeReference*. Here, there are also constraints: a *TransitionReference* can refer to either a *Transition* or other *TransitionReference*, while a *PlaceReference* can refer to either a *Place* or other *PlaceReference*. We show all these constraints using OCL in Figure 5. These constraints also affect the OCL constraint for arcs, we have already described, but we do not show their interaction due to the limited size of this paper. The second structuring mechanisms are modules. A *Module* consists of *ModelElement*s, and it can be instantiated (similarly as an object is instantiated from a class in object oriented paradigm). Accordingly, *Module* is a metaclass (stereotype in Figure 5), and *ModuleInstance* depends on *Module* (that shows a stereotyped *instanceOf* dependency from *ModuleInstance* to *Module*).

In Petri nets an additional property (or feature) can be attached to almost every core Petri net element (e.g. name, multiplicity, etc.). Thus,

we have included a description of features in the Petri net ontology and in Figure 6 we shortly depict how these features have been added. The root class for all features is *Feature*, which is also similar to the UML metamodel [24]. The Petri net ontology follows the PNML's classification of features: those that contain graphical information (annotation), and those that do not have them (attribute). In the Petri net ontology every feature directly inherited from *Feature* class is an attribute (e.g. *ArcType*), whereas *GraphicalFeature* class represents annotations. *GraphicalFeature* has a graphical information that can consist of, for instance, position (class *Position* and its children *Absolute Position*, and *Relative Position*). Examples of graphical features are: *Multiplicity*, *Name*, *InitialMarking*, and *Marking*. It is interesting to notice that marking, and initial marking consist of tokens (class *Token*). In order to support token colors, the *Token* class is abstract. In Figure 6 we show a case when we have no colors attached to tokens, instead we just take into account number of tokens (*IntegerToken*).

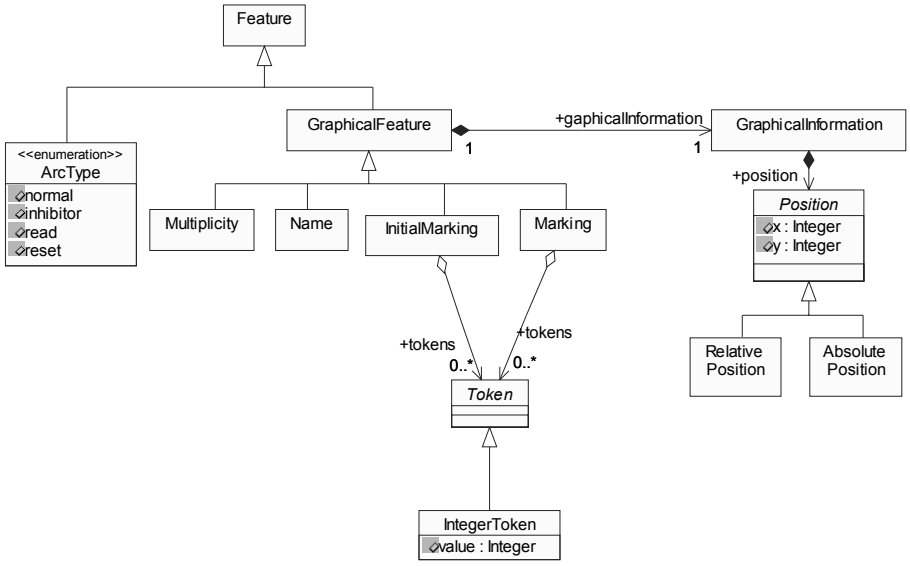


Fig. 6. Property hierarchy of the Petri net ontology

Attaching a new feature to a Petri net class requires just adding an association between a class and a feature. Figure 7 shows how *Name* and *Position* features are attached to the *Node* class. Using the same procedure one can attach features to other Petri net classes.

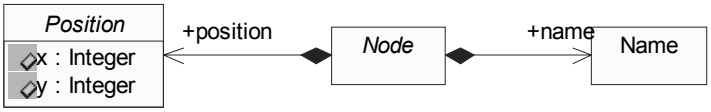


Fig. 7. An example of how features cab be attached to a class in the proposed Petri net ontology: a Petri net node has position and name

A UML description is a convenient way for representing Petri net semantics. Also, this Petri net model can be used as a Petri net metamodel in order to provide the future Petri net implementation that should take benefits from MDA and repository-based software development [26]. However, it does not enable us to semantically validate Petri net models. For example, we cannot use OCL statements to perform this task. Additionally, the standard UML has some semantics differences in regard of ontology properties. Unlike UML's attributes, ontology properties are first-class concepts that can exist independent of any ontology class [27]. In order to further refine the Petri net ontology, we have two directions. The fist one recommends using an UML profile [28] for ontology development. The second way is to use standard ontology development tools. Therefore we have decided to use: 1. Protégé 2000 since it provides all necessary ontology development features (constraints, and ontology languages), but it also has ability to import/export UML models we have previously shown; 2. The Ontology UML Profile [29] that is based on OWL – a W3C recommendation for Web ontology language [31].

5.2 Petri net ontology in Protégé 2000

We can precisely define Petri net ontology by Protégé tool, for example: we can make difference between a class and a metaclass (e.g. *Module* – a metaclass, *ModuleInstance* – a class), we can use different Semantic Web languages to represent Petri net ontology provided through Protégé's backends (RDF(S), OWL, DAML+OIL), we can define constraints that we specified in the UML model using OCL (e.g. PAL – Protégé Axiom Language). Afterwards, we can validate all ontology instances using these constraints, and detect if there an instance that does not conform to some of constraints.

Having created the initial design of the Petri net ontology, it was imported into the Protégé using Protégé's UML backend (http://protege.standford.edu/plugin/uml). This plugin has ability to read an XML format (i.e. UML XMI) for representing UML models. The main shortcoming of this UML backend is that it is unable to map UML class associations. Thus, we had to manually add all slots that are described in UML as association ends. A snapshot of the Petri net ontology after we imported it and inserted all slots (i.e. association ends) in Protégé is shown in Figure 8.

Of course, Protégé does not have an ability to transform OCL constraints into PAL constraints. Thus, we have also manually reconstructed all the OCL-defined constraints from the UML model of the

Petri net ontology into corresponding PAL constraints. For instance, a constraint that is attached to *TransitionReference* that can refer only to a *Transition* or other *TransitionReference* looks like this:

```
(forall ?transitionRef
   (or (instance-of (reference ?transitionRef) Transition)
       (instance-of (reference ?transitionRef) TransitionReference)
))
```

This constraint can be applied to instances of the Petri net ontology, and Protégé shows all those instances of *TransitionReference* that do not conform it. Applying the same principle we made constraints for *PlaceReference*, and *Arc*.

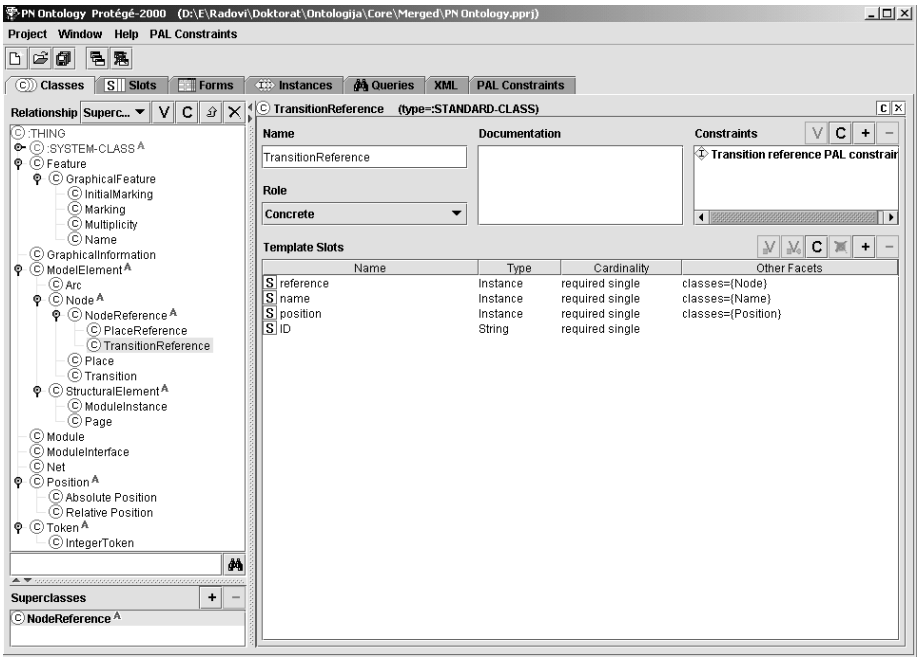


Fig. 8. Look on the Petri net ontology in Protégé 2000

Using Protégé we generated RDFS that describes the Petri net ontology. In this way, one can use it for reasoning about a document that contains a Petri net model. Figure 9 shows an excerpt of this RDFS. This figure depicts how RDFS defines classes for: *ModelElement*, *Node*, *Transition*, *Place*, *Arc*, and *ArcType*. Also, this figure shows how RDFS defines *Feature*, as well as how *name* feature is defined and attached to classes that should have this property.

Since Protégé supports more concepts for ontology definition than RDFS does, one can notice some extensions of RDFS in Figure 9. These Protégé extensions are manifested by namespace a, and for example, they are used

to define cardinality (a:maxCardinality, a:minCardinality), or to refer to a PAL constraint (a:slot_constraints), etc. Of course, it is neither limitation of the Petri net ontology nor Protégé tool, but it is limitation of RDFS itself, and most of them are overcome in the Web Ontology Language (OWL) [31], but this discussion is out of the scope of this paper.

```
<rdf:RDF xmlns:rdf="&rdf;" xmlns:a="&a;"
    xmlns:PN_Ontology="&PN_Ontology;" xmlns:rdfs="&rdfs;">

    <!-- … -->

    <rdfs:Class rdf:about="&PN_Ontology;ModelElement" a:role="abstract"
        rdfs:label="ModelElement">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&PN_Ontology;Node" a:role="abstract" rdfs:label="Node">
        <rdfs:subClassOf rdf:resource="&PN_Ontology;ModelElement"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&PN_Ontology;Place" rdfs:label="Place">
        <rdfs:subClassOf rdf:resource="&PN_Ontology;Node"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&PN_Ontology;Transition" rdfs:label="Transition">
        <rdfs:subClassOf rdf:resource="&PN_Ontology;Node"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&PN_Ontology;Arc" rdfs:label="Arc">
        <rdfs:subClassOf rdf:resource="&PN_Ontology;ModelElement"/>
        <a:_slot_constraints rdf:resource="&PN_Ontology;PN Ontology_00043"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&PN_Ontology;ArcType" rdfs:label="ArcType">
        <rdfs:subClassOf rdf:resource="&PN_Ontology;Feature"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&PN_Ontology;Feature" rdfs:label="Feature">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
    </rdfs:Class>

    <!-- … -->

    <rdf:Property rdf:about="&PN_Ontology;name" a:maxCardinality="1"
        a:minCardinality="1" rdfs:label="name">
        <rdfs:domain rdf:resource="&PN_Ontology;ImmediateTransition"/>
        <rdfs:range rdf:resource="&PN_Ontology;Name"/>
        <rdfs:domain rdf:resource="&PN_Ontology;Node"/>
        <rdfs:domain rdf:resource="&PN_Ontology;Place"/>
        <rdfs:domain rdf:resource="&PN_Ontology;PlaceReference"/>
        <rdfs:domain rdf:resource="&PN_Ontology;TimedTransition"/>
        <rdfs:domain rdf:resource="&PN_Ontology;Transition"/>
        <rdfs:domain rdf:resource="&PN_Ontology;TransitionFunction"/>
        <rdfs:domain rdf:resource="&PN_Ontology;TransitionReference"/>
    </rdf:Property>

    <!-- … -->
```

Fig. 9. A part of the RDF Schema of Petri net ontology

5.3 OWL-based Petri net ontology

We use the Ontology UML Profile (OUP) (see [29] for details of OUP) for ontology development that is based on the forthcoming ontology language

OWL. OUP provides stereotypes and tagged values for full ontology development. OUP models can be (automatically) mapped into OWL ontologies (e.g. using XSLT). The core Petri net hierarchy shown in Figure 5 is the same for the Petri net ontology represented in OUP. Actually, there is a difference regard of both associations and attributes in the model from Figure 5 since ontology development understands property as a fist-class concept. Thus, we should transform all association between classes as well as all class attributes into OUP's property stereotypes (<<DataTypeProperty>> and <<ObjectProperty>>). An example of this transformation is shown in Figure 10. In this figure we define the <<ObjectProperty>> *element* that is attached (through the <<domain>> association) to the following classes: *StructuralElement*, *Net*, and *Module*. That means each of these classes has a collection of *element*s (one or more). The *element* can take values from the *ModelElement* class. In the similar way, we define the other Petri net properties (e.g. name, reference, id, etc.). In addition, one can note that in OUP we use <<OntClass>> stereotype for representation of ontology classes.

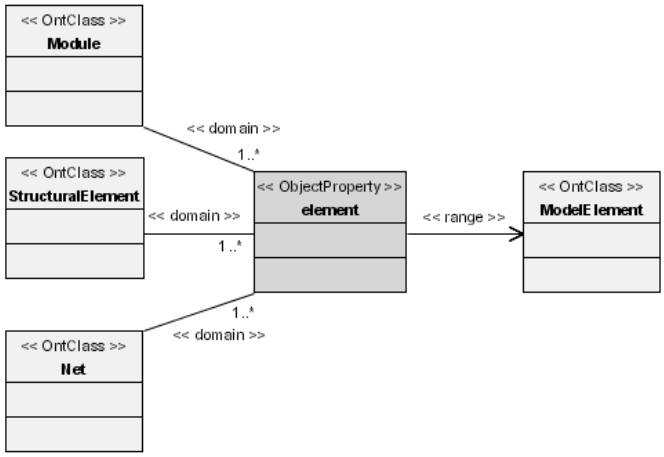


Fig. 10. Collection of Petri net model elements: the OUP element Property

Note that in the OUP Petri net ontology we do not need the *Feature* class since property is the first class in the ontology development. Accordingly, we have <<ObjectProperty>> and <<DatatypeProperty>> to represent the properties in the Petri net ontology. On the other hand, we want to provide support for graphical features. Figure 11 gives an example of the <<ObjectProperty>> *name* that has already been declared as a graphical feature. In this case, the *node* property has as its range (through association <<range>>) the *NodeDescriptor* <<OntClass>>. But, this class is inherited from the *GraphicalFeature*. This class is introduced in the Petri net ontology to be the root class for all the classes that are range for graphical feature. Similarly, we define other graphical features (e.g.

marking). In addition, the *name* property has domain (<<domain>> association): *Net*, and *Node*.

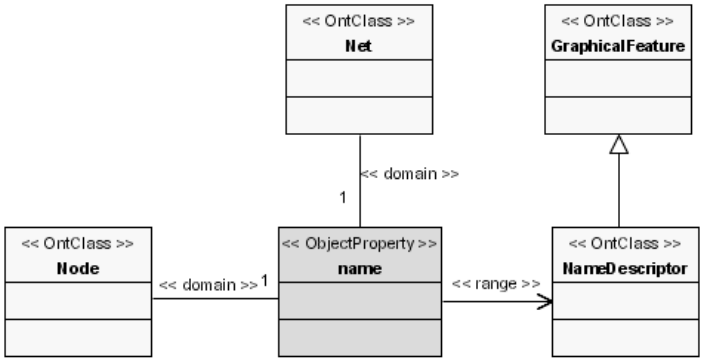


Fig. 11. An example of graphical feature defined in the Ontology UML Profile: name object property

Figure 12 shows how we make a restriction on a Petri net arc using Ontology UML Profile. Using this UML Profile we are able to restrict that a Petri net arc (<<OntClass>> *Arc*) only connects a *Place* and a *Transition*. This statement is expressed as a union (<<Union>>) of two intersections (<<Intersection>>). Our <<OntClass>> *Arc* is an equivalent class (<<equivalentClass>>) with this union. Since these two intersections are defined in symmetric way, we only explain the left one in Figure 12. This (the left) intersection says that an *Arc* takes all values from (<<allValuesFrom>> association between anonymous both <<OntClass>> and <<Restriction>>): *Place* (<<allValueFrom>> dependency between anonymous <<Restriction>> and <<OntClass>> Place) for the *fromNode* property (<<onProperty>> dependency between the anonymous <<Restriction>> and the <<ObjectProperty>> *fromNode*) and *Transition* for the *toNode* property. The second (right) intersection has the opposite statement: *Arc*'s *toNode* property takes all values from *Place*, and *Arc*'s *fromNode* property has all values from *Transition*.

It should be noted that having *Arc* restriction expressed in this way we are able to automatically map the UML model to an ontology language (e.g. OWL). On the contrary, this is very difficult if having these constraints given in OCL or PAL. In Figure 13 we give an excerpt of the Petri net ontology in OWL that we generated using an XSLT for transformation from the OUP (i.e. XMI) to the OWL ontology [30]. This figure illustrates a part of OWL *Arc* class definition that is equivalent with the OUP *Arc* restriction. It is important to note that in the OWL ontology we have logical expressions in the XML form (e.g. the *Arc* restriction) unlike the Protégé PAL constraints that are written in a Lisp-like form. It is more convenient to parse an ontology statement when it is

in an XML format that can be parsed using a standard XML parser as well as transformed using the XSLT mechanism.

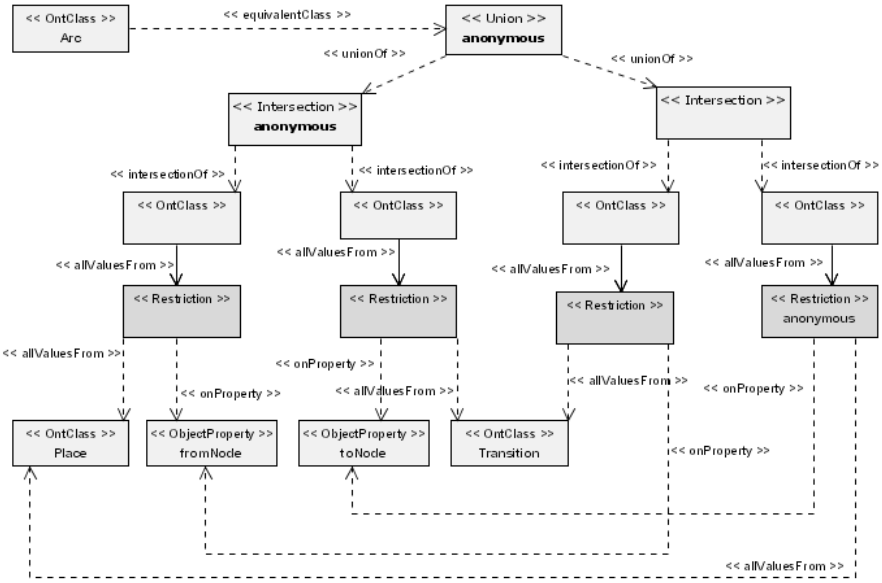


Fig. 12. Restriction that an arc only connects a transition and a place

6 P3 extensions for the Petri net ontology

So far, we have explained ontology design and definition details, but in order to have a practical use of the Petri net ontology we need software tools that would support and employ this ontology. Once more we focus on P3 tool. P3 tool is extended in accordance to the Petri net ontology. Firstly, it is implemented a procedure that saves a Petri net model into an RDF description. We implemented this feature using XSLT, and just invoke an XSLT from P3 tool. This RDF description has agreement with the Petri net ontology. Also, P3 is now able to import models described using the Petri net ontology compliant RDF documents. As P3 has features for importing PNML models, we also implemented an XSLT that transforms the Petri net RDF format into the PNML. In this way, we do not need to implement a special support for parsing RDF documents. Instead, when P3 imports an RDF document it only has to call an XSLT processor. It produces a PNML document, which can be parsed by P3 using the PNML parsing mechanism. Figure 14 illustrates the way P3 supports the Petri net ontology using RDF, PNML, and XSLT.

```
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Class rdf:ID="ModelElement"/>
    <owl:Class rdf:ID="Arc">
        <rdfs:subClassOf rdf:resource="#ModelElement"/>
        <!--OWL subClass statements -->
        <!--. . . -->
        <owl:equivalentClass>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <owl:Class>
                        <owl:intersectionOf rdf:parseType="Collection">
                            <owl:Restriction>
                                <owl:onProperty rdf:resource="#fromNode"/>
                                <owl:allValuesFrom rdf:resource="#Place"/>
                            </owl:Restriction>
                            <owl:Restriction>
                                <owl:onProperty rdf:resource="#toNode"/>
                                <owl:allValuesFrom rdf:resource="#Transition"/>
                            </owl:Restriction>
                        </owl:intersectionOf>
                    </owl:Class>
                    <owl:Class>
                        <owl:intersectionOf rdf:parseType="Collection">
                            <owl:Restriction>
                                <owl:onProperty rdf:resource="#fromNode"/>
                                <owl:allValuesFrom rdf:resource="#Transition"/>
                            </owl:Restriction>
                            <owl:Restriction>
                                <owl:onProperty rdf:resource="#toNode"/>
                                <owl:allValuesFrom rdf:resource="#Place"/>
                            </owl:Restriction>
                        </owl:intersectionOf>
                    </owl:Class>
                </owl:unionOf>
            </owl:Class>
        </owl:equivalentClass>
        <!--OWL subClass statements -->
        <!--. . . -->
    </owl:Class>
    <!--OWL Class, ObjectProperty, and DataTypeProperty statements -->
    <!--. . . -->
</rdf:RDF>
```

Fig. 13. A part Petri net ontology in OWL: Arc class restriction form Figure 12

Although, P3 uses RDF, it does not mean that we have abandoned PNML. On the contrary, since we implemented an XSLT (from RDF to PNML), we have continued to use PNML. Actually, we can say that we brought a new power to PNML because one can use P3 to convert PNML models into RDF, and after that a Petri net model can be validated against the Petri net ontology. In that way, we achieved a semantic validation of Petri net models. Of course, PNML is very useful as it contains well-defined concepts for interchanging Petri net models and now it is a growing proposal that is being used by many Petri net tools. Further, since we have the XSLTs from PNML to Petri net formats of other Petri net

tools we are additionally able to employ their Petri net analysis capabilities.

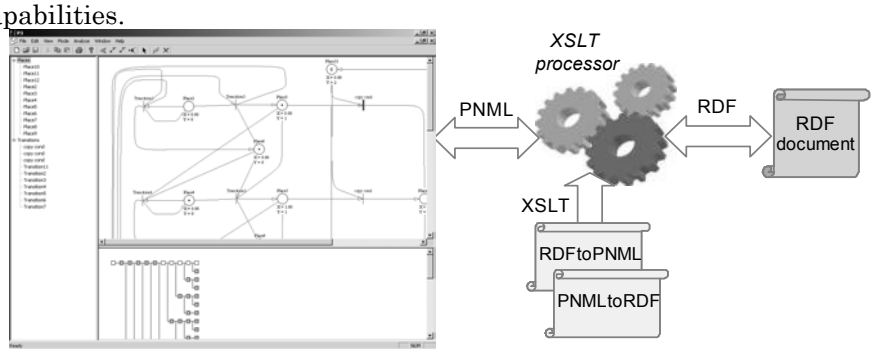


Fig. 14. P3 tool and its support of Petri net ontology using RDF

7 Conclusions

The main idea of this paper to provide the full semantic validation of Petri net model by defining a formal specification of Petri net concepts. We proposed a Petri net ontology as a solution for this issue. The Petri net ontology can detect semantic inconsistencies in a Petri net model, even though the model is syntactically correct (e.g. a transition reference can refer either a transition or other transition reference). Firstly, we designed the ontology using UML and OCL. Then, we used Protégé 2000 to achieve the full support for ontology development. For constraints we used PAL and as a result we have generated RDF Schema describing the ontology. Additionally, we represented the Petri net ontology in the Ontology UML Profile that is based on the OWL. From this UML profile description we generated the OWL ontology. With this solution as a base, we extended P3 tool to use Petri net models described by RDF and developed XSLT from RDF Petri net description to PNML format. The definitions of the Petri net ontology in RDFS and OWL languages and corresponding XSLTs are available at: http://afrodita.rcub.bg.ac.yu/~gasevic/ projects/PNO/.

This paper shows that semantics and syntax in some domain do not exclude one another. On the contrary, on the example of Petri net ontology we show their complementaries. This paper can help in efforts to obtain better formal description of Petri nets and improve their interoperability. Constraints we included in the Petri net ontology can be a direction for the future semantic validation of Petri net models.

The Petri net ontology is going to be extended with an intention to make it possible for different Petri net dialects to use core concepts from the Petri net ontology. Currently, P3 generates SVG documents of Petri net graph, and uses RDF to annotate graphical primitive, so one can restore

original Petri net model from this graphical format. The OWL ontology annotation will be supported in the new P3's version. Further, this annotation principle will be used to develop a Petri net Web-based learning environment as well as to create Learning Object Metadata (LOM) repositories of Petri net models.

References

1. Peterson, J.: Petri net theory and the modeling of systems. Prentice Hall, Englewood Cliffs, New Jersey, USA. (1981)
2. Berthelot, G., et al: A syntax for the description of Petri Nets. Petri Net Newsletter. No. 29, 4-15. (1988)
3. Petri Nets Tools Database, [Online]. Available: http://www.daimi.au.dk/PetriNets/tools/db.html
4. ISO/IEC/JTC1/SC7 WG19: New proposal for a standard on Petri net techniques, ISO/IEC/JTC1/SC7 N2658 (2002)
5. Gruber, T.: A translation approach to portable ontology specifications. Knowledge Acquisition, Vol. 5, No.2, 199-220. (1993)
6. Gómez-Pérez, A., Corcho, O.: Ontology Languages for the Semantic Web. IEEE Intelligent Systems, Vol. 17, No. 1, 54-60. (2002)
7. Breton, E., Bézivin, J.: Towards an Understanding of Model Executability. In Proceedings of the International Conference on Formal Ontology in Information Systems, Ogunquit, Maine, USA, 70-80. (2001)
8. Hansen, K.M.: Towards a Coloured Petri Net Profile for the Unified Modeling Language – Issues, Definition, and Implementation. Internal Center for Object Technology Report COT/2-52, University of Aarhus, Aarhus, Denmark (2001) [Online]. Available: http://www.daimi.au.dk/~marius/writings/cpn.pdf (current Nov. 2003)
9. Peleg, M., Yeh, I., Altman, R.: Modeling Biological Processes using Workflow and Petri Net Models. Bioinformatics, Vol. 18, No. 6, 825-837. (2002)
10. Noy, N. F., et al: Creating Semantic Web Contents with Protégé-2000. IEEE Intelligent Systems, Vol. 16, No. 2, 60-71. (2001)
11. Bause, F., et al: Abstract Petri Net Notation. Petri Net Newsletter, No. 49, 9-27. (1995)
12. Kummer, O., Wienberg, F.: The XML File Format of Renew. In Meeting report from 21st International Conference on Application and Theory of Petri Nets, Århus, Denmark (2000), [Online]. Available: http://www.daimi.au.dk/pn2000/Interchange/-papers/det_04.ps.gz (current Nov. 2003)
13. Kummer, O., et al: Renew - XML Format Guide. [Online]. Available: http://www.renew.de (current Nov. 2003)
14. Jüngel, M., et al: The Petri Net Markup Language. In Proceedings of the 7th Workshop Algorithms and Tools for Petri, Universität Koblenz-Landau, Germany, 47-52. (2000)
15. Weber, M., Kindler, E.: The Petri Net Markup Language. Petri Net Technology for Communication Based Systems, LNCS Vol. 2472, Springer-Verlag, Berlin Heidelberg New York (2003) forthcoming

16. Billington, J., et al: The Petri Net Markup Language: Concepts, Technology, and Tools. In Proceedings of the 24th International Conference on Applications and Theory of Petri Nets, LNCS Vol. 2679. Springer-Verlag, Eindhoven, The Netherlands, 483-505. (2003)
17. Kindler, E., Weber, M.: The Petri Net Kernel - An Infrastructure for Building Petri Net Tools. Software Tools for Technology Transfer (STTT), Vol.3, No.4, Springer-Verlag, Berlin-Heidelberg-New York, 486-497. (2001)
18. Gašević, D., Devedžić, V.: Teaching Petri nets using P3. IEEE Educational Technology & Society. (2004) (forthcoming)
19. Gašević, D, Devedžić, V., Veselinović, N.: P3 - Petri Net Educational Software Tool for Teaching Hardware. Petri Net Newsletter, No. 66, 3-13. (2003)
20. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE, Vol. 77, No. 4, 541-580. (1989)
21. Klein, M., et al.: The Relation between Ontologies and Schema-Languages: Translating OIL Specifications to XML Schema. In Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods, 14th European Conference on Artificial Intelligence ECAI'00, Berlin, Germany (2000)
22. Chandrasekaran, B., et al: What Are Ontologies, and Why Do We Need Them?. IEEE Intelligent Systems, Vol. 14, No. 1, 20-26. (1999)
23. Cranefield, S.: Networked Knowledge Representation and Exchange using UML and RDF. Journal of Digital information, Vol. 1, No. 8 (2001). [Online]. Available: http://jodi.ecs.soton.ac.uk
24. OMG Unified Modeling Language Specification v1.5, OMG document formal/03-03-01 (2003). [Online]. Available: http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.zip
25. Gamma, E., et al: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. (1995)
26. Bock, C.: UML without Pictures. IEEE Software, Vol. 20, No. 5, 33-35. (2003)
27. Baclawski, K., et al: Extending the Unified Modeling Language for ontology development. International Journal Software and Systems Modeling (SoSyM), Vol. 1, No. 2, 142-156. (2002)
28. Kogut, P., et al: UML for Ontology Development, The Knowledge Engineering Review, Vol. 17, No. 1, 61-64. (2002)
29. Djurić, D., Gašević, D., Devedžić, V.: Ontology Modeling and MDA. Journal on Object Technology, Vol. 4, No. 1. (2005) (forthcoming)
30. Gašević, D., Djurić, D, Devedžić, V., Damjanović, V.: From UML to Read-To-Use OWL Ontologies. In Proceedings of the 2nd IEEE International Conference on Intelligent Systems, Vol. II, Vrana, Bulgaria, 485-490. (2004)
31. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L. Patel-Schneider, P.F. and Stein, L.A.: OWL Web Ontology Language Reference. W3C Recommendation (2004). [Online]. Available: http://www.w3.org/TR/2004/REC-owl-ref-20040210

Dragan Gašević is a lecturer of computer science at the Department of Computer Engineering and Informatics, the Military Academy, Belgrade, Serbia and Montenegro where he teaches expert systems, object oriented

programming and modeling, and computer architecture. He is also a researcher at the GOOD OLD AI research group, Department of Information Systems and Technologies, FON – School of Business Administration, University of Belgrade where he is actively included in several international (ProLearn NoE, ARIADNE) and national projects. He has received his BSc, MSc, and PhD degrees from the University of Belgrade in 2000, 2002, and 2004, respectively. His research interests include Semantic Web, knowledge representation, Petri nets, intelligent systems, XML-based knowledge sharing, learning technologies, and software methodologies (MDA). So far, he has authored/co-authored more than 70 research papers published in international and national journals and conferences. He is a student member of the ACM.