ACTA UNIVERSITATIS AGRICULTURAE ET SILVICULTURAE MENDELIANAE BRUNENSIS
SBORNÍK MENDELOVY ZEMĚDĚLSKÉ A LESNICKÉ UNIVERZITY V BRNĚ

Ročník LII 4 Číslo 6, 2004

XML AS A FORMAT OF EXPRESSION OF OBJECT-ORIENTED PETRI NETS

P. Jedlička

Received: June 30, 2004

Abstract

JEDLIČKA, P.: *XML as a format of expression of Object-Oriented Petri Nets*. Acta univ. agric. et silvic. Mendel. Brun., 2004, LII, No. 6, pp. 45-54

A number of object-oriented (OO) variants have so far been devised for Petri Nets (PN). However, none of these variants has ever been described using an open, independent format – such as XML. This article suggests several possibilities and advantages of such a description. The outlined XML language definition for the description of object-oriented Petri Nets (OOPN) is based on XMI (description of UML object-oriented models), SOX (simple description of general OO systems) and PNML (an XML-based language used for the description of structured and modular PN). For OOPN, the XML form of description represents a standard format for storing as well as for transfer between various OOPN-processing (analysis, simulation, ...) tools.

Petri Nets, Object-Orientation, XML, portability

Petri Nets (PN) represent a popular formalism for discrete parallel systems modelling. The simple principle upon which they are based allows comprehensible graphic representation, simulation and analysis. An analysis of systems modelled using PN makes it possible to reveal a number of their characteristic features already in the early phases of the design, and thus to avoid problems that could arise later.

However, primitive PN variants, such as Condition-Event PN or Place-Transition PN (Peterson, 1981), are practically unusable for modelling more complex systems. This is why PN, during the many years of their development, were adapted to the needs and character of the modelled systems. Several variants of High-Level Petri Nets (HLPN) (Jensen, Rozenberg, 1991) were devised. In the first variant, referred to as "predicate Petri Nets", tokens represent particular data, which can be tested and modified by the net transitions. Coloured PN (Jensen, 1991) represent another variant; they made it possible to bind net tokens with datatype values. Hierarchical PN enabled modelling large systems; their characteristic features make them similar to structured programming. One of the latest trends in PN development is object orientation. Object-oriented Petri Nets (OOPN) (Janoušek, 1998; Martiník, 1999) reflect the current trend towards using an object-oriented approach in designing and implementing (not only) software systems. This is why OOPN are often described using a particular object-oriented programming language through which the modelled system is implemented. However, there are programming languages created specifically and exclusively for simulating OOPN. An example is the PNtalk language (Janoušek, 1995), which is based on Smalltalk. Other possibilities include graphic representation of PN, and description of PN using mathematical methods. Only one of the solutions mentioned above is practical enough to be used for processing OOPN (e.g. simulation) by means of computer technology: the use of an OO programming language. However, neither this solution is suitable for a general representation of OOPN. General and platform-independent representation of OOPN requires the use of an open and standardized format, the search for which is the aim of this article.

MATERIAL AND METHODS

Object model notation

The object-oriented (OO) approach to the analysis and design of information systems (IS) has gone through many years of development, and its methodology is quite sophisticated already. However, the *notation* of the OO model took a number of different ways of development, and it had not been until recently that consensus was reached in this area. During the 1990s there appeared several different OO methodologies that were coupled with their own notation sets. The three most popular methods were OMT (Rumbaugh), OOAD (Booch) and OOSE (Jacobson). Each of these methods had its own purpose and focus. OMT laid stress on analysis, OOAD on design and OOSE on the analysis of OO system behaviour. Around 1995, the UML language (OMG: UML…, 2004) was created by unifying the Booch and Rumbaugh notations and object-oriented notations devised by other methodologists. It is a visual modelling language for the creation and interchange of OO models. It is independent of any particular programming language or process of development and analysis. For the sake of an independent and open description of the UML model, an XML-based language called XMI (XML Metadata Interchange) (OMG: XML…, 2004) was created. It is maintained by the OMG group[1]. At present, XMI is the basic *de facto* standard for exchanging UML models between various modelling tools and, at the same time, it is one of the means of their persistent representation. The basic UML components are represented in XMI in the following way:

- Each metamodel of a **class** is decomposed into three parts: *properties*, *associations* and *composition*. Entities are declared for each metamodel in such a way that the name begins with the name of the class and ends with "Properties" for properties, "Associations" for associations, and "Compositions" for the composition of the class. The properties entity contains a list of XML elements that correspond with metamodel attributes; the associations entity contains XML elements that represent the roles of associated entities; and the composition contains XML elements that represent the roles of an association in the aggregation.
- Currently, the **inheritance** mechanism in XML is not completely resolved. This is why XMI specifies inheritance as "copy-down inheritance". It means that the child will copy the attributes and composition of the parent.
- In order to represent the metamodel **attributes**, XML elements and attributes are used. If the attribute in the metamodel is of the list or the primitive type, it is represented in XMI as an XML attribute. In other cases a separate element `<XMI.field>` is used.
- Each **association**[2] is represented by an XML element and XML attributes. The element bears the name of the association, and entities taking part in the association are listed as an attribute (therefore, n-ary associations can be displayed, too).

Interchange format for Petri Nets

As it was already mentioned at the beginning of this article, the most common ways of describing PN are: graphic representation, a mathematical model, and a program in an OO programming language. However, for transfer between applications that process PN models, and also for permanent storage of PN in an electronic form, the use of a different format is necessary. This format should be open, platform-independent and, if possible, human-readable. XML is such a format, and it is not a coincidence that some have already tried to use it for the description of Petri Nets. What seems to be the most advanced solution is the PNML language (Petri Net Markup Language), created by the Department of Computer Science at Humboldt-Universität in Berlin (Weber, 2004).

PNML

The PNML language is defined using the progressive meta-language RELAX NG. The basic version (*basic PNML*) makes it possible to describe standard P/T Petri Nets; *structured PNML* extends the language's expressive capabilities to cover the description of structured Petri Nets, and *modular PNML* allows describing the modularity of PN. A module is an independent net with an interface through which it can be interconnected with other nets and modules. Apart from the logical structure of the net, PNML describes also the physical layout of the PN graph components. It uses the Cartesian coordinate system.

The definition of PNML is general enough to be used for various PN variants without limitations. This is why it can be utilized as the basis of an OOPN description language.

[1] Object Management Group, `http://www.omg.org`
[2] An association expresses the fact that objects are in a mutual relation. Associations between two objects can be unidirectional or bidirectional.

Choosing a language suitable for the description of XML schemas

An XML-based language can be defined in a number of ways. The oldest language used for the description of XML document structure is DTD (*Document Type Definition*) (W3C: Extensible…, 2004), which has its origins in the SGML language. Today, the use of DTD has probably one advantage only – the widest application support. Disadvantages are, on the other hand, numerous: DTD does not support namespace, does not allow defining element and attribute datatypes, and uses a syntax different from everything that is used in the world of XML. All these drawbacks are overcome in the *XML Schema* language (*W3C XML Schema – WXS*) (W3C: XML Schema, 2004), adopted in May 2001 as a W3C recommendation. Table I shows some of the properties supported in the WXS language. The comparison includes also the RELAX NG (Clark, 2003) language, which was created by experts in reaction to the complexity and extensiveness of WXS. The comparison makes it clear that WXS has the widest range of features. This is one of the reasons why it will be used for the sake of defining an OOPN description language. Another argument in favour of WXS is the fact that XML is built on object-oriented principles, and only WXS can put a number of these principles in use. For the purpose of designing schemas there are a number of features available that are very similar to features known from object-oriented languages. The possibility to derive new types from the existing ones is nothing else than inheritance. This derivation can be used for simple as well as for complex types. Other OO-inspired features include substitution groups, abstract datatypes, and the possibility to block further inheritance from a certain datatype. Another argument supporting the choice of WXS is that it is well supported by both commercial and non-commercial software.

I: *Capabilities of languages as regards the description of XML document schemas*

Supported feature	DTD	XML Schema	Relax NG
XML format	no	yes	yes
PSVI[3]	yes	yes	no
own datatypes	no	yes	no
namespaces	no	yes	yes
referential integrity	yes	yes	no

As it has already been mentioned, when defining an XML-based language for the description of OOPN we will draw upon PNML. In particular, *structured PNML* seems to be the most suitable initial form. It is because object-oriented PN are also based on structured PN. The main difference between the two types of PN is that structured PN have a static structure while in OOPN the individual subnets can be dynamically created and deleted because they represent the system's class instances (objects) that are created or deleted. We will also make use of the tried-and-tested approaches of XMI, which is specifically designed for object-oriented systems modelling. Another language that cannot go unmentioned is *SOX* (*Schema for Object-oriented XML*) (W3C: Schema…, 2004). SOX allows using inheritance and polymorphism in schemas. The individual parts of schemas can be shared and reused. Despite being oriented specifically towards the description of object models, SOX is not suitable for our purposes due to its over-simplicity. This, however, does not mean that we will not use some of its principles.

RESULTS

This part of the article focuses on the components and principles of OOPN, for which a suitable form of XML representation has to be chosen. There are many OOPN conceptions in the world, of which at least two originate in the Czech Republic. According to available sources, no XML-based language has been defined for the description of OOPN until now. Therefore, the following steps will be chosen so that the expressive capabilities of the proposed language allow using it for various OOPN specifications.

The basic conception of PN is quite simple. A PN is composed of two types of node – places and transitions – and of directed arcs. Arcs interconnect places and transitions. Places can contain tokens, which can be distinguishable according to the type of PN and whose number differs depending on transition events. This structure and semantics is common to all PN types and levels. There are differences especially as regards the understanding of places and tokens, the character of transition functions, and the structure and modularity of the whole net. Taking into account the limited scope of this article, it is not possible to describe the entire PN development branch, from the basic types to OOPN. Therefore, we will focus solely on OOPN, and in particular on their features that have not been described by the *structured PNML language*.

[3] Post Schema Validation Infoset, a typed document that results from assigning datatypes based on validation against a schema. It is used, for example, in query languages (XQuery), which need to know the type of data in the individual elements and attributes.

OOPN structure and components

Just like any other object-oriented system, OOPN works with classes and their instances – *objects*. Both classes and objects consist of data items (*attributes*) and *methods*. All classes in a system exist throughout its existence while objects are dynamically created or deleted. As OOPN applies *inheritance*, some classes are descendants of other classes. Further, there are *abstract classes* that can be inherited from but it is not possible to instantiate objects. *Final classes*, on the other hand, are meant to be instantiated, and cannot be used for inheritance. Other supported properties are encapsulation and polymorphism.

A class and an *object* in OOPN are represented by a separate page of the net. *Attributes* are represented by tokens. Each token is assigned:

- an identifier;
- a datatype – either primitive, or an object reference;
- visibility – one of the set {public, protected, private};
- value.

A *place* is a node in the net; it contains tokens and is connected with transitions by input and output arcs. Apart from tokens, a place is assigned also the initial marking, i.e. the initial values of tokens contained in the place.

Each *method* of a class or object is represented by a separate subpage of a class or object page. The main properties of a method's subpage include:

- an identifier;
- *a visibility attribute* – one of the set {public, protected, private};
- *type attribute* – one of the set {normal, virtual, constructor, destructor};
- *input and output interface* constituted by an input and an output place.

An *arc* is described by an arc function consisting of an organized set of expressions. All expressions of the input arc of the transition are of the `boolean` type; expressions of the output arc of the transition correspond (in number and type) to the output place. Some of the arcs are inhibitory.

The function of a *transition* in OOPN is to perform an atomic operation, to create a new object, or to invoke a method of an existing object.

Expressing OOPN components and principles in XML

We will define our OOPN-describing XML language *from the bottom up*, i.e. start with the basic OOPN elements from which we will, step by step, compose the entire net.

A **token** is an elementary component of PN. Tokens are located in places, where they are manipulated via transitions. In addition to their value, tokens have some other properties; this is why they will be represented through elements in XML. For the purpose of describing a token's identifier, visibility and datatype, element attributes will suffice. The token value can be expressed by element content. Therefore, the token as a whole can be described in WXS in the following way:

```
<element name="token">
<complexType>
     <element name="value" type="anySimpleType"/>
          <attribute name="id" type="ID" use="required"/>
          <attribute name="visibility" type="visibilityType" use="required"/>
          <attribute name="type" type="string" use="required"/>
     </complexType>
</element>
```

The following is a definition of the `visibilityType` datatype:

```
<simpleType name="visibilityType">
     <restriction base="string">
          <enumeration value="public"/>
          <enumeration value="protected"/>
          <enumeration value="private"/>
     </restriction>
</simpleType>
```

The other types used (`anySimpleType`, `ID` and `string`) are built-in (predefined) types of the XML language.

A **place**, being a container of tokens with initial marking, can be defined as follows:

```
<element name="place">
    <complexType>
          <element ref="token" maxOccurs="unbounded"/>
          <attribute name="id" type="ID" use="required"/>
    </complexType>
</element>
```

An example of describing a place using the above-mentioned WXS definitions:

```
<place id="p1">
    <token id="jméno" visibility="protected" type="string">
          <value>Karel</value>
    </token>
    <token id="věk" visibility="private" type="unsignedByte">
          <value>28</value>
    </token>
</place>
```

For the purpose of defining a **transition** the following structure can be used:

```
<element name="transition">
    <complexType>
          <choice>
             <element name="function" type="string"/>
             <element name="new" type="IDREF"/>
             <element name="call" type="IDREF"/>
          </choice>
          <attribute name="id" type="ID" use="required"/>
    </complexType>
</element>
```

For the description of an **arc** we could use the following definition:

```
<element name="arc">
    <complexType>
          <element name="expr" type="string" maxOccurs="unbounded"/>
          <attribute name="id" type="ID" use="required"/>
          <attribute name="type" type="arcType" use="required"/>
          <attribute name="inhib" type="boolean" use="required"/>
          <attribute name="source" type="IDREF" use="required"/>
          <attribute name="target" type="IDREF" use="required"/>
    </complexType>
</element>

<simpleType name="arcType">
    <restriction base="string">
          <enumeration value="pt"/>
          <enumeration value="tp"/>
    </restriction>
</simpleType>
```

Now that we have described all the elementary components of OOPN we can proceed to define the OOPN **method** (of class or object), as defined by Martiník (1999). The method, in OOPN expression, consists of following elements: identifier (ID), visibility attribute, type attribute (normal, virtual, constructor, destructor), interface (one input and one output place), inner places (optional – for eventual artificial variables) and transitions. Since XML doesn't support inheritance of elements, we have to define input and output places of the method (i.e. its interface) repeatedly, according to the definition of common places.

```
<element name="method">
    <complexType>
        <sequence>
            <element name="interface">
                <complexType>
                    <sequence>
                        <element name="inputPlace">
                            <complexType>
                                <element ref="token" maxOccurs="unbounded"/>
                                <attribute name="id" type="ID" use="required"/>
                            </complexType>
                        </element>
                        <element name="outputPlace">
                            <complexType>
                                <element ref="token" maxOccurs="unbounded"/>
                                <attribute name="id" type="ID" use="required"/>
                            </complexType>
                        </element>
                    </sequence>
                </complexType>
            </element>
            <element ref="place" minOccurs="0" maxOccurs="unbounded"/>
            <element ref="arc" minOccurs="1" maxOccurs="unbounded"/>
            <element ref="transition" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="id" type="ID" use="required"/>
        <attribute name="visibility" type="visibilityType" use="required"/>
        <attribute name="type" type="methodType" use="required"/>
    </complexType>
</element>

<simpleType name="methodType">
    <restriction base="string">
        <enumeration value="normal"/>
        <enumeration value="virtual"/>
        <enumeration value="constructor"/>
        <enumeration value="destructor"/>
    </restriction>
</simpleType>
```

The following is a definition of an **object**. Object contains an identifier (ID), methods (transitions, min. 1, max. unbounded), arcs and attributes (1 place).

```
<element name="object">
    <complexType>
        <sequence>
            <element ref="transition" maxOccurs="unbounded"/>
            <element ref="arc" minOccurs="2" maxOccurs="unbounded"/>
            <element ref="place"/>
        </sequence>
        <attribute name="id" type="ID" use="required"/>
    </complexType>
</element>
```

At the end we show a XML definition of a **class**. Class consists of an identifier (ID), visibility attribute, type attribute (normal, abstract, final), reference to the superclass (optional), methods (transitions, min. 1, max. unbounded), arcs and attributes (1 place).

```
<element name="class">
    <complexType>
        <sequence>
            <element ref="transition" maxOccurs="unbounded"/>
            <element ref="arc" minOccurs="2" maxOccurs="unbounded"/>
            <element ref="place"/>
        </sequence>
        <attribute name="id" type="ID" use="required"/>
        <attribute name="visibility" type="visibilityType" use="required"/>
        <attribute name="type" type="classType" use="required"/>
        <attribute name="superclass" type="IDREF"/>
    </complexType>
</element>

<simpleType name="classType">
    <restriction base="string">
        <enumeration value="normal"/>
        <enumeration value="abstract"/>
        <enumeration value="final"/>
    </restriction>
</simpleType>
```

DISCUSSION

The existence of various Petri Net types results from efforts to increase their modelling and description capabilities for a certain type of system that is studied. Taking into account the popularity of the object-oriented approach, especially in the sphere of software design, it is not surprising that a number of *object-oriented* Petri Net design proposals have been made recently. However, these attempts have only got as far as to describe OOPN using mathematical methods or a programming language. There is a need for a specialized yet open and independent form of OOPN description, which would be suitable for persistent storage and transfer of OOPN models between various applications.

This article shows the possibilities of creating a custom-made XML-based language describing OOPN. The proposed solution draws upon certain principles of XMI, SOX and PNML; however, with regard to the given purpose it is the author's own solution. Due to the limited scope it was not possible to mention all the possibilities the XML language offers for OOPN modelling. We have merely suggested how to describe the individual Petri Net components (tokens, places, arcs, transitions) and the entire subnets (classes, objects, methods). The solution proposal nevertheless covers all traditional properties of object-oriented systems, i.e. inheritance, encapsulation and polymorphism. Further development of the language is expected to bring, in particular, refinement of its expressive capabilities in the sphere of OOPN semantics. The new capabilities could include:

- using the `unique` element and the XPath query language, uniqueness of values can be ensured, e.g. for object identifiers within the entire net, or for tokens within a place or a page of the net;
- by restricting the built-in datatypes or by means of regular expressions, element and attribute datatypes can be restricted;
- using XML's implicit datatypes `ID` and `IDREF`, referential integrity can be defined.

The above-mentioned XML properties clearly suggest that the proposed solution is suitable for the description of object-oriented Petri Nets. The author of the article will, therefore, make efforts to further develop the language.

SUMMARY

The aim of the article was to evaluate the possibilities of using XML as a basis for creating an open format for the description of object-oriented Petri Nets (OOPN). We have suggested how to express the basic OOPN components as well as how to implement the three cornerstones of the object-oriented approach – inheritance, encapsulation and polymorphism. Certain partial principles have been derived from existing languages (XMI, SOX, PNML), others had to be newly modelled using XML features. Our efforts are focused on the creation of a language that would be, to the greatest extent possible, able to describe the existing OOPN conceptions; at the same time, the language should be backwards compatible, i.e. capable of describing lower-level Petri Nets.

SOUHRN

XML jako formát vyjádření objektových Petriho sítí

Petriho sítě se od doby svého vzniku dočkaly mnoha variant a rozšíření. Nevyhnul se jim ani v současnosti moderní objektově orientovaný (OO) přístup. Dosud vytvořené koncepce objektových Petriho sítí (Object-Oriented Petri Nets – OOPN) však zatím nedisponují otevřeným a nezávislým formátem vyjádření, jaký poskytuje například jazyk XML. Ukazuje se, že pomocí XML je možné popsat nejen základní komponenty OOPN, ale lze také postihnout všechny tři pilíře objektově orientovaného přístupu – dědičnost, zapouzdřenost a polymorfismus. Nastíněná definice XML jazyka pro popis OOPN stojí zčásti na základech jazyků XMI (popis objektových modelů UML), SOX (jednoduchý popis obecných OO systémů) a PNML (jazyk na bázi XML pro popis predikátových, strukturovaných a modulárních Petriho sítí), jako speciální XML jazyk pro vyjádření OOPN je však podle dostupných zdrojů zatím pokusem prvním a jediným.
XML forma vyjádření představuje pro OOPN standardní formát pro uložení i převod mezi nástroji pro jejich zpracování (analýzu, simulaci, …). Jedná se o formát obecně rozšířený, platformově nezávislý, snadno automatizovaně zpracovatelný a (v případě potřeby) čitelný pro člověka. Veškerá zde popsaná snaha směřuje k vytvoření XML jazyka, který by byl co nejvyšší měrou schopen postihnout různé existující koncepce OOPN, současně by měl být zpětně kompatibilní, tedy schopný popsat Petriho sítě nižších úrovní.

Petriho sítě, objektová orientace, XML, přenositelnost

REFERENCES

CLARK, J.: RELAX NG home page. [online] Ver. 2.0 Last upd. 2003-09-24. [cit. 2004-06-26]. URL: <http://www.relaxng.org/>.

JANOUŠEK, V.: Modelování objektů Petriho sítěmi. 1. vyd. Brno: VUT, 1998. 137 p. Doctoral thesis.

JANOUŠEK, P.: PNtalk: Object Orientation in Petri Nets. In Proceedings of European Simulation Multiconference '95, pages 196–200. Prague: Czech Technical University, 1995. ISBN 1-56555-080-3

JENSEN, K.: Coloured Petri nets : basic concepts, analysis methods and practical use. 1st ed. Berlin; New York; Paris: Springer, 1991. 234 p. ISBN 3-540-55597-8.

JENSEN, K., ROZENBERG, G.: High-level Petri nets: theory and application. 1st ed. Berlin; New York; Paris: Springer, 1991. 724 p. ISBN 3-540-54125-X.

MARTINÍK, I.: Metodologie tvorby objektově-orientovaných programových systémů s využitím teorie objektových Petriho sítí. 1. vyd. Ostrava: VŠB-TU, 1999. 218 p. Doctoral thesis.

OMG – Object Management Group XML Metadata Interchange (XMI) Specification. [online] Ver. 2.0 ©1997–2004. [cit. 2004-06-25]. URL: <http://www.omg.org/docs/formal/03-05-02.pdf>.

OMG – Object Management Group UML™ Resource Page [online]. ©1997–2004. [cit. 2004-06-25]. URL: <http://www.uml.org/>.

PETERSON, J. L.: Petri Net Theory and the Modeling of Systems. 1st ed. Upper Saddle River, NJ, USA: Prentice Hall, 1981. 290 p. ISBN 0136619835.

W3C Extensible Markup Language (XML) 1.0 (Third Edition). [online]. ©1994–2004. [cit. 2004-06-26]. URL: <http://www.w3.org/TR/REC-xml/>.

W3C Schema for Object-Oriented XML 2.0. [online] Ver. 2.0 ©1994–2004. [cit. 2004-06-26]. URL: <http://www.w3.org/TR/NOTE-SOX/>.

W3C XML Schema. [online]. ©1994–2004. [cit.

2004-06-26]. URL: <http://www.w3.org/XML/Schema>.

WEBER, M.: Petri Net Markup Language [online] [cit. 2004-06-26]. URL: <http://www.informatik.hu-berlin.de/top/pnml/about.html>.

Address

Ing. Petr Jedlička, Ústav informatiky, Mendelova zemědělská a lesnická univerzita v Brně, Zemědělská 1, 613 00 Brno, Česká republika