Available online at www.sciencedirect.com
SCIENCE DIRECT®
Knowledge-Based Systems xxx (2006) xxx–xxx
Knowledge-Based SYSTEMS
www.elsevier.com/locate/knosys

Petri net ontology

Dragan Gašević [a], Vladan Devedžić [b,*]

[a] School of Interactive Arts and Technology, Simon Fraser University Surrey, 13450 102 Ave., Surrey, BC V3T 5X3, Canada
[b] FON – School of Business Administration, University of Belgrade, POB 52, Jove Ilica 154, 11000 Belgrade, Serbia and Montenegro

Received 22 April 2004; accepted 16 December 2005

Abstract

The paper presents the Petri net ontology that enables sharing Petri nets on the Semantic Web. Previous work on formal methods for representing Petri nets mainly defines tool-specific descriptions or formats for model interchange. However, such efforts do not provide a suitable description for using Petri nets on the Semantic Web. This paper uses the Petri net UML model as a starting point for implementing the ontology. Resulting Petri net models are represented on the Semantic Web using XML-based ontology languages, RDF and OWL. We implemented a Petri net tool, P3, which can be used as a knowledge acquisition tool based on the Petri net ontology.

Keywords: Petri nets; Interoperability; Ontology; Semantic Web

1. Introduction

The main idea of this paper is to propose a suitable way for Petri nets [35] to be used on the Semantic Web, i.e., to enable full semantic interoperability of Petri net models. Currently, Petri net interoperability is possible at the level of syntax for model sharing. It was first introduced in [3], where the authors said that it would be very useful if Petri net researchers could share their Petri net model descriptions. That way more software tools could be used for analyzing the same model. So far, all Petri net interchange attempts have been mainly tool-specific, but with very low (or without any) general acceptance. For example, DaNAMiCS tool (http://www.cs.uct.ac.za/Research/DNA/DaNAMiCS) uses regular text-based format, and Renew tool (http://www.renew.de) uses XML-based format. The *Petri Net Markup Language* (*PNML*) [4] is a recent Petri net community effort that tries to provide XML-based model sharing. PNML tends to be a part of the future ISO/IEC High-level Petri net standard [20]. A particularly important advantage of this approach is that XML documents can be easily transformed using *eXtensible Stylesheet Language Transformations (XSLT)* into other formats (that need not necessarily be XML-based). However, this initiative is also syntactically oriented, i.e., it introduces some constraints that enable validation of documents against their definition (e.g., validating whether an arc connects two nodes of different types, i.e., a place and a transition).

On the other hand, all approaches that formally define Petri nets are generally intended to be used as a basis for developing Petri net tools. Currently, these attempts are defined using *Model Driven Architecture* (*MDA*) standards [29] (e.g., Unified Modeling Language – UML [33], and Meta-Object Facility – MOF [32]).

A suitable way to represent Petri nets is needed in order to reuse them more effectively on the Semantic Web. It requires defining the *Petri net ontology* for semantic description of Petri net concepts and their relationships. The Petri net ontology enables describing a Petri net using Semantic Web languages (e.g., *Resource Description Framework – RDF*, and *Web Ontology Language – OWL*) [15,40]. Petri nets described that way can be inserted into other, non-Petri net XML-based formats, such as *Scalable Vector Graphics* (*SVG*, the XML-based WWW Consor-

* Corresponding author. Tel.: +381 11 3950853/+381 11 3971440; fax: +381 11 461221.
E-mail addresses: dgasevic@acm.org (D. Gašević), devedzic@etf.bg.ac.yu (V. Devedžić).

0950-7051/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2005.12.003

tium (W3C) standard for 2D vector graphics [21]), which makes possible to reconstruct Petri net models using metadata and annotations according to the Petri net ontology.

We defined the Petri net ontology using experience from previous Petri net formal descriptions (metamodel, ontologies, and syntax). They indicate very useful directions for selecting key Petri net concepts and specifying their mutual relations. PNML is of primary importance here – it is closely related to the Petri net ontology. Actually, it is a medium (syntax) for semantics [36]. We additionally empowered the PNML usability by defining mappings to/from the Semantic Web languages (i.e., RDF and OWL).

The next section describes existing formal descriptions of Petri nets: metamodels, UML profiles, ontologies, and syntax. We concentrate on Petri net syntax because most work has been done in solving this problem (we specifically discuss the PNML). Section 3 enumerates advantages of the Petri net ontology, and gives guidelines for its conceptualisation. Section 4 outlines development of the Petri net ontology – its initial design and implementation using UML and Protégé [31] (i.e., RDF Schema (RDFS)-based implementation), whereas Section 5 extends the ontology using an OWL-based UML profile in order to support diversity of Petri net dialects. In Section 6, we present the tool we implemented to support the Petri net ontology, as well as an ontology-driven infrastructure for sharing Petri nets using PNML, XSLT, and RDF. This work is a part of the effort of the Good Old AI research group (http://goodoldai.org.yu) in developing its platform for building intelligent systems, called AIR.

2. Previous work on Petri net sharing

This section discusses previous work in developing a formal description of Petri nets that can be used in different software tools for Petri net: model sharing, software implementation, model validation, and so on. Therefore, we analyze present Petri net: metamodels, UML profiles, ontologies, and syntax. The purpose of this analysis is to identify the Petri net conceptualization underlying these approaches.

2.1. Petri net metamodels

An illustrative and very comprehensive Petri net metamodel is proposed by Breton and Bézivin [6]. They define a Petri net metamodel in the context of the OMG's MDA initiative since they use the MOF for metamodel definition. The authors assume that metamodel is closely related to ontology. Their starting point is that a metamodel defines a set of concepts and relations, i.e., the terminology and a set of additional constraints (assertions). They see each model as encompassing both a static part and a dynamic part. Accordingly, they define Petri net metamodel as a composite 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). In addition, 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 a particular Petri net situation, they introduce the 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 (e.g., Move) needed for Petri net execution, since Petri net execution consists of a sequence of transition firings in regard of place marking.

Note that this proposal is very important for development of Petri net tools. However, in spite of giving a useful classification of Petri net concepts in three different parts, it has a few shortcomings. It does not take into account the existence of different Petri net dialects and Petri net structuring mechanisms (e.g., pages). Moreover, it does not 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 language (e.g., RDF(S)). It also does not suggest what general MOF-based tools can be used for model validation against the metamodel.

Hansen proposes a Petri net UML profile [18]. Defining a Petri net UML profile produces a solution similar to the metamodel-based one, because UML profiles extend the UML metamodel by introducing stereotypes, tagged values and constraints. The main idea of a UML profile is to enable using standard UML tools for different purposes. Hence 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 the stereotypes for places (i.e., initial marking, and colour set), transitions (i.e., guard), and arcs (expression). Also, OCL is used in order to define more precise semantics for the UML profile. Finally, this solution has software implementation as 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. This means that all UML concepts are introduced in the Petri net metamodel, but most of them are needless for the Petri net semantics. Also, this approach has the same limitation as the previous one in terms of 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 Workflow [34], since it has ability to

represent process knowledge. On the other hand, Workflow can be mapped to Petri nets, which allows verification of formal properties and qualitative simulation (i.e., reachability analysis). The authors developed a Petri net ontology using Protégé and a specific graphical user interface (GUI) that extends the standard GUI of the Protégé tool. Actually, this GUI provides graphical tools for all Petri net concepts (Places, Transitions, and Arcs). 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, it has serious limitations. It covers only Time Petri nets, and no other kinds of Petri nets. It neither defines Petri net structuring mechanisms, nor provides precise constraints (e.g., types of an arc's source and target nodes that can be done using Protégé Axiom Language (PAL) constraints). Finally, it does not enable using other ontology languages for representing the Petri net ontology (e.g., DAML or OWL).

2.3. Petri net syntax

We analyzed two kinds of Petri net syntax: general-purpose and tool-specific. Tool-specific syntax is analyzed in the following tools: DaNAMiCS and Renew. DaNAMiCS (http://www.cs.uct.ac.za/Research/DNA/DaNAMiCS) 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 and structural analysis, as well as a few performance analyzers, both simple and advanced. For model recording, DaNAMiCS uses a file format with the *bim* extension. The *bim* format has many internal marks whose documentation is not available to the authors of this paper. The second file format used by DaNAMiCS has the *wam* extension and these files are used for model import (menu *File*, option *Import net*). The authors are not aware of any documentation for this format. However, it is a text-based format with a structure that evidently corresponds to a certain Petri net object. It has been analyzed and compared with the models obtained by importing files of this format into DaNAMiCS. The meaning of every format element has been obtained as well. An example Petri net description in *wam* format is shown in Fig. 1a, whereas the corresponding graph of that net is shown in Fig. 1b.

Renew Petri net software tool can be freely downloaded from http://www.renew.de, and is also Java-based. 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 are: support for synchronized channels as an advanced communication mechanism; support for the modeling object-oriented concepts; support for numerous arc types; a rich graphical environment. The XML document model description is defined using *Document Type Definition* (*DTD*) [26,28]. The assumptions included in the formulation of this DTD are the same as in PNML, since they use the same elements to describe the net (XML tag `net`), place (`place`), transition (`transition`) and arc (`arc`), and have similar content models. Each element in the Renew's XML format can have a graphical information and an arbitrary number of annotations.

Abstract Petri Net Notation (APNN) is the first attempt to define a general-purpose Petri net syntax (i.e., it has ability to describe different Petri net dialects) [2]. To increase the readability of this notation, the keywords are similar to LATEX commands. This notation should satisfy the following requirements:

- 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;

a

```
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)
]
]
```

b

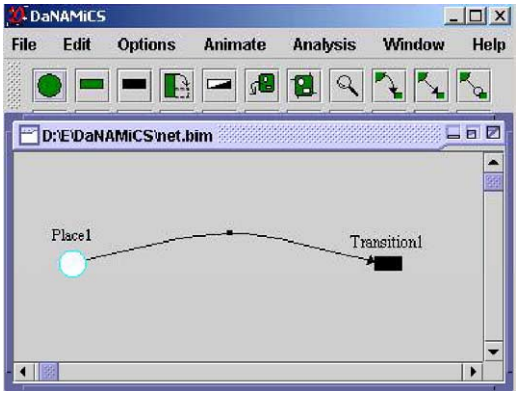

Fig. 1. Wam format for Petri net description, which can be imported into DaNAMiCS: (a) example of a net described in this format, (b) graphical representation.

- *Modularity and hierarchy* – the Petri net description in the file should be reusable;
- *Readability* – text notation should be easy to transform into a human-readable format, as well as suitable for printing.

This notation does not store information about 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 is that terminals are written using lower-case letters and non-terminals using upper-case letters. A short review of APNN, using an example of P/T nets, is shown in Fig. 2. This grammar is useful from the extensibility and modularity perspectives. These requirements should be supported by the Petri net ontology as well, and they are used as guidelines for defining the PNML.

2.4. Petri Net Markup Language

The Petri net community is working for three years already on development of the Petri Net Markup Language (PNML) [22,41] that might become a part of the future High-level Petri nets ISO/IEC standard [20]. The PNML is a proposal that is based on XML. The design of PNML was governed by the following principles [4]:

- *Flexibility* – PNML must be able to represent any kind of Petri net with its 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 specifies eligible labels for a particular Petri net type.
- *Compatibility* – Unlimited exchange of information between different types of Petri nets should be provided. The 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.

PNML specification is based on the PNML technology metamodel that formulates the structure of PNML documents. Actually, this metamodel defines the basic Petri net concepts (places, transitions, and arcs), as well as their relations that can be presented in a PNML document. Current version of PNML is 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 defined using XML Schema as well. The full PNML definition, as well as a few examples of PNTD can be found on the PNML home page (http://www.informatik.hu-berlin.de/top/pnml/about.html).

PNML, being more matured, is currently supported (or will be supported soon) 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 formats considerably similar to PNML (e.g., Renew). In this paper we emphasize PNK – a tool that is closely related to the PNML technology [4]. PNK is not just a Petri net tool, but also an infrastructure for building Petri net tools [23]. It is not limited to a single 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

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

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

c
```
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
```

Fig. 2. Abstract Petri net notation: (a) the set of terminal symbols, (b) the set of non-terminal symbols, (c) the set of grammar productions.

that of PNML. Each place, transition, arc, or even the net, may contain several labels related to the Petri net type.

For educational purposes, we developed *P*3,a Petri net tool that supports PNML [12]. P3's details are discussed in the Section 6.

3. The Petri net ontology guidelines

As we have seen so far, Petri net formats use different concepts for defining its syntax. Some of these syntax-based approaches actually have problems with syntax validation. For instance, it is very difficult to validate a text-based document (i.e., DaNAMiCS) without a special-purpose software for checking the corresponding format. A slightly better solution is to use DTD for XML definition as the Renew does. 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 DTD documents have a non-XML structure.

The W3C 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 no full support for describing semantics [24]. In fact, XML Schema is only a way for defining syntax. For example, current PNML definition does not have the 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 [4]. In order to perform this kind of validation, one must use some specific tools (e.g., for PNML it is proposed to use the *Jing* validator), but these tools are not widely known in the XML community. Furthermore, if we want to share Petri net models not only with Petri net tools, we must have a formal way for representing Petri net semantics since we can not expect that a non-Petri net tool performs semantic validation.

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 a formal way for representing shared conceptualization in some domain [16]. Ontology has formal mechanisms to represent concepts, concept properties, and relations between concepts in the domain of discourse. With the Petri net ontology, we can overcome validation problems that we have already noticed. However, the Petri net ontology does not exclude current Petri net formats (especially PNML). Ontology is closely related to syntax, in the sense that syntax should enable ontological knowledge sharing [7]. With the Petri net ontology, we can use ontology development tools for validation of Petri net models (e.g., Protégé). Also, having the Petri net ontology one can use Semantic Web languages for representing Petri net models (e.g., RDF, RDF Schema -RDF, DAML + OIL, OWL, etc.) [15]. Thus, we show how PNML can be used as a guideline for the Petri net ontology.

Accordingly, we extracted common Petri net concepts form the PNML, as well as from other analyzed Petri net syntax formats. The review of common concepts is given in Table 1. Attributes of particular concepts are written in normal face font in order to distinguish them from their related elements (bold face font) for each format. This also constitutes the basic guidelines for building the Petri net ontology. The main purpose of the ontology is to construct semantic armature around which the extracted Petri net concepts would be built [9].

An additional important requirement for the Petri net ontology is to support different Petri dialects. It is obvious that each Petri net dialect defines its conceptualisation. In our approach, the Petri net ontology should have its core. This core should contain the common Petri net concepts.

Table 1
Review of Petri net concepts, attributes and contents extracted form existing Petri net syntax formats: PNML, APNN, DaNAMiCS format, and Renew 's XML format for model sharing

Concept	APNN	DaNAMiCS	Renew	PNML
Net	Identifier, **place**, **transition**, **arc**	–	Identifier, type, **place**, **transition**, **arc**, **annotation**	Identifier, type, **place**, **transition**, **arc**, **page**, **reference place and transition**
Place	Identifier, **name**, **initial marking**, **capacity**	Name, initial marking, marking, graphical information	Identifier, type, **graphical information**, **annotation**	Identifier, **name**, **initial marking**, **graphical information**
Transition	Identifier, **name**	Kind (immediate and time), name, graphical information, possibility, time, delay	Identifier, type, **graphical information**, **annotation**	Identifier, **graphical information**, **name**, **tool specific**
Arc	Identifier, source, target, multiplicity	Source, target, multiplicity, graphical information	Identifier, source, target, type, **graphical information**, **annotation**	Identifier, source, target, **graphical information**, **multiplicity**
Graphical information	–	Position	**Position**, **size**, **text size**, **color**...	**Absolute position**, **relative position**
Initial marking	Init	Contained by place tag	Annotation: type, identifier, **text**, **graphical information**	Value, **graphical information**
Name	Value	Contained by place tag	Annotation: type, identifier, text, **graphical information**	Value, **graphical information**

Furthermore, each Petri net dialect extends this ontology core with its specific concepts. In the next section we define the Petri net ontology using UML and Protégé ontology development tool.

4. The Petri net ontology – initial implementation

There are many different ways to develop an ontology [39] and one can use different tools for ontological engineering tasks (Protégé, OilEd, OntoEdit, etc.). In order to develop the Petri net ontology, we decided to use UML [8]. UML was suitable because it is a generally accepted and standardised tool for analysis and modeling in software engineering. We were also able to employ UML-based Petri net descriptions existing within the PNML definition [4]. However, neither UML tools nor the UML itself are intended to be used for ontology development. Thus, in order to achieve more precise Petri net definition than a UML model provides, it is necessary to use an ontology development tool. We decided to use Protégé 2000 [31] since it is a popular tool for ontology development and can import UML models. This is enabled by Protégé's UML backend that imports UML models (represented in *XML Metadata Interchange* (*XMI*) format) into a Protégé ontology.

4.1. The underlying idea

The hierarchy of core concepts of the Petri net ontology is shown in Fig. 3. In our design of the Petri net ontology, there is a single root element that we call *ModelElement.* This element is the parent for all elements of Petri net structure. The name of this class is *ModelElement* because the UML metamodel uses the same name for its root class [33]. A Petri net (the *Net* class) can contain many different *ModelElements. ModelElement* and *Net* have the ID attribute (unique identifier) of String type, and *Net* has also an attribute that describes the type of the Petri net. It is in accordance with PNML. The three main Petri net concepts (place, transition, and arc) define the structure of a Petri net, and they are represented in Fig. 3 with the corresponding classes (*Place*, *Transition*, and *Arc*). Places and transitions are kinds of nodes (*Node*). In some Petri nets, an arc connects two nodes of different kinds. This constraint can be represented using 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)
```

However, it is important to say that this is not a general the Petri net ontology statement, since there are Petri net dialects where an arc can connect, for instance, two transitions. In this case, one understands that there is a place between transitions [37]. Hence, we did not include this statement in the core Petri net ontology, but it should be

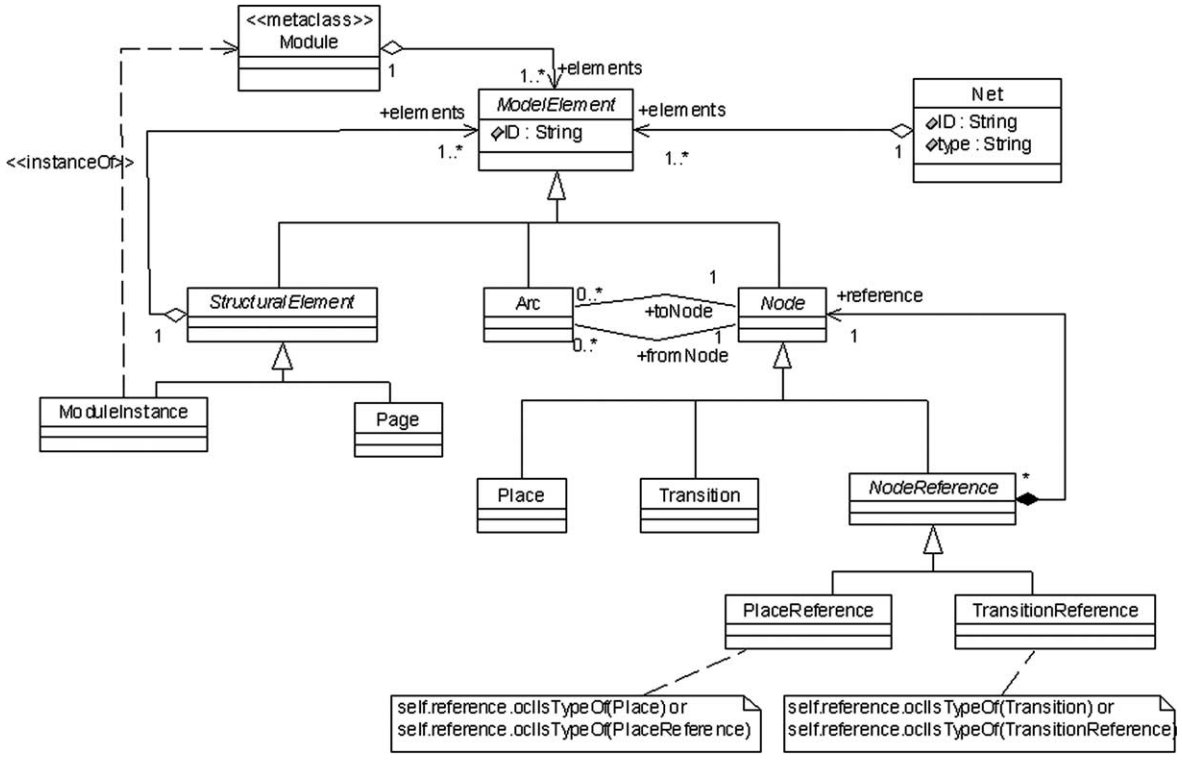

Fig. 3. The Petri net ontology – hierarchy of core Petri net concepts.

defined in ontology extensions for different Petri net dialects.

The *Node* class is introduced in the ontology in order to have a common way to reference both places and transitions. In order to make Petri net models easy to maintain, 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 (the *Page* class), and modules (the *Module* class). A *Page* may consist of other Petri net *ModelElements* – it may even consist of other pages. A *NodeReference*, which can be either a *TransitionReference* or a *PlaceReference*, represents an appearance of a node. The *Decorator* design pattern [11] was used to represent referencing of a *NodeReference*. Here, there are also constraints: a *TransitionReference* can refer to either a *Transition* or another *TransitionReference*, while a *PlaceReference* can refer to either a *Place* or another *PlaceReference*. We show these constraints using OCL in Fig. 3. These constraints also affect the OCL constraint for arcs that we have already described, but we do not show their interaction due to the limited size of this paper. Unlike the OCL statement for arcs, this statement can be applied on all Petri net dialects.

The second kind of structuring mechanisms are modules. A *Module* consists of *ModelElements*, and it can be instantiated (much like an object is instantiated from a class in the object-oriented paradigm). Accordingly, *Module* is a metaclass (the stereotype in Fig. 3), 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 in the Petri net ontology a description of features and in Fig. 4 we shortly depict how these features have been added. The root class for all features is *Feature*. This is also similar to the UML metamodel. 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 graphical information that can consist of, for instance, position (the *Position* class 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 (the *Token* class). In order to support token colors, the *Token* class is abstract. In Fig. 4 we show a case when there are no colors attached to tokens; instead, we just take into account the number of tokens (the *IntegerToken* class).

Attaching a new feature to a Petri net class requires just adding an association between a class and a feature.

Fig. 5 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.

A UML description is a convenient way for representing the Petri net semantics. Also, this Petri net ontology can be used as a Petri net metamodel in future Petri net implementations that can take advantage of the MDA concept and repository-based software development [5]. However, it does not let us semantically validate Petri net models. For example, we cannot use OCL statements to perform this task. In addition, UML attributes and ontology properties are semantically different concepts. Unlike a UML's attribute, ontology property is a first-class concept that can exist independently of any ontology class [1].

There are to ways to further refine the Petri net ontology. The first one is to use a UML profile [25] for UML-based ontology development. The second one recommends using standard ontology development tools. We decided to use: 1. Protégé 2000, since it provides all the necessary ontology development features (constraints and support for ontology languages), but it also has the ability to use

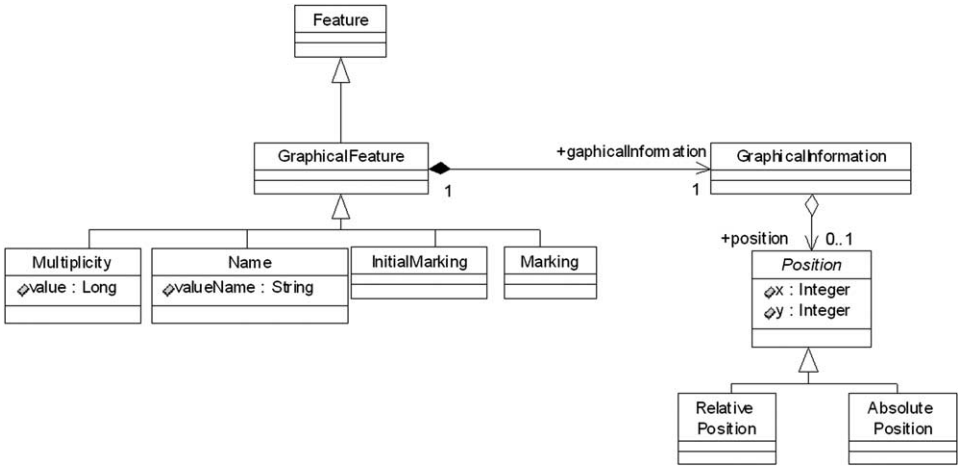

Fig. 4. Property hierarchy of the Petri net ontology.

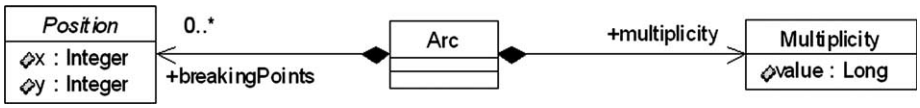

Fig. 5. An example of how features are attached to a class in the Petri net ontology: a Petri net node has position and name.

the UML models we have shown; 2. The Ontology UML profile [10] that is based on OWL.

4.2. The Petri net ontology implementation using Protégé 2000

We can precisely define the Petri net ontology in Protégé 2000. We can differ between a class and a metaclass (e.g., *Module* – a metaclass, *ModuleInstance* – a class), we can use different Semantic Web languages provided through Protégé's backends (RDF(S), OWL, DAML + OIL) to represent the Petri net ontology, and we can specify the constraints that we defined in the UML model using OCL (e.g., PAL). We can then validate all ontology instances using these constraints, and detect if there is any instance that does not conform to some of the constraints.

After the initial UML design of the Petri net ontology, it was imported into the Protégé using Protégé's UML backend (http://protege.standford.edu/plugin/uml). This plug-in has the ability to read an XML format (i.e., 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 add manually all the slots that are represented 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 Fig. 6.

Of course, Protégé does not have the ability to transform OCL constraints into PAL constraints. Thus, we have also manually reconstructed all OCL-defined constraints from the UML model of the Petri net ontology into a set of corresponding PAL constraints. For instance, a constraint attached to *TransitionReference* that can refer only to a *Transition* or another *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 to the ontology.

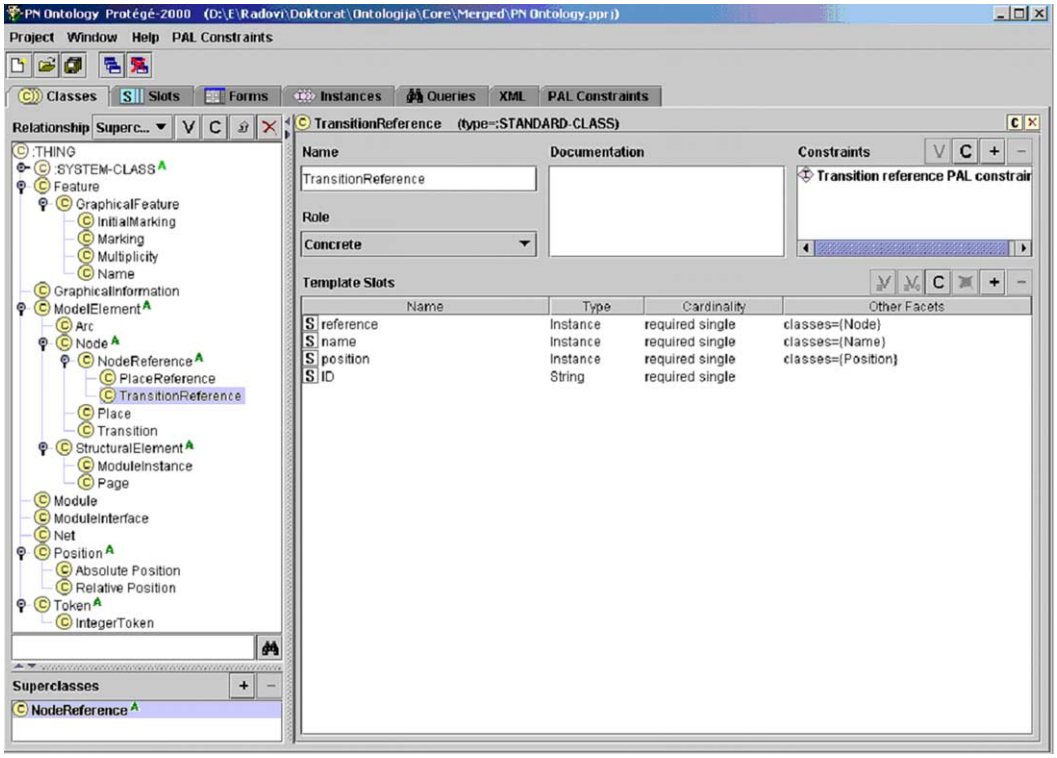

Fig. 6. The Petri net ontology in Protégé 2000.

Applying the same principle, we inserted the constraints for *PlaceReference*, and *Arc*.

Using Protégé we generated the RDFS that describes the Petri net ontology. One can use it for reasoning about a document that contains a Petri net model. Listing 1 shows an excerpt of this RDFS. This listing depicts how RDFS defines the classes for *ModelElement*, *Node*, *Transition*, *Place*, *Arc*, and *ArcType*. Also, this listing 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 Listing 1. These Protégé extensions are manifested by namespace a. For example, they are used to define cardinality (`a:maxCardinality`, `a:minCardinality`), to refer to a PAL constraint (`a:slot_constraints`), etc. Of course, this is neither a limitation of the Petri net ontology nor of the Protégé tool, but of RDFS itself. Most of such limitations are overcome in OWL [40], but this discussion is beyond the scope of this paper.

On the other hand, one can see that the RDFS/Protégé Petri net ontology does not take into account Petri net dialects. In this version of the Petri net ontology we can add Petri net dialect-specific properties or constraints, but we have no ability to distinguish between the core concepts form the Petri net ontology and concepts Petri net dialect-specific concepts. One possible solution is to use XML/RDF namespace mechanism. But, this solution is also limited to use in Protégé. We need a better way to represent ontology modularization. Accordingly, we decided to use OWL and an OWL-based UML profile in order to overcome these Petri net ontology limitations.

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

    <!-- ... -->

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

    <!-- ... -->

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

    <!-- ... -->

</rdf:RDF>
```

Listing 1. A part of the RDF Schema of the Petri net ontology.

5. OWL-based Petri net ontology

For ontology development we use the Ontology UML profile (OUP) (see [10] for details) that is based on OWL [40]. The OUP provides stereotypes and tagged values for full ontology development. OUP models can be (automatically) transformed into OWL ontologies (e.g., using XSLT) [14].

Using the OUP, one can represent relations between the core concepts of the Petri net ontology and the specifics of a Petri net dialect. For that purposes we suggest using the OUP's package mechanism. In the OUP, we attach "ontology" to a package. That means the package is an ontology. Accordingly, we can put all core concepts of the Petri net ontology in an `<<ontology>>` package. If we extend the Petri net ontology with concepts of a Petri net dialect we only need to create a new `<<ontology>>` that would be related with the core `<<ontology>>` through the `<<include>>` dependency. In Fig. 7 we illustrate this extension principle.

The example from Fig. 7 depicts how we extend the core Petri net ontology (`<<ontology>>` *Petri net core*) with concepts of Time Petri nets and Upgraded Petri nets. An additional advantage of this approach is that we have the ability to merge concepts from a number of ontologies (i.e., `<<ontology>>` packages). As a result we obtain one ontology definition, for instance, in OWL (by applying XSLT). Comparing with the current PNML proposal for the PNTDs [4] one can see that this approach improves

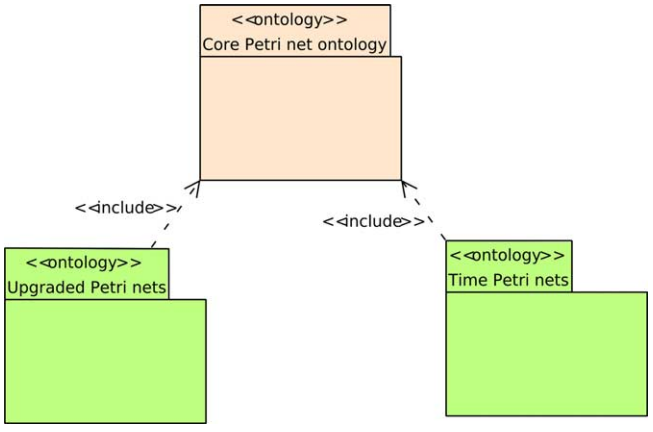

Fig. 7. Extension mechanism of the Petri net ontology: support for Petri net dialects.

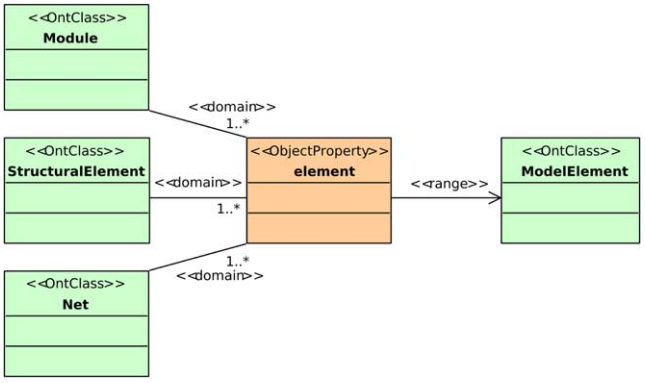

Fig. 8. Collection of Petri net model elements: the OUP element property.

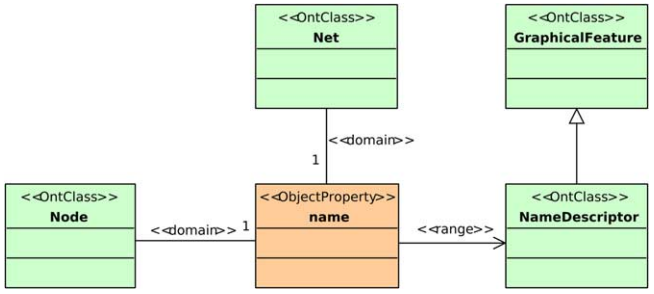

Fig. 9. An example of a graphical feature defined in the Ontology UML profile: name object property.

the maintainability of Petri net concepts, and better supports reusability of the Petri net ontology concepts. So far we have defined the Petri net ontology extensions for: P/T nets, Time Petri nets [30], and Upgraded Petri nets [38].

The rest of this section focuses on the discussion of the core concepts of the Petri net ontology. The core Petri net hierarchy, which is shown in Fig. 3, is the same for the Petri net ontology represented in the OUP. Actually, there is a difference with regard to both associations and attributes in the model from Fig. 3, since ontology development understands property as a fist-class concept. Thus, it is necessary to transform all association between classes as well as all class attributes into the OUP property stereotypes (<<DataTypeProperty>> and <<ObjectProperty>>). An example of this transformation is shown in Fig. 8. The <<ObjectProperty>> *element* defined in Fig. 8 is attached (through the <<domain>> association) to the following classes:

StructuralElement, *Net*, and *Module*. It means that each of these classes has a collection *of elements* (one or more). The *element* can take values from the *ModelElement* class. In a similar way, we define the other Petri net properties (e.g., name, reference, id, etc.). In addition, one can note that in the OUP we use the <<OntClass>> stereotype for representation of ontology classes.

Note that in the OUP Petri net ontology we do not need the *Feature* class since property is the first class in ontology development. Accordingly, we have <<ObjectProperty>> and <<DatatypeProperty>> that represent properties in the Petri net ontology. On the other hand, we want to provide support for graphical features (*GraphicalFeature*). Fig. 9 gives an example of the <<ObjectProperty>> *name* that has already been declared as a

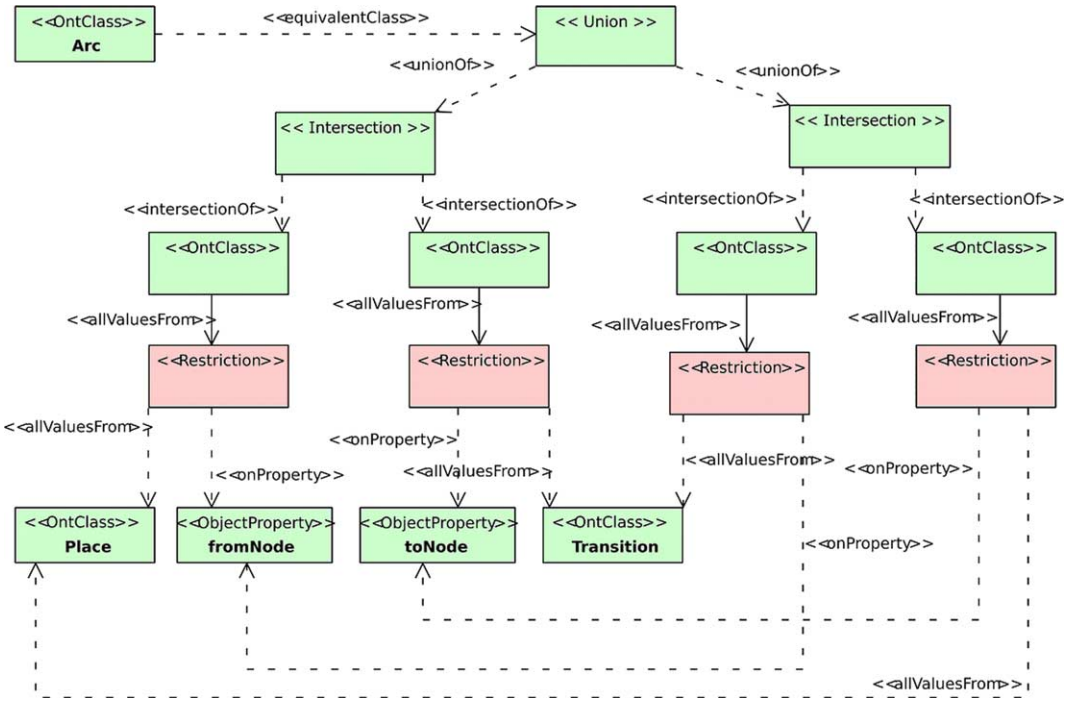

Fig. 10. Restriction: an arc only connects a transition and a place. This restriction is represented in the Ontology UML profile that can be transformed to OWL usingXSLT.

graphical feature. In this case, the *name* property has as its range (through the <<range>> association) the *NameDescriptor* <<OntClass>>. However, this class is inherited from the *GraphicalFeature*. *GraphicalFeature* is introduced in the Petri net ontology to be the root class for all the classes that constitute the range for a graphical feature. Similarly, we define other graphical features (e.g., marking). In addition, the *name* property has domain (the <<domain>> association): *Net* and *Node*.

Fig. 10 shows how we make restriction on a Petri net arc using the Ontology UML profile. Note that this restriction is not a part of the core Petri net ontology since we have already mentioned that is not a generally applicable rule for all Petri net dialects. However, most of Petri net dialects have this restriction, and hence we take it into account here. The restriction means 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>>) of this union. Since these two intersections are defined in a symmetric way, we only explain the left-hand one in Fig. 10. This intersection says that *anArc* takes all values from (<<allValuesFrom>> association): <<OntClass>> *Place* for the *fromNode* property and <<OntClass>> *Transition* for the *toNode* property.

```
<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>
```

Listing 2. A part of the Petri net ontology in OWL: the Arc class restriction from Fig. 3.

The second (right) intersection specifies the opposite statement: *Arc's toNode* property takes all values from *Place*, and *Arc sfromNode* property has all values from *Transition*.

It should be noted that having the *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 these constraints are specified in OCL or PAL. Listing 2 shows an excerpt of the Petri net ontology in OWL. It was generated using an XSLT for transformation from the OUP ontology (i.e., XMI) to OWL. The listing illustrates a part of OWL *Arc* class definition that is equivalent to the OUP *Arc* restriction. It is important to note that in the OWL ontology logical expressions take an 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 represented in an XML format using standard XML parser, as well as transform it using the XSLT mechanism.

6. Ontology-driven Petri net sharing

In order to show practical tool support for the Petri net ontology, we overview the P3 tool. This tool has been initially developed for Petri net teaching [12], but we extended it, and thus it can be used in conjunction with the Petri net ontology.

6.1. P3 – Petri net tool

Being based on the PNML concepts, P3 is compatible with PNML. The P3 tool supports P/T nets and Upgraded Petri nets. The major parts of the P3 architecture are the following:

- *Petri net structure* – The central part of the structure is a Petri net that includes the basic Petri net concepts: places, transitions, and arcs [30]. Important parts of the Petri nets that pertain to their structure are *marking* and *initial marking*, even though these concepts are not real parts of the Petri net structure.
- *Petri net graph* – It is closely related to the Petri net structure. Roughly speaking, Petri net graph is a graphical notation for the Petri net structure.
- *Petri net simulation* – It implements two different modes of simulation: parallel execution of all enabled transitions with a previous conflict resolution, and single execution of an enabled transition.
- *Petri net analysis tools* – P3 supports two well-known Petri net analysis tools – *Reachability Tree* and *Matrix Equations* [35]. We also introduced new analysis tools appropriate for teaching purposes – *Fireability Tree* and *A cyclic firing graph*.
- *Petri net model sharing* – P3's model sharing is based on PNML.

A P3 screenshot is shown in Fig. 11a. The P3's architecture is shown in Fig. 11b. The Petri net class organization is shown on the left in Fig. 11b, whereas the supported formats are on the right side.

The formats supported by P3 are the main point of interest for the Petri net ontology. The P3's model sharing mechanism is based on using PNML. All other formats are implemented in P3 using XSLT (from the PNML). Accordingly, P3 can export to the following Petri net tool formats:

- *DaNAMiCS* – A tool that uses an ordinary text format. Model exchange with this tool is useful since DaNAMiCS provides many Petri net analysis tools such as: matrix invariants and transition matrices, structural analysis, as well as some simple and advanced performance analyses.
- *Renew* – A tool that uses another XML-based format. Its advantages include: support for synchronization channels, which is an advanced communication mechanism; support for modeling object-oriented concepts; a number of supported types of arcs; rich graphical environment [27]. This tool can be used for modeling different agent-based systems.
- *PNK* – A tool that uses PNML, but since there are some differences between this PNML application and the P3's PNML, we had to implement an XSLT. PNK is not focused on a specific Petri net dialect; it is possible to use PNK with Petri net dialects with specific extensions [23].

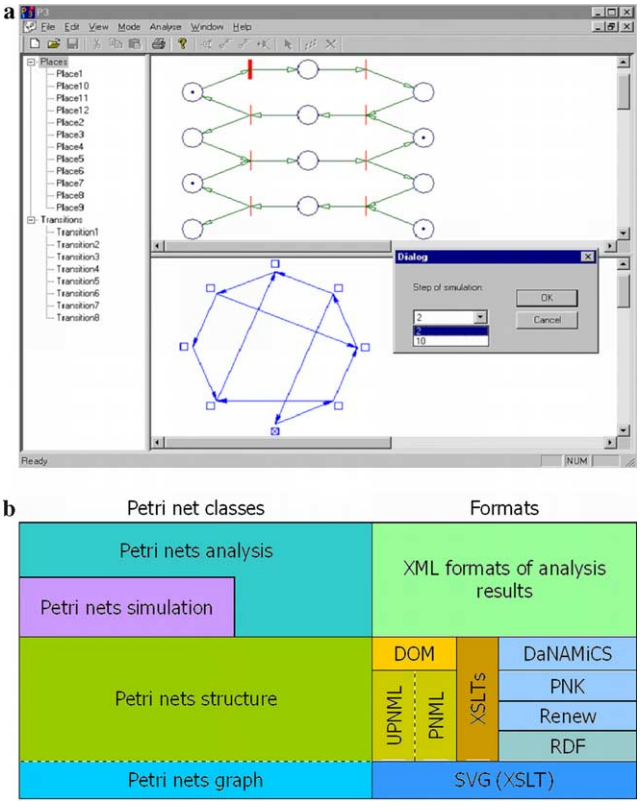

Fig. 11. P3, ontology-based Petri net tool: (a) P3 screenshot; (b) P3 architecture: class organization and supported XML formats.

PIPE – A tool that uses PNML. No additional transformation is needed in order to exchange Petri net models between P3 and PIPE. The models can be exchanged in both directions: P3 ↔ PIPE. PIPE provides different Petri net analysis tools such as invariant analysis and state space analysis.

Our experiences obtained in exchanging Petri net models between P3 and other Petri net tools can be found in [13].

P3 tool has the ability to generate RDF description of a Petri net. This P3's feature is also implemented using XSLT. The generated RDF is in accordance with the Petri net ontology (in its RDFS form). By having the RDF description of a Petri net model we are able to additionally describe other non-Petri net documents – we can incorporate a Petri net into, the for example, SVG documents. We also implemented the XSLT for the opposite direction, i.e., to transform RDF into PNML, and hence we can analyze RDF-defined Petri nets using standard Petri net tools. Additionally, other XSLTs that we have explained above can be applied on PNML to obtain tool specific formats (e.g., DaNAMiCS).

P3 implements conversion of the PNML Petri net model description to SVG. Since this format can be viewed in standard Web browsers (e.g., Internet Explorer), it is suitable for creating, for instance, Web-based Petri net teaching materials. Learning objects, created in this way, have their underlying semantics described in RDF form, and can be transformed into PNML as well as analyzed with standard Petri net tools. P3 provides two kinds of SVG annotations [17]:

1. *As embedded metadata* – An RDF description is incorporated in SVG documents. The standard SVG has the metadata tag as an envelope for metadata.
2. *As remote metadata* – An RDF description is in a separated document.

Listing 3 shows a simple RDF-annotated SVG document using the first kind of annotation. Within the metadata tag is the RDF description of SVG primitives that draw Petri net graph (e.g., `g`, `path`, `text`, `rect`, `circle`). The Petri net *n*1 (`net` tag) contains a place (*p*1), a transition (*t*1), and an arc (*a*1). The arc *a*1 connects the place *p*1 and the transition *t*1 (*t*1 input arc). Each SVG group of primitives that correspond to a Petri net element is enveloped with the `g` tag (group in terms of SVG) with the id attribute. For each group of SVG primitives there is an RDF description (e.g., `Place`) in the metadata part of the document. The `id` attribute of an SVG group has the same value as the `rdf:about` attribute of the corresponding RDF description. Listing 3 shows just a part of the RDF description due to its size. In addition, this RDF description contains appropriate elements for each

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<svg width="2000px" height="2000px">
    <metadata>
        <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://protege.stanford.edu/PN_Ontology#"
        xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
            <Place rdf:about="p1" ID="p1" rdfs:label="p1">
                <position rdf:resource="p1posGIdesc"/>
                <initialMarking rdf:resource="p1initmarking"/>
                <marking rdf:resource="p1marking"/>
                <name rdf:resource="p1name"/>
            </Place>
            <Transition rdf:about="t1" ID="t1" rdfs:label="t1">
                <position rdf:resource="t1posGIdesc"/>
                <name rdf:resource="t1name"/>
            </Transition>
            <Arc rdf:about="a1" ID="a1" arcType="normal" rdfs:label="a1">
                <fromNode rdf:resource="p1"/>
                <toNode rdf:resource="t1"/>
                <multiplicity rdf:resource="a1multi"/>
            </Arc>
            <!-- RDF descriptions for properties of Place, Transition,
            and Arc are omitted due to their length  -->
            <Net rdf:about="n1" ID="n1" rdfs:label="n1">
                <elements rdf:resource="p1"/>
                <elements rdf:resource="t1"/>
                <elements rdf:resource="a1"/>
            </Net>
        </rdf:RDF>
    </metadata>
    <title>Petri net - SVG format</title>
    <desc>Exported from the P3 Petri net tool</desc>
    <g id="a1">
        <path class="Line" d="M76 176 L 217 176"/>
        <g transform="translate(217 ,176)">
            <g transform="translate(-22, 0)">
                <g transform=" rotate(0)">
                    <path class="Line" d="M 0 -3 L 7 0"/>
                    <path class="Line" d="M 7 0 L 0 3"/>
                </g>
            </g>
        </g>
    </g>
    <g id="p1">
        <circle class="Place" r="15" cx="76" cy="176"/>
        <text class="Label" text-anchor="middle" x="76" y="160">p1</text>
    </g>
    <g id="t1">
        <rect class="Transition" x="202" y="161" width="30" height="30"/>
        <text class="Label" text-anchor="middle" x="217" y="160">t1</text>
    </g>
</svg>
```

Listing 3. An example of an RDF-annotated SVG document that contains a simple Petri net (n1) consisting of a place (p1), a transition (t1), and an arc (a1) that connects p1 and t1.

Petri net property (e.g., `position`, `marking`, `initialMarking`, `name`, etc).

6.2. Semantic Web infrastructure for Petri nets

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 continued using PNML. Actually, we enhanced PNML because one can use P3 to convert a PNML model to RDF, and then the Petri net model can be validated against the Petri net ontology. That way, we achieved a semantic validation of Petri net models. Of course, PNML is very useful since it contains well-defined concepts for Petri net models interchange and it is now used by many Petri net tools. Furthermore, since we implemented the XSLTs from PNML to Petri net formats of other Petri net tools, we can also employ PNML's analysis capabilities.

In Fig. 12 we show the Semantic Web infrastructure for Petri nets, which is now implemented. This infrastructure summarizes all major features of P3. The central part of this infrastructure is PNML, since it would be (probably) a part of the future High-level Petri net standard [20]. P3 can be linked with other Petri net tools though PNML (e.g., with PIPE), or by using additional XSLTs on PNML models (DaNAMiCS, Renew, and PNK). Also, P3 has XSLTs for conversions between PNML and RDF in both directions. Besides, P3 generates SVG by using XSLT form PNML to SVG. An XSLT is developed to generate the RDF-annotated SVG from the PNML. We have also developed the XSLT that transforms RDF-annotated SVG documents to PNML. This XSLT is based on the XSLT form RDF to PNML. Hence, we have XSLTs for conversions between PNML and SVG in both directions.

Since we use RDF for describing Petri nets, we can use Protégé tool to create Petri net models. Of course, this is pretty tedious since Protégé does not have graphical tools for Petri net modeling. However, some future implementations can develop this tool in the form of a Protégé's plug-in. A good starting point can be the GUI developed in [34]. A more important feature we obtain from Protégé is its capability to semantically validate documents containing ontology instances (e.g., RDF documents). In other words, we can import an RDF document (that describes a Petri net) into Protégé together with the Petri net ontology. Then we can validate this RDF document using the ontology (as well as PAL constraints we presented in Section 5).

In future P3 implementations, we will provide support for OWL. It will enable P3 to import/export Petri net models represented in OWL. These OWL-based Petri net models can be validated against the Petri net ontology we defined in Section 6. For that purpose we can also use Protégé and its OWL plug-in (http://protege.stanford.edu/plugins/owl/).

Our Petri net Semantic Web infrastructure understands the integration of the Petri net ontology with Web Services that can analyze Petri nets. Currently, we are integrating the Petri net ontology with the Petri net Web Service [19]. This Web Service has Web methods that perform a simulation. The Web Service is based on PNML (its input and output are represented in PNML). We believe that in the future there will be many different Petri net Web Services, which will have different Petri net analysis features. These Web Services may replace the need to use current Petri net tools (or some of them, since some Petri net analyses are very time- or space-consuming).

In a hypothetical scenario, a Web Service would understand, for example, that we have a Web-based application showing a Petri net using the RDF-annotated SVG. When the user requests a simulation step from the Web application, the application first transforms the SVG (i.e., RDF) format of the Petri net model into PNML. Then it submits the PNML document to the Web Service that performs the

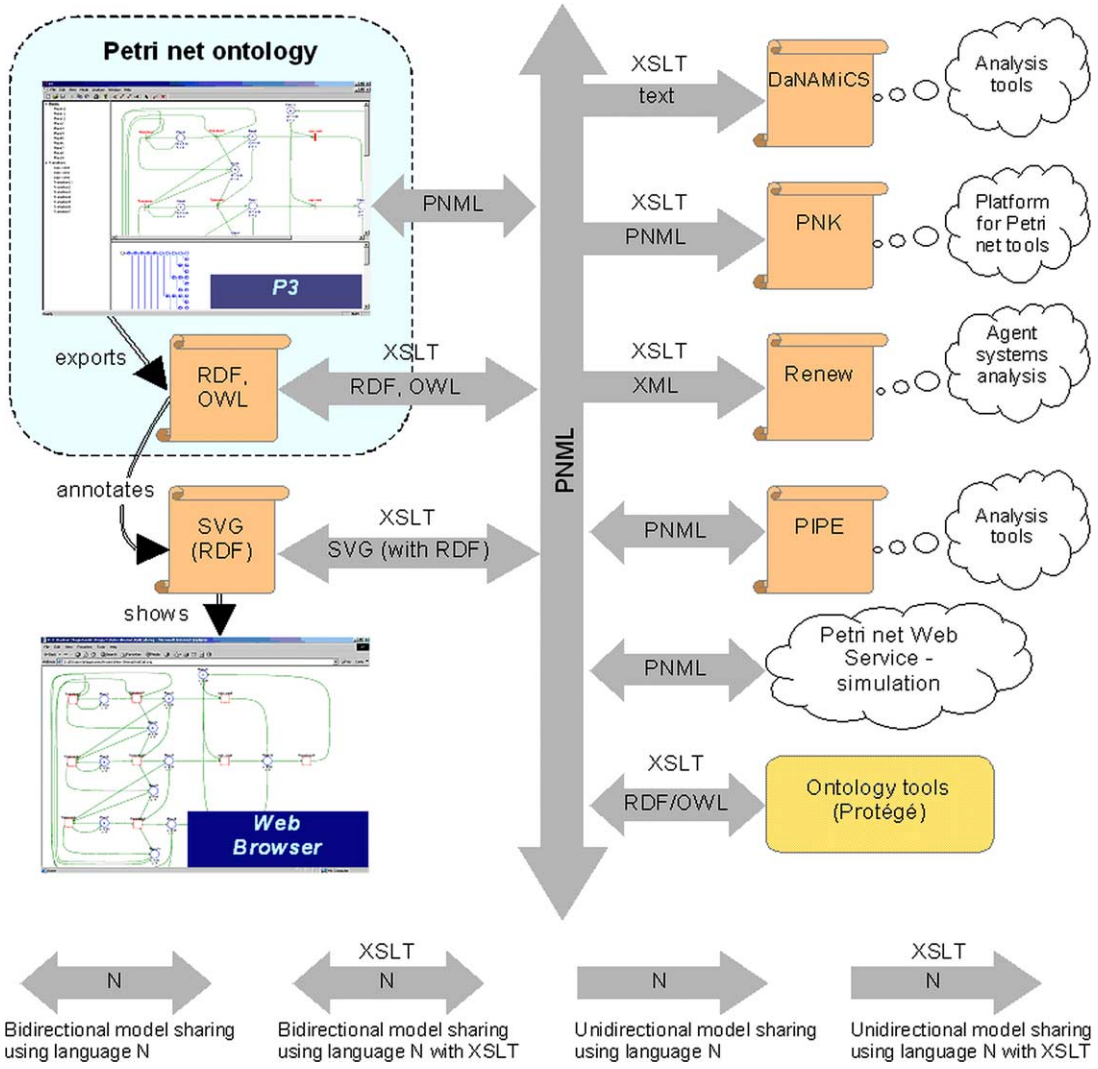

Fig. 12. Petri net infrastructure for the Semantic Web (that uses "PNML-based bus" for model sharing): the Petri net ontology, current Petri net tools, P3 tool, Web-based applications, Petri net Web Service, and ontology tools for validation of Petri net documents using the ontology.

simulation. When the simulation is over, the Web Service sends the new Petri net state (expressed in PNML) to the Web application. This PNML document is transformed to the RDF-annotated SVG document, and the Web application refreshes its view. In our on-going work, we apply this usage scenario in a Web-based learning system for teaching computer architecture and operating systems.

7. Conclusions

The main idea of this paper is that the Petri net ontology should provide the necessary Petri net infrastructure for the Semantic Web. The infrastructure understands Petri nets sharing using XML-based ontology languages (i.e., RDFS and OWL). The Petri net ontology and Semantic Web languages do not abandon PNML. On the contrary, we presented the "PNML-based bus" that takes advantage of PNML together with the Petri net ontology. That way, we can exploit potentials of current Petri net tools in the context of the Semantic Web. We also presented P3, the Petri net tool that creates ontology-based Petri net models. Its abbreviated version, its technical description, as well as a few developed XSLTs can be downloaded from http://www15.brinkster.com/p3net.

The paper gives guidelines for putting Petri nets on the Semantic Web. It also shows complementary features of the Petri net syntax and semantics by the example of PNML and the Petri net ontology. The example of RDF-based annotation of SVG documents indicates how to annotate other XML formats (e.g., Web Service Description Language – WSDL). This opens the door to incorporating "Petri net-driven intelligence" into Web-based applications (e.g., Web Service composition).

In the future, the P3 tool will support OWL-based annotation of SVG documents with Petri net graphs. Furthermore, we will use this annotation principle 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] K. Backlawski et al., Extending the Unified Modeling Language for ontology development, International Journal Software and Systems Modeling 1 (2) (2002) 142–156.
[2] F. Bause, P. Kemper, P. Kritzinger, Abstract Petri net notation, Petri Net Newsletter 49 (1995) 9–27.
[3] G. Berthelot, J. Vautherin, G. Vidal-Naquet, A syntax for the description of Petri nets, Petri Net Newsletter 29 (1988) 4–15.
[4] J. Billington 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, Eindhoven, 2003, pp. 483–505.
[5] C. Bock, UML without pictures, IEEE Software 20 (5) (2003) 33–35.
[6] E. Breton, J. Bézivin, Towards an understanding of model executability, in: Proceedings of the International Conference on Formal Ontology in Information Systems, IEEE Computer Society Press, Maine, USA, 2001, pp. 70–80.
[7] B. Chandrasekaran, J.R. Josephson, V.R. Benjamins, What are ontologies, and Why do we need them? IEEE Intelligent Systems 14 (1) (1999) 20–26.
[8] S. Cranefield, Networked Knowledge Representation and Exchange using UML and RDF, Journal of Digital information 1 (8) (2001) [Online]. Available from: <http://jodi.ecs.soton.ac.uk>.
[9] V. Devedžić, Understanding ontological engineering, Communications of the ACM 45 (4) (2002) 136–144.
[10] D. Djurić, D. Gašević, V. Devedžić, Ontology modeling and MDA, Journal on Object Technology 4 (1) (2005) 109–128.
[11] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.
[12] D. Gašević, V. Devedžić, Software support for teaching Petri nets: P3, in: Proceedings of the Third IEEE International Conference on Advanced Learning Technologies, IEEE Computer Society Press, Athens, Greece, 2003, pp. 300–301.
[13] D. Gašević, Petri net ontology, Ph.D. Thesis, FON – School of Business Administration, Belgrade, Serbia and Montenegro, 2004.
[14] D. Gašević, D. Djurić, V. Devedžić, V. Damjanović, Converting UML to OWL ontologies, in: Proceedings of the 13th International WWW Conference on Alternate track papers & posters, ACM Press, NY, USA, 2004, pp. 488–489.
[15] A. Gómez-Pérez, O. Corcho, Ontology languages for the Semantic Web, IEEE Intelligent Systems 17 (1) (2002) 54–60.
[16] T. Gruber, A translation approach to portable ontology specifications, Knowledge Acquisition 5 (2) (1993) 199–220.
[17] S. Handschuh, R. Volz, S. Staab, Annotation for the Deep Web, IEEE Intelligent Systems 18 (5) (2003) 42–48.
[18] K.M. Hansen, 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 Århus, Århus, Denmark [Online]. Available from: <http://www.daimi.au.dk/~marius/writings/cpn.pdf>, 2001.
[19] M. Havram, D. Gašević, V. Damjanović, A Component-based Approach to Petri Net Web Service Realization with Usage Case Study, in: Proceedings of the 10th Workshop Algorithms and Tools for Petri, Eichstätt, Germany, 2003, pp. 121–130.
[20] ISO/IEC/JTC1/SC7 WG19, New proposal for a standard on Petri net techniques [Online]. Available: <http://www.daimi.au.dk/PetriNets/standardisation/>, 2002.
[21] D. Jackson (Ed.), Scalable Vector Graphics (SVG) Specification v1.2, W3C Working Draft [Online]. Available: <http://www.w3.org/TR/2003/WD-SVG12-20030715/>, 2003.
[22] M. Jüngel, E. Kindler, M. Weber, The Petri Net Markup Language, in: Proceedings of the Seventh Workshop Algorithms and Tools for Petri, Universität Koblenz-Landau, Germany, 2000, pp. 47–52.
[23] E. Kindler, M. Weber, The Petri net kernel – an infrastructure for building Petri net tools, Software Tools for Technology Transfer 3 (4) (2001) 486–497.
[24] M. Klein, D. Fensel, F. van Harmelen, I. Horrocks, 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, Berlin, Germany, 2000.
[25] P. Kogut et al., UML for ontology development, The Knowledge Engineering Review 17 (1) (2002) 61–64.
[26] O. Kummer, F. Wienberg, The XML File Format of Renew, in: Meeting report at 21st International Conference on Application and Theory of Petri Nets, Århus, Denmark [Online] Available: <http://www.daimi.au.dk/pn2000/Interchange/-papers/det_04.ps.gz>, 2000.
[27] O. Kummer, F. Wienberg, Renew – the Reference Net Workshop – Tool Demonstrations, in: Proceedings of the 21st International Conference on Application and Theory of Petri Nets, Århus, Denmark, 2000, pp. 87–89.
[28] O. Kummer, R. Wienberg, M. Duvigneau, Renew - XML Format Guide [Online]. Available: <http://www.renew.de>, 2001.
[29] J. Miller, J. Mukerji, eds., OMG Document: omg/2003-05-01. MDA Guide Version 1.0 [Online]. Available: <http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf>, 2003.
[30] T. Murata, Petri nets: properties, analysis and applications, Proceedings of the IEEE 77 (4) (1989) 541–580.
[31] N.F. Noy, M. Sintek, S. Decker, M. Crubézy, R.W. Fergerson, M.A. Musen, Creating Semantic Web contents with Protégé-2000, IEEE Intelligent Systems 16 (2) (2001) 60–71.
[32] OMG Document formal/02-04-03, Meta Object Facility (MOF) Specification v1.4 [Online]. Available: <http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf>, 2001.
[33] OMG document formal/03-03-01. OMG Unified Modeling Language Specification v1.5 [Online]. Available: <http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.zip>, 2003.
[34] M. Peleg, I. Yeh, R. Altman, Modeling biological processes using Workflow and Petri net models, Bioinformatics 18 (6) (2002) 825–837.
[35] J. Peterson, Petri Net Theory and the Modeling of Systems, Prentice Hall, Englewood Cliffs, New Jersey, 1981.
[36] J. Robie, The Syntactic Web – Syntax and Semantics on the Web, Markup languages: Theory and Practice, MIT Press, Cambridge, MA, 2001, 411–440.
[37] A. Semenov, A.M. Koelmans, L. Lloyd, A. Yakovlev, Designing an asynchronous processor using Petri nets, IEEE Micro 17 (2) (1997) 54–64.
[38] P. Štrbac, An Approach to Modeling Communication Protocol by Using Upgraded Petri Nets, Ph.D. Thesis, Department of Computer Engineering and Informatics, Military academy, Belgrade, Serbia and Montenegro, 2002.
[39] Y. Sure, R. Studer, A methodology for ontology-based knowledge management, in: J. Davies, D. Fensel, F. van Harmelen (Eds.), Towards the Semantic Web – Ontology-driven knowledge management, John Wiley & Sons, 2003, pp. 33–46.
[40] S. Bechhofer, et al., OWL Web Ontology Language Reference, W3C Recommendation. <http://www.w3.org/TR/2004/REC-owl-ref-20040210>, 2004.
[41] M. Weber, E. Kindler, The Petri net kernel, in: Petri Net Technology for Communication Based Systems, in: H. Ehrig, W. Reisig, G. Rozenberg, H. Weber (Eds.), Lecture Notes in Computer Science, vol. 2472, Springer, Berlin, 2003, pp. 109–124.