Structure of Abstract Syntax trees for Colored Nets in PNML

F. Kordon & L. Petrucci
Fabrice.Kordon@lip6.fr
Laure.Petrucci@lipn.univ-paris13.fr

version 0.2 (draft)

June 26, 2004

Abstract

Formalising the description of a Coloured Petri Net eases the implementation of tools based on such a representation. We propose in this document a structure to represent of places, arcs, and transitions by means of abstract syntax trees to be used in the standardised representation for Petri nets to be proposed as ISO-IEC 15909 – part 2.

This document will only deal with information that is not supported in Petri nets. Hence, information such as the name of a place or a transition as well as graphical information are neither mentioned nor described. We focus on the extensions required for handling high-level features only.

This document mainly focuses on the abstract syntax trees structure. The translation into XML should be performed when an agreement on this proposal is reached.

Contents

1 Introduction

1 This document is a proposal contributing to part 2 of the ISO 15909 standard. It is based on [1].

2 We propose to base the Colored Petri net standard on the formal definition of Well Formed Petri Nets, proposed in [2] (called WFPN in the rest of this document). The incentive for that choice is that WFPN correspond to a basic type of high level Petri nets for which the theory is sufficiently strong and are equipped with a sufficient number of high level features. WFPN are supported by several tools [4, 3] and a stochastic extension allows for performance evaluation. This last point is of interest since the standard has to consider several extensions that should be compatible.

3 Well Formed Petri Nets are a high-level net model that includes, besides the graphical features of a Petri net - places, transitions, arcs and their names - more complex annotations:

- markings and tokens,
- types and initial markings of places,
- transition conditions,
- arc terms,

4 All these require an enriched syntax.

5 The additional information we previously enumerated is used for the definition of the semantics of a Well Formed Petri Nets, which is briefly and informally described in the next subsections.

1.1 Behavior of a Well Formed Petri Nets

6 The behavior of a Well Formed Petri Net is controlled by the same set of rules used for general colored nets:

- A type is associated with each place and transition of the model. Elements of these domains are called colors.
- Each token in a place is colored by an element of the place type (several tokens may have the same color). The marking of a place is thus a multiset of colors - a set in which an element may occur several times.
- The enabling and firing rule comply with clause 7 of [1]. For a transition mode to occur, each input place of the transition must contain a sufficient (possibly null) number of tokens for every color of the place domain. These tokens will be taken from the place when the transition fires. Similarly, the firing will produce colored tokens in the output places of the transition. As in Petri nets, the arcs annotations determine the number of tokens to be taken or produced. However, this annotation is now a function associating a multiset of colors of the place type with each mode of the transition.
- The mode used to enable a transition must satisfy some predicate called transition condition (clause 7.4.1 in [1]).

1.2 Declarative part of a Well Formed Petri Nets

7 The description of a Well Formed Petri Nets net is composed of two parts: the first part contains the declaration of the Petri net (places, transitions and arcs). This declaration is included in the present model and takes place in the first part of the PNML standard. We describe in the second part, all the high-level features of the model for which we give the syntactic construction rules.

8 These high-level annotations are listed in sections 1.4 to 1.6. However, their description, i.e., the description of types, markings, transition conditions must refer to the type definitions of the model.

9 These descriptions are done in different extensions of the standard that are all optional :

- The class declaration section defines object classes. The objects are the elementary entities (sites, memories, etc.) that appear in the description of the model;
- The domain declaration section defines the set of color domains associated with the different places and transitions of the net. A color associates one or several elements in a tuple;
- The variable declaration section defines the name and the domain to which variables used for the valuation of the arcs belong.
- The macro declaration allows parameterization of expressions.

1.3 Predefined color functions

10 The only predefined color functions are:

- $++n(x)$ that increments a variable x of value n. This is a circular increment as defined in [2].
- $--n(x)$ that decrements a variable x of value n. This is a circular decrement as defined in [2].
- $D.all$ that associates, for color type D, a token for each possible value of the color type[1].
- $< x_1,...,x_n >$ that aggregates several tokens into a structured one. Implicitly, the extraction function is defined as the one that allows extraction of a component in a structured token.

11 Boolean functions are also available to compare tokens:

- $=$ and $\neq$ allow to compare two tokens having the same color type.
- $<$, $>$, $\geq$ and $\leq$ allow sorted comparison for class tokens only. As in programming languages, there are no predefined operators for these comparisons on composed types. Such operators may be defined in extensions.

12 Finally boolean functions can be used to combine comparison functions on tokens:

[1] all is usually named: "broadcast function".

- *AND* corresponds to the boolean product.
- *OR* corresponds to the boolean addition.
- *NOT* corresponds to the negation operator.

13 The priority between these boolean operators is the one of ”traditional” languages, *NOT* is evaluated prior to *AND* which is evaluated prior to *OR*.

1.4 Place Type and Marking

14 Since token present in places are colored, a place must be associated with a color type. Types may be basic (e.g. integer or enumeration) or built upon basic types using e.g. cartesian product (see aggregation function is section 1.3). A place without type is assumed to contain untyped tokens (as in Petri nets).

15 Thus, a label contains the name of the color type for tokens stored in the corresponding place. This label should be an identifier.

16 The initial marking is part of the information associated with the place. A marking is described as a set of tokens that may have a multiplicity greater than 1. It is possible to use the *broadcast function* to define the initial marking of a place in a compact manner (see section 1.3).

17 If the domain of the place is *null*, the place is considered as a P/T one and the only possible labels are positive integers.

1.5 Arc terms

18 The terms are built from predefined operations and variables (as listed in 1.3, paragraph 10). If not defined, it is assumed that the arc takes/produces one untyped token from/to the connected place.

1.6 Transition Conditions

19 The aim of a guard is to restrict the possible bindings of a transition by adding constraints on the variables. The guards are not defined in a particular section. They only appear in the information associated with a transition.

20 The conditions included in the definition of WFPN can compare two terms, possibly with variables. Both terms must belong to the same type. The comparison operators are those of section 1.3, paragraph 11.

21 Conditions can be combined using usual boolean functions *AND*, *OR* and *NOT* (see section 1.3, paragraph 12). It is also possible to add parentheses to locate sub-expressions.

2 Proposed Structure for Abstract Syntax Trees

22 This section details all the possible abstract syntax trees (AST) identified to de-

scribe unambiguously a WFPN in the PNML format. We focus on the tree structure that could be easily expressed using XML.

23 We outline the following four types of AST for:

- global declaration (i.e. global to the whole net[2]),
- places,
- arcs,
- transitions.

2.1 Global declarations

2.1.1 Macro definitions

RDB: these macros should help to define constants to enable parameterization of the model. Basically, the idea is to replace a macro by a string (representing a number, a label, etc.) directly in the abstract syntax trees of the description.

2.1.2 Basic Types

24 The Abstract syntax tree associated with a basic type should respect the structure provided in figure 1. The root node is labeled by the keyword **BASICTYPE** and has two subtrees:

- the first one corresponds to a string node and is the name of the basic type,
- the second one describes the type content and refers to the structure presented in Figure 2.

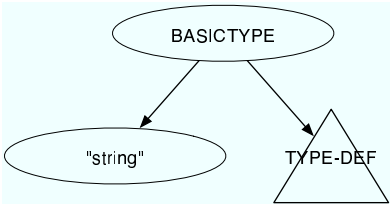

Figure 1: Structure of an AST describing a basic type.

25 A basic type may be an integer class (with possible bounds) or an enumeration type. According to these possibilities, the definition of the type content has to respect one of the two structures presented in Figure 2.

26 The left subtree of Figure 2 corresponds to an integer-type content. The root is labeled by the keyword **INTEGER**. If it has no subtree, then it corresponds to the default integer class supported by the standard. Otherwise, it contains two nodes that have integer values. The first one contains the lower bound and the second one the upper bound.

[2]There should be some way to factor out declarations when considering hierarchically described Petri Nets.

27 The right subtree of Figure 2 corresponds to an enumerate type content. The root is labeled by the keyword **ENUM**. The subtrees list all the possible values in the type ; these nodes labels are strings corresponding to one enumeration value. Values are totally ordered based on the order they appear in the abstract syntax tree: the first subtree (leftmost) corresponds to the ”lowest” value of the type and the last one (rightmost) to the ”highest”.

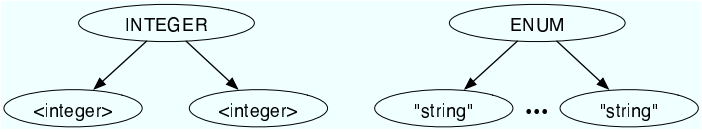

Figure 2: Structure of an AST defining the content of a basic type.

2.1.3 Type Definition

28 A type is a cartesian product of types (basic or not).

29 Its structure should conform to the tree in figure 3. The root node is labeled by the keyword **TYPE**. The left subtree is labeled by a string providing the name of the color type. The right subtree root is labeled by the keyword **PRODUCT**. Under this node, there are as many nodes as involved color type. The s^{th} subtree is labeled by the identifier of the color type of the s^{th} component.

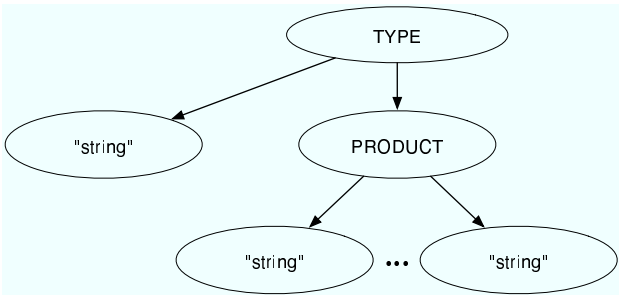

Figure 3: Structure of an AST describing a color type.

2.1.4 Variable Declaration

30 A variable may contain any value of a given color type.

31 It should be declared and associated with the corresponding color type. The corresponding abstract syntax tree must respect the structure presented in figure 4. The root node is labeled by the keyword **VAR**. The first subtree (leftmost) is labeled by a string that identifies the variable name and the second subtree is labeled by the name of the type.

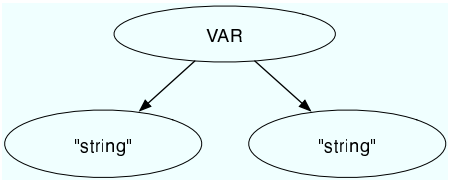

Figure 4: Structure of an AST describing a variable declaration.

2.1.5 Global Declarations

32 Global declarations gather all types and variables declared in a Petri net[3].

33 The global declaration abstract syntax tree appears only once in the model[4] and is structured as shown in figure 5. The root node is labeled by the keyword **DECLARATIONS** and has three subtrees:

- the first one has a root labeled by the keyword **BASICTYPES**. Below this node, there is one subtree per basic type. These subtrees respect the structure defined in section 2.1.2.
- the second one has a root labeled by the keyword **TYPES**. Below this node, there is one subtree per type. These subtrees respect the structure defined in section 2.1.3.
- the last one has a root labeled by the keyword **VARS**. Below this node, there is one subtree per variable. These subtrees respect the structure defined in section 2.1.4. If there are no variable declared in the model, the **VARS** node has no subtree.

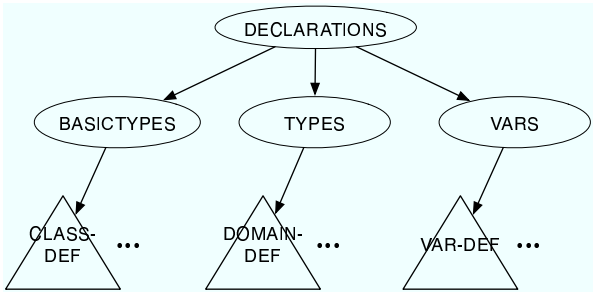

Figure 5: Structure of an AST describing global declarations.

2.2 Expressions

34 This section defines abstracts syntax trees that correspond to expressions that can be used to represent the initial marking of places, arcs terms or transitions conditions.

[3] or in a sub-Petri net model if hierarchical nets are considered. Visibility rules should be defined then.
[4] or in the Petri net component.

35 There are two types of expressions:

- general expressions that define one or more tokens.
- boolean expressions that can refer to color tokens or color variables but involve boolean operators only,

2.2.1 General expressions

36 Figure 6 shows the general structure of a general expression tree, which is recursive. The root of a general expression abstract tree is labeled by the keyword **EXPR**. The first subtree is a string that designates the function to be applied and the other subtrees are the required parameters. Parameters are represented using general expression abstract syntax trees.

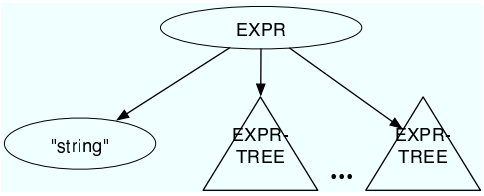

Figure 6: Structure of an AST describing a general expression.

37 Table 1 references all the predefined functions that are allowed for WFPN[5].

38 For the *varref* operator, the second subtree is reduced to a node labeled by a string.

39 For the *const* operator, the second subtree is reduced to a node labeled by a string that represents the value of the constant. Let us note that, for the *const* function, the type of the value can be deduced from the ”surrounding” information (color type of a place, associated variable in a comparison, etc.).

40 For the *int* operator, the second subtree is reduced to a node labeled by an integer value.

2.2.2 Boolean expressions

41 Figure 7 shows the general structure of an expression tree, which is also recursive. The root of a boolean expression abstract tree is labeled by the keyword **BEXPR**. The first subtree is a string that designates the function to be applied and the next subtrees are the required parameters.

42 Table 2 references all the boolean functions that are allowed for WFPN[6].

[5] User-defined functions should be handled in an extension of this format. Can it be independent from a given tool?

[6] User-defined functions should be handled in an extension of this format. Can it be independent from a given tool?

Function	Parameter(s)	Definition
$--$	variable or token constant; positive	predecessor function of rank N
$++$	variable or token constant; positive	successor function of rank N
$\times$	natural; expression, variable or token constant	multiplication of the expression
$+$	two expressions, variables or token constants	set addition of two expressions
$-$	two expressions, variables or token constants	set substraction of two expressions
all	a color class or a color domain	broadcast function that associates one token of each possible color of the class or domain.
product	at least one expression	cartesian product of the involved expressions
extract	an expression; a positive	extraction of the n^{th} element in a product that is represented by an expression (for example, a reference to a structured variable)
varref	one identifier	reference to a declared variable
const	one identifier	reference a valid value of a color class
int	integer value	reference to a integer constant for use when necessary in the structure of an expression

Table 1: List of predefined non boolean operators.

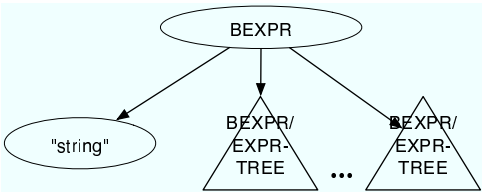

Figure 7: Structure of an AST describing a boolean expression.

2.3 Places

43 Each place has to reference a declared color type. The abstract syntax tree is reduced to a node labeled by a color class or a color domain. The string *null* denotes the absence of color type (tokens in the place are untyped tokens).

44 Each place may reference a marking. If absent, the default marking of a place is assumed to be ”no token”.

45 The description of a place marking has to respect the structure presented in figure 8. The root of the abstract syntax tree is labeled using the keyword **MARKING**. This root has only one subtree that is a general expression tree (see section 2.2.1) with the following restrictions:

- if the color type associated with the place is *null* (no color type), then the expression is an integer expression (function *int*) that denotes the number

Function	Parameter(s)	Definition
$>$	2 expressions corresponding to an elementary tokens	comparison (greater than)
$<$	2 expressions corresponding to an elementary tokens	comparison (less than)
$\geq$	2 expressions corresponding to an elementary tokens	comparison (greater or equal than)
$\leq$	2 expressions corresponding to an elementary tokens	comparison (less or equal than)
$=$	2 expressions corresponding to a token	comparison (equal)
$\neq$	2 expressions corresponding to a token	comparison (different)
and	2 boolean expressions	boolean *and*
or	2 boolean expressions	boolean *or*
not	1 boolean expression	boolean negation
true	none	true boolean function
$false$	none	false boolean function

Table 2: List of boolean operators.

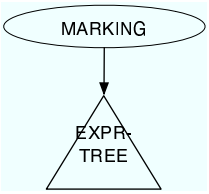

Figure 8: Structure of an AST describing the marking of a place.

of non colored tokens located in the place.

- if there is a color type associated with the place, then the subtree should denote a finite number of tokens. Thus, only the following functions can be found in the definition of the marking: $\times$ (when several tokens have the same profile), $+$ (union of tokens), all (the broadcast function), $-$ (subtraction from a set of tokens[7])

  all is a faster way to enumerate all the values of a color type.

  product allows to build structured tokens.

  const references a possible value of the type.

  int refers to any integer value when necessary.

2.4 Arcs

46 The description of an arc term has to respect the structure presented in figure 9. The root of the abstract syntax tree is labeled using the keyword **ATERM**. This root has only one subtree that is a boolean expression tree (see section 2.2.2) with the following restrictions:

[7] It is typically associated with the broadcast function, for example, C.$all - value$ that means "all the possible values for class C except the one that is listed".

- when the arc is connected to a untyped place, Then the PNML notation for Petri net is assumed.
- the root function below the **ATERM** can only be tagged by $+$ (union of tokens), $-$ (substraction from a set of tokens), *product* (to build structured tokens), $\times$ (when several tokens have the same profile), *all* (the broadcast function), *varref* (to use a color variable that is declared and visible).
- below the root of the expression (see previous item), all expression functions can be referenced.

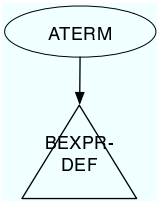

Figure 9: Structure of an AST describing arc terms.

2.5 Transitions

47 A transition may have a condition that prevents its firing for some bindings. If no condition description is supplied, then a default value **true** is assumed. It means that the transition can be fired for any possible bindings from the input places.

48 The description of a transition condition has to respect the structure presented in figure 10. The root of the abstract syntax tree is labeled using the keyword **TCOND**. This root has only one subtree that is a boolean expression tree (see section 2.2.2) with the following restriction. The general expressions contained in the boolean expression tree can only contain the following functions: $--$ and $++$ (to compare a modified input token), *product* (to hold a structured token) , *extract* (to extract a component in a structured token when necessary) , *varref* (to reference a color variable of an input arc label), *const* (to denote a valid value of a color class), *int* (to refer any integer when it is necessary).

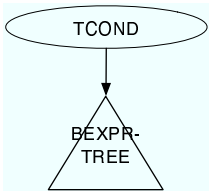

Figure 10: Structure of an AST describing the condition of a transition.

49 Note that a transition may have a condition only if at least one of its input places contains colored tokens.

References

[1] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On Well-Formed Coloured Nets and their symbolic reachability graph. In G. Rozenberg and K. Jensen, editors, *LNCS : High Level Petri Nets. Theory and Application*. Springer Verlag, June 1991.

[2] GreatSPN: GRaphical Editor, Analyzer for Timed, and Stochastic Petri Nets. url : `http://www.di.unito.it/~greatspn/`.

[3] The CPN-AMI Home page. url : `http://www.lip6.fr/cpn-ami`.