

A High-level Petri Net Ontology Compatible with PNML

Juan C. Vidal, Manuel Lama, and Alberto Bugarín

Department of Electronics and Computer Science.

University of Santiago de Compostela.

E15782 Santiago de Compostela, Spain.

{jvidal,lama,alberto}@dec.usc.es *

Abstract. In this paper, a High-level Petri Nets (HLPN) ontology is presented. This ontology is based on the International Standard ISO/IEC 15909-1 and describes semantically and declaratively both the static and dynamic behaviour of a HLPN. The ontology is composed by: *(i)* a concept taxonomy that describes the main components of a HLPN, following the vocabulary and semantics specified in the standard; and *(ii)* a set of axioms that constraint how the instances of the taxonomy are created, restricting the range and domain of the concept relations, and the values of the concept attributes. These axioms guarantee that the HLPN is correctly constructed, and define how the HLPN should be executed. The ontology is compatible with the PNML specification through a number of transformations that enable the translation from the PNML elements into the ontology concept taxonomy. The ontology has been implemented in the Frame Logic and OWL languages, and it can be downloaded from <http://www-gsi.dec.usc.es/ontologies/hlpn>.

1 Introduction

Petri Net Modelling Language (PNML) [1] is an XML-based format whose aim is to facilitate the interchange of Petri net definitions between different tools or applications. It is important to emphasize that PNML focuses on the representation issues of the net components (such as places, transitions, arcs, terms, etc.), specifying the vocabulary needed to represent a given Petri net. Thus, if two tools share the same vocabulary and they give the *same semantics* to each term of this vocabulary, the exchange of the Petri net would be guaranteed. From the point of view of its technological use, the main drawback of the PNML is that its XML-based specification does not provide a set of axioms that constraint, for example, the range of the attributes or how the instances of the concepts are created. Therefore, the semantics of the Petri net rest on the tools compatible with the specification. With the use of ontologies to represent Petri nets, this issue could be solved.

* Authors wish to thank the Xunta de Galicia, Martínez Otero Contract, S.A. and the Ministerio de Ciencia y Tecnología for their financial support under the projects PGIDIT04SIN206003PR, PGIDIT04DPI096E and TIC2003-09400-C04-03.

In recent years ontologies have evolved in line with the need of technologies and products to dynamically mediate discrepancies in business semantics. Ontologies contribute with a common semantics that improve the communication among communities. Ontologies arise from the fact that a common vocabulary is not enough. In this sense, an XML is only the first step to ensuring that computers can communicate freely, but does not provide any semantics. As a formal and explicit specification of a shared conceptualization [2], ontologies capture the shared knowledge of a given domain and represent this knowledge in a declarative and expressive language (explicit) that follows a logical paradigm (formal). Therefore, an ontology looks for the description of the concepts (aka classes), attributes, and relations between those concepts, but also for the axioms that constraint their meaning. In fact, without axioms the ontology has a free interpretation and, consequently, no semantics.

In this paper, we propose an ontology stack that semantically describes High-level Petri Nets (HLPN) [3,4]. Each layer of this stack uses the elements defined in the lower layers and is composed by the following ontologies: *(i)* an ontology of the data types used in the specification of a HLPN; *(ii)* a knowledge representation ontology that deals with the primitives (for example, concepts, attributes, axioms, etc.) needed to represent the domain objects; *(iii)* a HLPN ontology that describes semantically all the components of the HLPN. This ontology is based on the ISO/IEC 15909-1 standard [5] and it translates the HLPN mathematical specification into a machine-readable formulation (like the Frame Logic (F-Logic) or OWL languages). In this way, the common understanding of HLPN can be shared among people but also among software agents or programs. Our ontology provides the set of axioms that constraints the definition of HLPN and avoids the definition of incorrect Petri nets. We must remark that contrary to other modelling proposals, our ontology moreover than dealing with the static structure of HLPN also defines its dynamic behaviour: both execution and state. This allows the sharing of the current state of a HLPN execution among several agents but also to reason about its behaviour.

Another advantage of the ontology is the separation between the domain knowledge (which describes the domain objects) from the operational knowledge. The second and third layer of our ontology stack facilitates this objective. In this way, the domain knowledge can be defined independently from the Petri nets that use it to annotate the arcs, define terms or colour the places. The algebraic specification of HLPN also facilitates this separation. HLPN are defined independently from the domain objects and use an operational terminology to define the colours of the net. Thus, it is possible to configure the operational terminology of a HLPN to deal with different domain knowledge and thus enable its reuse. This strategy also allows the reuse of HLPN as a piece of knowledge: HLPN specify an operational knowledge that is defined by means of an ontology, like domain knowledge. In this sense, other technologies, like problem-solving methods, semantic web services, and software agents, can base its operational semantics on a HLPN ontology instance and associate to it a specific domain knowledge.

The paper is structured as follows: section 2 defines the need of an ontology for HLPN description and section 3 describes the stack of ontologies developed to model HLPN. In this section, we pay special attention to the development of the Knowledge Representation and HLPN layers, section 3.1 and 3.2, respectively. Then, sections 4 and 5 present a comparison with PNML meta-models and other works related to the modelling of HLPN. Finally, an example and the main conclusions of the paper are summarized.

2 The Need for a HLPN Ontology

This section describes the advantages of our proposal for modelling HLPN. For this purpose, our solution is compared with the Petri Net Markup Language (PNML) which is the major trend for modelling Petri Nets. PNML is an interchange format for Petri nets, which supports the exchange of all kinds and versions of Petri nets among different Petri net tools. It should be noted, that the comparison only refers to the way by which PNML models HLPN (between the PNML meta-models and the ontology) and its technological solution (between RELAX NG and F-Logic or OWL).

2.1 From a Modelling Point of View

The purpose of this ontology is to translate the HLPN mathematical specification of ISO/IEC 15909-1 standard [5] into a machine-readable formulation. In this sense, a first difference with regard to PNML is that our ontology does not deal with the definition of hierarchical (such as pages or modules) or graphical (such as styles or positions) concepts. In our opinion, since these concepts are not defined in the ISO/IEC 15909-1 standard, they cannot be defined in this layer of the ontology. However, an upper layer can be defined to deal with them. For example, a layer can be defined over the HLPN layer in order to specify the hierarchical concepts (such as pages, fusion or substitution nodes) that extends the semantics of HLPN.

From a modelling point of view, the main difference between both proposals is that PNML does not deal with the modelling of the state and execution of HLPN. In our opinion, although this feature is not one of the current objectives of PNML, it is a key factor in the description of HLPN. In fact, state and execution models provide the basis for reasoning over the behaviour of a HLPN. For example, the explicit definition of these models will facilitate the use of HLPN in the definition of Web or Grid services. In these domains, state and execution semantics is needed to reason about the behaviour of the services.

The static model of HLPN graphs defined in the PNML meta-model is similar to the one described in our ontology. The main difference is related to the algebraic specification of HLPN. Our model keeps the separation between the signature and the algebra used in the ISO/IEC 15909-1 standard. The motivation to preserve this separation is to facilitate *the reuse of algebras*. For example,

a HLPN that deals with a supply chain management can be used in different domains each of them with a different algebra (domain ontology).

Both solutions provide a set of axioms that constraint the instances of the taxonomy. However, PNML meta-model does not define the OCL sentences that ensures the correct definition of signatures or terms. For example, that the terms used in transition guards have a boolean sort or that the list of subterms of a function application define the same domain as the function the term is based on.

2.2 From a Technological Point of View

PNML is the major input for Part 2 of the International Standard ISO/IEC 15909 on a Transfer Format for High-level Petri Nets [6]. PNML specification for HLPN design is based on a conceptual model that defines the vocabulary and the functional relations between the concepts involved in the definition of a Petri net. PNML specification has been formally modelled through the RELAX NG language [7]. However, the knowledge model of this language is not expressive enough to describe the semantics (or meaning) associated to the elements of the PNML specification. Thus, the main limitations of the RELAX NG language are:

- Hierarchical (is-a) relations between two or more concepts cannot be explicitly defined. Therefore, there are no inheritance mechanisms facilitating the representation of concept taxonomies. For example, in the PNML specification, the **Place** and **Transition** elements do not inherit the attributes and relations of the **Node** element: they are just included as XML sub-elements of the **Node** element. Figure 1 compares both the F-Logic and RELAX NG specifications based on the definition of the **Node** concept. F-Logic is the formalism used in our proposal for ontologies representation (see section 3.1). In F-Logic, the hierarchical relation between the **Place** and **Node** concepts is explicitly defined through the *place :: node*, and, therefore, the attributes of **Node** will be automatically inherited by **Place**. In RELAX NG, however, the attributes of **Node** are directly added to the definition of **Place**, but no hierarchical relation is established between them.
- Properties of relations cannot be defined. RELAX NG language does not provide primitives to represent neither mathematical properties (like symmetry or transitivity) nor taxonomic properties (like disjoint and exhaustive partitions) of a relation. For example, the PNML specifies that an instance of the **Place** cannot be a **Transition** for any given **Node**, what means that **Place** and **Transition** are disjoint concepts. Figure 1 shows how this taxonomic property can be explicitly expressed in an axiom of F-Logic.
- General and formal constraints (or axioms) between concepts, attributes, and relations cannot be specified. These axioms describe more precisely the semantics of the concepts as they constrain how the instances of the concepts could be created. For instance, the axiom an arc relates nodes of different type could not be represented in the RELAX NG language. This restriction,

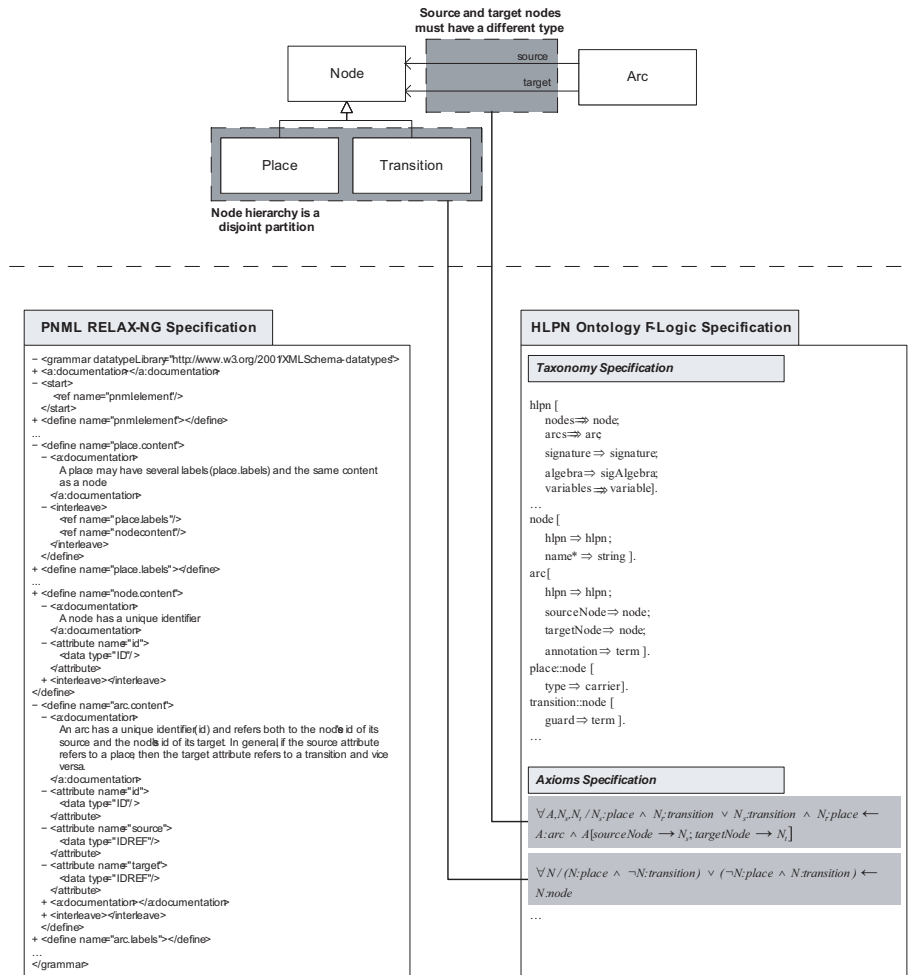


Fig. 1. Comparison between the PNML and HLPN Ontology specifications

however, can be defined in ontology languages like F-Logic [8] (see Fig. 1) or Ontolingua [9] or OWL [10].

In our opinion, an ontology for modelling HLPN must fit the International Standard ISO/IEC 15909-1 [5]. In this sense, it should take its mathematical specification that specifies the constraints imposed to the design and execution of HLPN. The fact that PNML supports the exchange of all kind of Petri nets can also be a drawback if we talk about ontologies since it could favour the definition of concepts too general and with less semantics. However, as a transfer format, PNML cannot define a complete semantics for these concepts. In this sense, it cannot ensure the correctness of the net. For example, the following PNML code

```

<pnml xmlns="http://www.informatik.hu-berlin.de/top/pnml/ptNetb">
  <net id="example1" type="http://www.informatik.hu-berlin.de/top/pntd/ptNetb">
    <name>
      <text>incorrect representation of a Petri net graph structure</text>
    </name>
    <place id="p1">
      <name>
        <graphics><offset x="0" y="22" /></graphics>
        <text>p1</text>
      </name>
      <initialMarking>
        <graphics><offset x="-20" y="10" /></graphics>
        <text>1</text>
      </initialMarking>
      <graphics><position x="50" y="150" /></graphics>
    </place>
    <place id="p2">
      <name>
        <graphics><offset x="-1" y="20" /></graphics>
        <text>p2</text>
      </name>
      <initialMarking>
        <graphics><offset x="24" y="5" /></graphics>
        <text>0</text>
      </initialMarking>
      <graphics><position x="250" y="150" /></graphics>
    </place>
    <arc id="a1" source="p1" target="p2">
      <inscription>
        <graphics><offset x="20" y="0" /></graphics>
        <text>1</text>
      </inscription>
      <graphics><position x="75" y="75" /></graphics>
    </arc> </net>
  </pnml>

```

Fig. 2. Example of an incorrect Petri net definition using PNML

depicted in Fig. 2 describes an incorrect definition of a Petri net where two places are linked by an arc. In this case, PNML has not the semantics to check that the source and target attributes of an arc are of type node and that these nodes are of different type (place or transition). PNML only specifies that the source and target attributes are IDREF types.

Therefore, the HLPN specification (and PNML) requires a representation language capable of describing explicitly and formally the semantics of its elements. This language should enable the semantic description of the conceptual model as well as the definition of formal axioms related to the correct definition and execution of a Petri net. The ontology proposed here has been implemented in F-Logic and OWL languages, which have the required features to represent and reason both concepts and axioms. This ontology could be used to access and check the correctness of the Petri net in the following way (Figure 3): *(i)* translate the Petri net from the RELAX NG PNML specification into the F-Logic language; *(ii)* access to the ontology through a general reasoner (for example, FLORA2 for F-Logic) that will check whether the ontology axioms are verified or not. With this reasoner a Petri net tool or an API designed to manage the

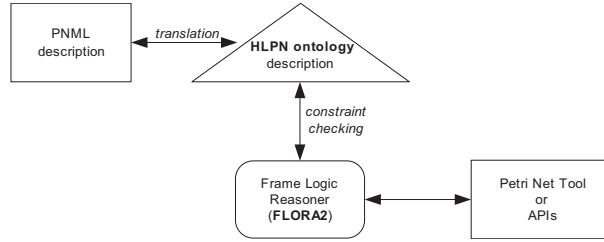


Fig. 3. Use of the ontology to check the definition of a Petri net

ontology [11] does not need to codify those set of axioms; and *(iii)* if required, modify the ontology and then the PNML specification.

3 HLPN Description Ontology

In this section, we propose a layered ontology for modelling the High-level Petri Nets (HLPN). This ontology is depicted in Fig. 4 and it is built over 3 layers. Each layer is itself defined by an ontology and describes the elements that will be used in the upper layers.

- Data Types Ontology (DTO). This layer contains the data types associated to the concept attributes of the domain ontology. The data types included in this ontology are the same as the ones defined in the XML Schema Data Types specification [12].
- Knowledge Representation Ontology (KRO). This ontology is on the top of the XML Schema Data Types and describes the representation primitives used to specify the domain ontology managed by the Petri nets. The components of the domain knowledge will be instances of this ontology and will be used to define the colours of the Petri nets and to reason about the domain ontology. This explicit representation of the domain ontology is necessary to perform inferences about the input and output data that the Petri nets use in its operations. The KRO defines the primitives for describing classes, attributes, methods and axioms.
- High-level Petri Nets Ontology (HLPNO). This layer describes by means of the KRO the formal semantic of the HLPN. This ontology is based on the International Standard ISO/IEC 15909-1 [5] but also takes some of its features from the International Standard ISO/IEC 15909-2 [6].

The following subsections will describe with more detail the described ontologies stack. First the basic concepts related to the KRO level will be briefly introduced in section 3.1. Then, the last layer of this ontology, which is the main focus of this paper, is described in section 3.2. The objective is to provide the basis for the definition of both the structure and the behaviour of HLPN graphs.

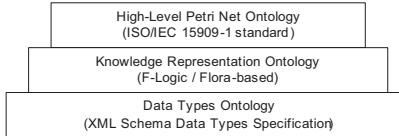


Fig. 4. HLPN ontology stack

3.1 Knowledge Representation Ontology

This ontology describes the representation primitives used to specify the domain ontology managed by HLPN in its operations. That is, the components of the domain ontology will be KRO instances. This ontology is needed because higher ontologies could need to reason about the domain ontology. For example, the conditions imposed by a transition guard over an attribute.

The knowledge model used to develop domain ontologies is the F-Logic formalism. F-Logic accounts in a clean and declarative fashion for most of the structural aspects of object-oriented and frame-based languages. These features include object identity, complex objects, inheritance, polymorphic types, query methods, encapsulation and others. F-Logic has a model-theoretic semantics and a sound and complete resolution-based proof theory. Theoretical foundations of F-Logic are described in [8].

Along this paper, the taxonomy structure of the ontology is described by means of semantic networks. Its underlying idea is that the meaning of a concept comes from the way in which it is connected to other concepts. In this sense, a semantic net consists of a set of nodes and links where the nodes represent objects and descriptive information about those objects and links describe the relationships between the nodes. As regards, we extend the notation of semantic networks and represent nodes by means of boxes, "has" relations by arrows and "is-a" relations by white head arrows.

The use of semantic networks is for clarity: the F-Logic definition of the ontology is represented in a visual way. In fact, a semantic network can be directly translated to a F-Logic definition changing nodes by classes, "has" relations by class slots and "is-a" relations by class hierarchies. F-Logic syntax is described in the following example which describes the semantic network of Fig. 5.

```

node[]
token[]
transition :: node
place :: node[tokens ⇒ token]
arc[sourceNode ⇒ node, targetNode ⇒ node]

```

This network defines a graph structure which could represent low-level Petri nets. The following F-Logic code defines the instantiation depicted in Fig. 5 which represents a Petri net with a p_1 place, a t_1 transition and an a_1 arc with relates the place with the transition.

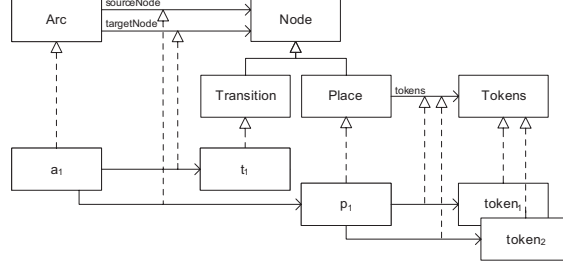


Fig. 5. Semantic network example

$t_1 : transition$
 $p_1 : place[tokens \rightarrow \{token_1, token_2\}]$
 $a_1 : arc[sourceNode \rightarrow p_1, targetNode \rightarrow t_1]$

In these description: (i) the ":" represents the subclass-relationship while the ":" represents the class membership, (ii) double-shafted arrows \Rightarrow and \Leftrightarrow specify types while the arrows \rightarrow and \rightsquigarrow describe values of attributes and (iii) double-headed arrows \Leftrightarrow and \rightsquigarrow are used in conjunction with set-valued attributes, while \Rightarrow and \rightarrow signify scalar attributes.

The selection of F-Logic as the KRO is based on its facilities for the definition of axioms. In this sense, some of current ontology languages do not support the definition of axioms. Moreover F-Logic provides the tools to reason over the ontology.

3.2 High-level Petri Nets Ontology

This section describes the ontology for defining HLPN. We developed a *concept taxonomy*, which describes the elements and relations of the international standard ISO/IEC 15909-1 [5], and a set of *axioms*, which formally constraint the semantics of the concept taxonomy. It should be noted that this model is not limited to the static model of a HLPN. A static model is a model that does not evolve and can only represent invariants and particular states [13]. As the purpose of a HLPN is to be executed, this ontology also defines the dynamic model of HLPN, and makes explicit its underlying execution rules.

High-level Petri Nets Static Model. The HLPN definition taxonomy is depicted in Fig. 6. The proposed ontology describes HLPN as graphs by means of its nodes, arcs, variables, signature and sig-algebra (many-sorted algebra) concepts. The **Node** concept is a design abstraction for representing both places and transitions. **Place** concept associates a carrier to each place (**type** attribute) which restricts the tokens that can mark this place. **Transition** concept defines the guard term of a transition. The **Arc** concept completes the definition of the graph structure and connects a source node to a target node.

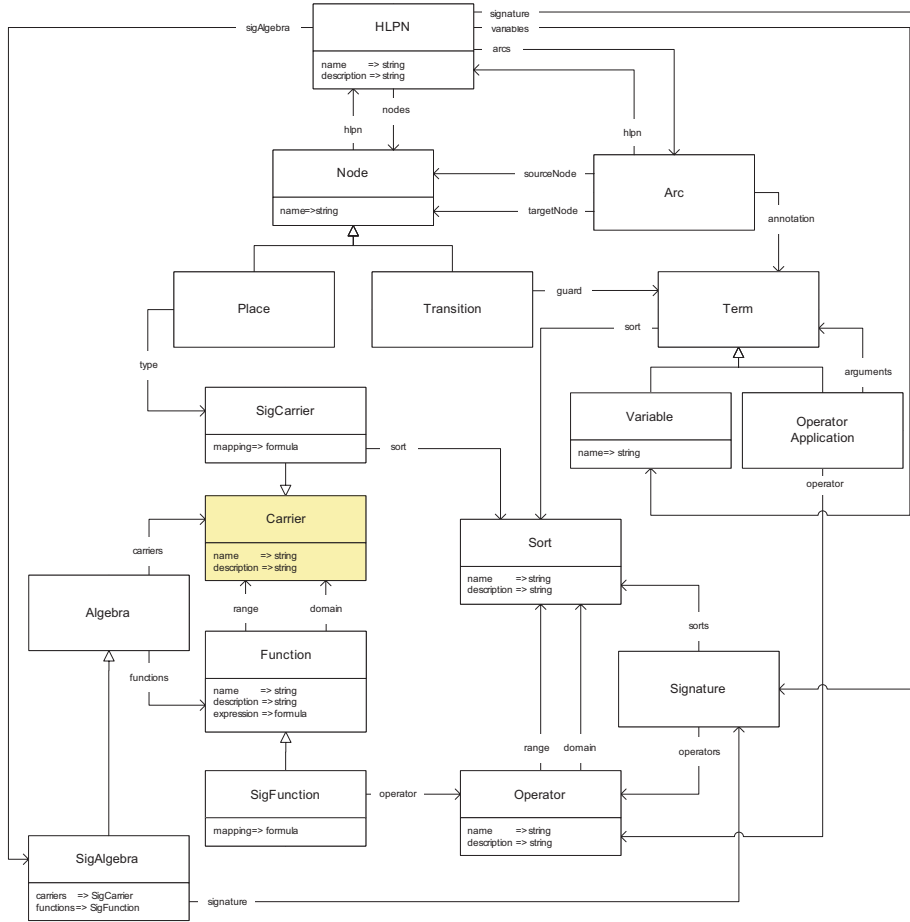


Fig. 6. Static model of the High-level Petri Nets Ontology

The algebraic specification for defining arc annotations and transition guards is defined by means of the **Signature**, **SigAlgebra** and **Term** concepts. The **Signature** concept provides a way to characterize algebras at a syntactic level. The separation between (i) sorts and carriers and (ii) operators and functions allows the reuse of algebras (semantic level) for the same signature (syntactic level). Moreover, this separation allows us to define HLPN independently from the algebra used for its execution and to relate both by means of the sig-algebra which acts as a mediator between the logical and the Knowledge Representation levels. The signature defines the set of sorts and operators a HLPN can use in its algebraic definition. The **Sort** concept defines by means of its **name** attribute the syntactic representation of a concept or data type defined in an algebra.

However, sorts can be more complex structures and must be extended in order to define multisets, products or domain sorts. The same happens with the **Operator** concept as regards functions. An operator is defined by its name and its arity. The arity is defined by means of sorts, where the attributes **domain** and **range** define a list of input sorts and the output sort of the operator.

The **Algebra** concept has the same meaning as the signature although at the Knowledge Representation level. In this sense, an algebra defines the carriers (types) and functions of the language. The difference is that the **Carrier**¹ concept defines data types and the **Function** concept defines the semantics of a function that can be directly interpreted or called by the engine that interprets the Knowledge Representation ontology. As regards, functions incorporate the **expression** attribute that stores a formula that gives the desired behaviour to the function. The **SigAlgebra** concept is an extension of the **Algebra** concept and acts as a mediator between a signature and an algebra. A sig-algebra is composed by a set of **SigCarrier** and **SigFunction** concepts. The first one mediates between sorts and carriers. For example, between a *Person* sort defined in a signature and a *User* carrier defined in an algebra, the mediator must map the attributes between the *Person* and *User* concepts. The second one mediates between operators and functions. This concept ensures that the domain and range of an operator are compatible with those of a function.

Arc and transition annotations are represented by means of terms. Terms are expressions comprising constants, variables (e.g., x , y) and operator applications (e.g., $f(x, y)$) built from the signature. The **OperatorApplication** concept represents a constants or an operator call where the attribute **arguments** defines the list of input terms of the operator.

High-level Petri Nets Dynamic Model. At this point, the model does not allow the analysis of a particular state of the net, but it states that static elements may be defined independently from any situation. The situation and execution models are depicted in Fig. 7. The upper node of this model is the **HLPNExecution** concept which defines (i) the initial state, (ii) the current state and (iii) the sequence of firings (state changes) from the initial to the current state of the net.

The **Marking** concept defines a particular state of a HLPN, i.e. it defines the set of tokens in each place of the net. The **PlaceMarking** concept defines the state of a place and relates a multiset of elements with the same type. In this context, a marking stores the state of the net at a particular state while tokens represent the knowledge associated to this state.

In a certain sense, a flow of knowledge is defined between two markings. The **Firing** concept is in charge of defining this change of state between a source and a target marking. At a given state, if a transition fires, it must remove the tokens from each input places and add the new ones to each output place. In order to define this behaviour, a firing is related to the fired transition and to the set of term evaluations defined for both the transition guard, and its input and output

¹ **Sort** and **Carrier** concepts and its hierarchies define meta-classes of the model.

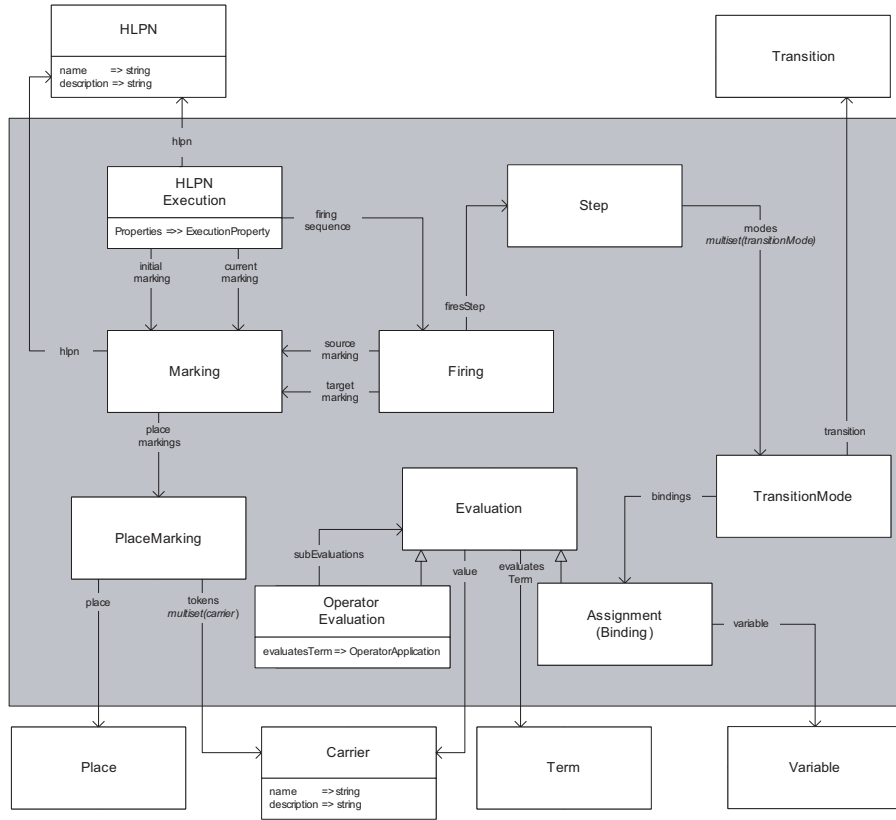


Fig. 7. Dynamic model of the High-level Petri Nets Ontology

arc annotations. However, the defined concept is not limited to the firing of one enabled transition at a source marking. The **TransitionMode** concept defines the enabling of a transition for a set of variable bindings (**Assignment** concept). As regards, the **Firing** concept allows the firing of a step of simultaneous occurrence of a multiset of transition modes that are concurrently enabled in a marking.

The **Evaluation** concept plays an important role in the definition of a firing. Term evaluations define the enabled transitions and the values of the arc annotations. In this sense, the target marking of a firing is computed by means of the evaluations of a fired transition. The proposed ontology defines two type of evaluations: variable evaluations and operator call evaluation. On one hand, the **Assignment** concept (aka binding) represents the evaluation of a variable and assigns it a carrier value. On the other hand, the **OperatorEvaluation** concept evaluates operator application terms. This kind of evaluation represents the operator call with the evaluation of its term arguments as subterms. One

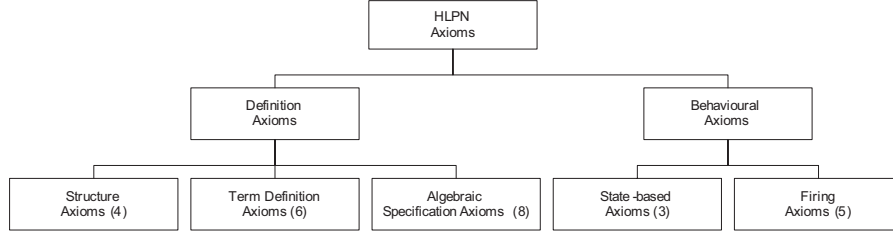


Fig. 8. Classification of the axioms identified in the HLPNO

last feature of evaluations must be mentioned. In the design of this ontology, we considered that evaluations should be defined as an explicit concept of the ontology. The motivation is that it is not always possible to execute a term. For example, suppose that a term is supported by a web service call and that this service is not always available or it is not free.

Description of the HLPN Ontology Axioms. The taxonomy defined in both the static and the dynamic models is not yet sufficient to have a precise specification of the way in which a HLPN is executed. The previous models provide a good vision of the HLPN but a more detailed definition is necessary in order to define its full semantics.

From a modelling point of view, the formal definition of the semantics constraints of the HLPN ontology concepts is the main advantage of this ontology when compared with other specifications. On one hand, the semantics of the concepts is *completely* included in the ontology (an ontology is more than a simple taxonomy structure), and, on the other hand, it will ensure that the HLPN models created with this ontology are correct.

The HLPN definitions of the standard ISO/IEC 15909-1 contain a mathematical description of the semantics of all the taxonomy concepts, including the *constraints* their instances should verify. These constraints were identified and formalized in F-Logic logic and structured according to the classification depicted in Fig. 8. The *structural*, *term definition*, and *algebraic specification axioms* complete the semantics of the static model. *State-based* and *firing axioms* restricts the semantics of the dynamic model.

4 Compatibility with PNML

One of the objectives of this ontology is to make it compatible with the specified in the International Standard ISO/IEC 15909-2. Mappings are defined for the three different levels defined in PNML for HLPN specification. It is clear that the models proposed by the PNML and HLPN ontology are different. PNML tries to define a common file transfer format for all kind of Petri nets among

different tools while our ontology aims for the definition and execution of HLPN. However, it is important to maintain the compatibility between both specifications especially considering that PNML is the standard de facto for transferring Petri nets. In this sense, PNML files could be the inputs of our ontology which will validate that the transferred nets are in compliance with the HLPN specification. For example, that the domain and range of an operator are defined by sort elements of the signature.

Compatibility with the PNML Core Model. This meta-model of PNML defines *(i)* the graph structure of Petri Nets, *(ii)* the graphical information associated to this structure and *(iii)* tool-related information. It should be noted that graphical and tool-related concepts are out of the scope of our ontology. For example, `ToolInfo` or `Graphics` concepts are not taken into account. The main differences between both models are:

- The `PetriNetDoc` concept is not modelled in the HLPN ontology because a document does not pertain to the HLPN conceptual level.
- Concepts such as `Page`, `RefTrans` or `RefPlace` are not defined in this HLPN ontological layer since we do not take in consideration the concepts related to hierarchical composition of HLPN.
- The way in which some attributes are modelled. For example, PNML Core Model defines a set of `Label` attributes related to the nodes and arcs of the Petri net graph. In a certain sense, this is due to the extensibility of PNML which has been created to allow the representation of all kinds of Petri Nets. However, from an ontological point of view, this relationship does not contribute any semantics.

Table 1 shows some of the mappings defined for PNML Core Model. In this table, each attribute of a PNML concept is related to its equivalent definition in F-Logic. A XSLT template has been implemented for this purpose.

Compatibility with the PT-Net Model. This meta-model of PNML refines the Core Model and allows the representation of Place/Transition Nets. This model introduces the initial marking and annotation concepts in order to associate a natural number as the initial state of places and a non-negative natural number to arcs. Since both concepts will be extended in the PNML HLPNG Model we do not compare them with our ontology. However, the most interesting feature of this meta-model is the introduction of OCL axioms in the UML diagram. Specifically, this model introduces two axioms that avoid arcs to connect nodes of the same type (a place to a place or a transition to a transition). These axioms are also captured in our ontology within the structural axioms (see Fig. 8). It should be noted that due to our KRO, some extra axioms are needed to define the cardinality constraints. For example, the fact that an arc has an input and an output node is defined through the following axiom:

$$\forall A, N_S, N_T / A[sourceNode \rightarrow N_S; targetNode \rightarrow N_T] \leftarrow A : arc$$

Table 1. Transformations between the HLPN ontology and the PNML Core Model

<i>PetriNet ISO/IEC 15909-2 concept</i>		
<i>Attribute</i>	<i>Mapping</i>	<i>F-Logic definition</i>
<i>id</i>	<i>The instance identifier is implicitly defined in the HLPN concept.</i>	<i>X:hlpn</i>
<i>type</i>	<i>This attribute relates PNML to a specific type of Petri net. Our ontology is defined for the PNTD of HLPN.</i>	–
<i>Place ISO/IEC 15909-2 concept</i>		
<i>Attribute</i>	<i>Mapping</i>	<i>F-Logic definition</i>
<i>id</i>	<i>The instance identifier is implicitly defined in the Place concept.</i>	<i>X:place</i>
<i>name</i>	<i>Takes the value of the place slot identified by "name".</i>	<i>place[name → X]</i>
<i>Transition ISO/IEC 15909-2 concept</i>		
<i>Attribute</i>	<i>Mapping</i>	<i>F-Logic definition</i>
<i>id</i>	<i>The instance identifier is implicitly defined in the Transition concept.</i>	<i>X : transition</i>
<i>name</i>	<i>Takes the value of the transition slot identified by "name".</i>	<i>transition[name → X]</i>
<i>Arc ISO/IEC 15909-2 concept</i>		
<i>Attribute</i>	<i>Mapping</i>	<i>F-Logic definition</i>
<i>id</i>	<i>The instance identifier is implicitly defined in the Arc concept.</i>	<i>X : arc</i>
<i>source</i>	<i>Takes the value of the arc slot identified by "sourceNode".</i>	<i>arc[sourceNode → X]</i>
<i>target</i>	<i>Takes the value of the arc slot identified by "targetNode".</i>	<i>arc[targetNode → X]</i>

Compatibility with the HLPNG Model. The Petri Net Type Definition (PNTD) of PNML defines the specific concepts of a special kind of Petri net. Although the PNTD for HLPN is still not defined, an outline of this specification has been proposed in the International Standard ISO/IEC 15909-2. This meta-model extends the PT-Net Model with signature, type and term concepts. The difference between both models is stated below:

- **Signature** concept of our ontology does not include the declaration of function and sorts. These concepts are implicitly declared.
- **Term** hierarchy is modelled in the same way. The difference between both models is linked with the way in which PNML defines the **Type** of a place, the **Condition** of a transition, the **HLAnnotation** of an arc or the signature of a Petri Net. PNML defines the previous concepts as annotations and relates these annotations through the **structure** attribute with **Type**, **Term** and **Signature** concepts. In our opinion, the solution of PNML is influenced by the extensibility and graphical requirements. Our ontology relates directly a places with its type, and a transition guard or arc annotation with its term.
- PNML attribute **hlinitialMarking** defines the initial marking of a place by means of a term. Contrary to this definition, our ontology follows the

Table 2. Axioms included in the HLPNO but not included in PNML HLPNG Model

<p>Structural axioms <i>The node hierarchy is defined as an exhaustive partition so that every node is defined as a place or a transition. This axiom also states that the partitions defined by places and transitions are disjoint (a node cannot be simultaneously a place and a transition).</i></p>
$\forall N / (N : \text{place} \wedge \neg(N : \text{transition})) \vee (\neg(N : \text{place}) \wedge N : \text{transition}) \leftarrow N : \text{node}$
<p>Term definition axioms <i>The term hierarchy is defined as an exhaustive disjoint partition so that every term is defined as an operator application or a variable and cannot be simultaneously an operator application and a variable.</i></p>
$\forall T / (T : \text{variable} \wedge \neg(T : \text{operatorApplication})) \vee (\neg(T : \text{variable}) \wedge T : \text{operatorApplication}) \leftarrow T : \text{term}$
<p>Term definition axioms <i>Transition guard terms have a boolean sort.</i></p>
$\forall T, G / G[\text{sort} \rightarrow \text{bool}] \leftarrow T : \text{transition}[\text{guard} \rightarrow G : \text{term}]$
<p>Term definition axioms <i>The sort of a term used in an arc or a transition annotation is defined in the sort set of the hlpn signature.</i></p>
$\forall A, HN, T, S, Sig / Sig[\text{sorts} \rightarrow \{S\}] \leftarrow A : \text{arc}[\text{hlpn} \rightarrow HN : \text{hlpn}; \text{annotation} \rightarrow T : \text{term}] \wedge T[\text{sort} \rightarrow S : \text{sort}] \wedge HN[\text{signature} \rightarrow Sig : \text{signature}]$
$\forall G, HN, T, S, Sig / Sig[\text{sorts} \rightarrow \{S\}] \leftarrow T : \text{transition}[\text{hlpn} \rightarrow HN : \text{hlpn}; \text{guard} \rightarrow G : \text{term}] \wedge G[\text{sort} \rightarrow S : \text{sort}] \wedge HN[\text{signature} \rightarrow Sig : \text{signature}]$
<p>Term definition axioms <i>The arguments of an operator application have the same sort as the domain of the operator it is based on. The "compareSorts" predicate checks that the sorts of a list of terms and a list of sorts match.</i></p>
$\forall L1, L2, O, T / T[\neg \text{arguments} \rightarrow L1] \wedge O[\neg \text{domain} \rightarrow L2] \vee T[\text{arguments} \rightarrow L1] \wedge O[\text{domain} \rightarrow L2] \wedge \text{compareSorts}(L1, L2) \leftarrow T : \text{operatorApplication}[\text{operator} \rightarrow O : \text{operator}]$
<p>Term definition axioms <i>The type of a place and the sort of its input arc annotations are related by the algebra.</i></p>
$\forall A, C, P, S, T / C[\text{sort} \rightarrow S] \leftarrow A : \text{arc}[\text{targetNode} \rightarrow P : \text{place}; \text{annotation} \rightarrow T : \text{term}] \vee P[\text{type} \rightarrow C : \text{sigCarrier}] \vee T[\text{sort} \rightarrow S : \text{sort}]$

Standard ISO/IEC 15909-1 and defines a marking as a set of place markings which relates a multiset of tokens with a place.

- An algebra translate the signature specification to an operative specification. This specification is the major difference between PNML meta-model and the static model of our ontology. In this sense, PNML only defines the sig-carriers of the algebra (concept **Type**). The motivation of our ontology is to provide algebras that could be reused both in the definition of HLPN and in the interchange of HLPN between tools. The sig-algebra defines both the carriers that interpret the sorts and the functions that interpret the operators defined at the syntactic level. With regard functions, its definition provides the formulas that define their operational behaviour.

The OCL axioms included in this level are related with terms definition and are also included in the ontology. We must remark that our ontology contains more axioms to constraint the signature definition, the term hierarchy definition or the graph structure. Tables 2 and 3 show the axioms identified in our ontology that are not included in the PNML specification. We omit the axioms related to the algebra of the HLPN. Axioms are represented in F-Logic. It should be

Table 3. Axioms included in the HLPNO but not included in PNML HLPNG Model

<p>Algebraic specification axioms <i>The signature associated to a HLPN is a Boolean Signature, that is, a signature which contains the bool sort.</i></p>
$\forall HN, Sig / bool : sort \wedge Sig[sorts \rightarrow \{bool\}] \leftarrow HN : hlpn[signature \rightarrow Sig : signature]$
<p>Algebraic specification axioms <i>The domain and range of an operator are defined by sort elements of the signature. This axiom also checks that each of the input sort of the operator (list of input parameters) is defined in the signature of the operator.</i></p>
$\forall D, O, S, Sig / Sig[sorts \rightarrow \{S\}] \leftarrow Sig : signature[operators \rightarrow \{O : operator\}] \wedge O[domain \rightarrow D : list(sort)] \wedge member(S : sort, D)$
$\forall O, S, Sig / Sig[sorts \rightarrow \{S\}] \leftarrow Sig : signature[operators \rightarrow \{O : operator\}] \wedge O[range \rightarrow S : sort]$
<p>Algebraic specification axioms <i>A Boolean Signature includes the $true_{bool}$ and the $false_{bool}$ constants which correspond to the values true and false in any associated algebra, respectively.</i></p>
$\forall HN, Sig / true_{bool} : operator[range \rightarrow bool; \neg domain \rightarrow L] \wedge Sig[operators \rightarrow \{true_{bool}\}] \leftarrow Sig : signature[sorts \rightarrow \{bool\}]$
$\forall HN, Sig / false_{bool} : operator[range \rightarrow bool; \neg domain \rightarrow L] \wedge Sig[operators \rightarrow \{false_{bool}\}] \leftarrow Sig : signature[sorts \rightarrow \{bool\}]$
<p>Algebraic specification axioms <i>The type of a place is a carrier defined in the algebra of the hlpn. In PNML this axiom should check that the type of a place is included in the signature of the HLPN.</i></p>
$\forall Alg, C, HN, P / Alg[carriers \rightarrow \{C\}] \leftarrow P : place[hlpn \rightarrow HN : hlpn; type \rightarrow C : sigCarrier] \wedge HN[sigAlgebra \rightarrow Alg : sigAlgebra]$

noted that F-logic represents the *implication* connective with the "←" symbol. In F-logic, this connective is defined $\varphi \leftarrow \psi$ as a shorthand for $\varphi \vee \neg\psi$ (the same semantics as in classical logic). Along the axioms description, we assume the definition of *list* data type and its predicates.

5 Comparison with other ontologies

To our knowledge, only one Petri net ontology has been proposed in the literature [14]. This ontology is based on the basic Petri Net Markup Language (PNML) [1] from which it takes the graph representation of the net and the concepts related to the graphical information used by visual editors. Basically, this ontology extends the basic PNML specification with the introduction of (i) some axioms to ensure the correctness of the graph structure, (ii) the colour of the tokens and (iii) the state of the net. However, this ontology presents some lacks:

- The graphical information is conceptualized in the HLPN layer. In our opinion, graphical concepts cannot be defined with HLPN concepts. Concepts of graphical nature related to positions, colours or font styles cannot be merged with the HLPN definition. These concepts should be defined at another ontological level that refines the HLPN ontology.
- The algebraic specification of HLPN is not treated. Term, signature and algebra-related concepts are simply not defined. In fact, transitions and arcs are not related to guard conditions or annotations, respectively. This limits the sharing and reuse of domain knowledge between HLPN.

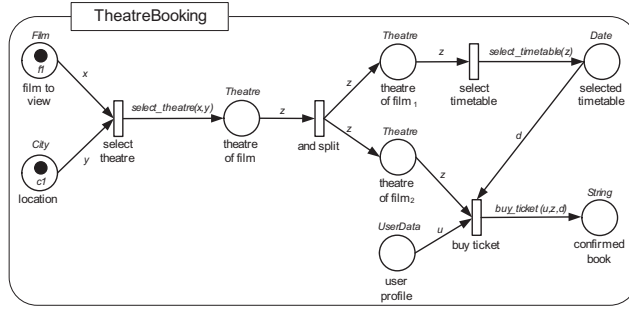


Fig. 9. HLPN of the theatre booking service

- Places are not typed. It is possible to assign tokens with different colours to a place. No axiom is defined to constraint this behaviour.
- The state of HLPN is weakly defined. The state is defined by the marking and the token concepts. Places are related to an initial marking and a current marking and the markings to tokens. This solution limits the expressiveness of the model since the sequence of markings is lost.
- The state of a places cannot be related with a multiset of tokens. Markings are directly related to tokens.
- The execution model of the HLPN is not defined. In this sense, this ontology is not able to define a sequence of firings and states of an execution. Concepts like term evaluations, bindings or transition modes are also lost. This lack also limits the reasoning over a particular state or a dynamic property of the net.

It should be noted that this ontology seems to have been developed to provide an axiomatic framework to the PNML and not for defining HLPN. In fact, it does not provide the concepts or the semantics to define HLPN and less to reason over a particular state of the net.

6 Ontology Instantiation Example

Let us suppose a Semantic Web Service for booking a theatre ticket for a particular film ($film_f$) in a given city ($city_c$). Figure 9 shows this service's design using HLPN. As we can see, the service comprises three subservices: *select theatre*, *select timetable* and *buy ticket*. Information about the credit card is included in the user data ($userdata_u$), and the *buy ticket* service validates it. In this example, subservices are modelled as functions of the algebra.

Throughout this section, the description of the example is written in F-Logic [8]. The following description defines the HLPN instantiation of the service depicted in Fig. 9. In this example, due to space limitations only the main features of the service will be described.

```

theatreBooking : hlpn[ name → "theatre booking service"; variables → {x, y, z, d, u};
                      signature → sig1; sigAlgebra → sigAlg1; arcs → {a1, a2, a3, ...};
                      nodes → {filmToView, location, selectTheatre, theatreOfFilm, ...}
...
filmToView : place[name → "Film to view"; type → film]
location : place[name → "Location"; type → city]
theatreOfFilm : place[name → "Theatre of film"; type → theatre]
...
selectTheatre : transition[name → "Select theatre"; guard → term1]
...
a1 : arc[sourceNode → filmToView; targetNode → selectTheatre; annotation → x]
a2 : arc[sourceNode → location; targetNode → selectTheatre; annotation → y]
a3 : arc[sourceNode → selectTheatre; targetNode → theatreOfFilm; annotation → term1]

```

The algebraic specification of the *theatreBooking* service is defined by means of the *sig*₁ signature and the *sigAlg*₁ algebra. The first one specifies the set of operators and sorts (syntactic types) used in the definition of the HLPN. Following with the example, the signature must at least define the *select_theatre*, *select_timetable* and *buy_ticket* operators and the sorts used by these operators (among others the *movie*, *city* or *cinema* sorts).

```

sig1 : signature[ sorts → {movie, city, cinema, ...};
                  operators → {select_theatre, select_timetable, buy_ticket}]
select_theatre : operator[ name → "select theatre";
                           description → "Selects a city theatre which has the film still on";
                           domain → [movie, city]; range → cinema]
...

```

The *sigAlg*₁ algebra translates the signature specification to an operative specification. In fact, several algebras are compatible with a HLPN specification. For example, the following code defines two possible domain ontologies that could have been used to define the carriers of the *theatre booking service*:

```

film[name ⇒ string; directed_by ⇒ string; played_by ⇒ actor].
city[name ⇒ string; country ⇒ string].
theatre[name ⇒ string; location ⇒ city; present ⇒ picture].
...
picture[name ⇒ string; directed_by ⇒ string; played_by ⇒ string].
location[name ⇒ string; state ⇒ string; country ⇒ string].
cinema[name ⇒ string; location ⇒ address; capacity ⇒ integer; show ⇒ film].
...

```

It should be noted that Fig. 9 is depicted with the first ontology. Suppose now that the *movie* sort is defined as *movie* : *sort*[*movie_name* ⇒ *string*, *director* ⇒ *string*, ...] and that the *film* sig-carrier will interpret this sort. Mappings are needed to relate both concepts. For example, *director* slot of the *movie* sort must be related with the *directed_by* slot of film carrier. The following mapping axiom will link both properties:

$$\forall D, F / M[\textit{directed_by} \rightarrow D] \leftarrow F : \textit{film}[\textit{sort} \rightarrow M; \textit{director} \rightarrow D]$$

Our ontology defines mapping axioms as *formulas*. In this sense, it is possible to define a simple renaming (as the previous example) or more complex formulas.

The following lines describe how the *film* sig-carrier is linked with the *movie* sort. The same happens with regard the operator *select_theatre* that is related to a function of the algebra. It should be noted that functions have its behaviour defined in the expression method.

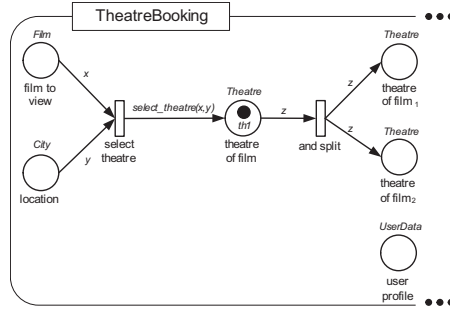


Fig. 10. Firing of select theatre transition

```

sigAlg1 : sigAlgebra[ carriers → {film, city, theatre, ...};
                    functions → {select_cheaper_theatre, ...}]
film : sigCarrier[sort → movie; ...] city : sigCarrier[sort → cityName; ...]
...

```

Finally, terms are defined by means of constants, variables (e.g., x , y) and operator applications built from the signature. Terms define tree/graph structures where operator applications are the composite nodes and the variables and constants are the leaf nodes of the tree/graph. For example, the following sentences describe the definition of the term $select_cheaper_theatre(x, y)$.

```

select_cheaper_theatre : function[ name → "select cheaper theatre";
                                  operator → select_theatre;
                                  domain → [film, city]; range → theatre;
                                  expression → $ select cheaper theatre(F, C) ← ... $]
...

```

This is an *operator application* definition based on the operator $select_theatre$ previously defined. In its definition we can see how (i) the sort of the term matches with the range and (ii) the sort of its arguments match with the domain of the operator it is based on.

```

term1 : operatorApplication[ sort → cinema; arguments → [x, y]; operator → select_theatre]
x : variable[name → "x"; sort → movie]
y : variable[name → "y"; sort → cityName]

```

Let us suppose that Fig. 9 defines the initial marking of the *Theatre Booking HLPN* and Fig. 10 defines its state after the firing of the $select_theatre$ transition. The following code defines the HLPN execution and both markings of the net. It should be noted that we use lists to simulate multisets.

```

hlpnExecution1 : hlpnExecution[ hlpn → theatreBooking; firings → [f1];
                               initialMarking → m0; currentMarking → m1]
m0 : marking[hlpn → theatreBooking; placeMarkings → {pm10, pm20, pm30, ...}]
m1 : marking[hlpn → theatreBooking; placeMarkings → {pm11, pm21, pm31, ...}]

pm10 : placeMarking[place → filmToView; tokens → [f1]]
pm20 : placeMarking[place → location; tokens → [c1]]
pm30 : placeMarking[place → theatreOfFilm; tokens → []]

```

```

pm10 : placeMarking[place → filmToView; tokens → []]
pm20 : placeMarking[place → location; tokens → []]
pm30 : placeMarking[place → theatreOfFilm; tokens → [th1]]

```

The following code describes the firing of the *selectTheatre* transition. In this case, the step fired contains only one transition mode. This mode defines the transition fired and the bindings that enable the transition.

```

fi1 : firing[firesStep → step1; sourceMarking → m0; targetMarking → m1]
step13 : step[modes → [tm1]]
tm1 : transitionMode[transition → selectTheatre; bindings → {eva1, eva2}]
...

```

The following axiom is defined to check the concurrent enabling of transition modes. This axiom translate the mathematical axiom described in the International Standard ISO/IEC 15909-1 and checks that a step *S* of transition modes is enabled in a marking, *M_S*, *iff* for all places of the net, the sum of the input arc evaluations for the list of modes defined in the step is greater than the source marking of the fired transition.

$$\begin{aligned}
&\forall D, F / ms_minus(SMTL, SI, S1) \leftarrow HE : hlpnExecution[hlpn \rightarrow HN; firings \rightarrow FL] \\
&\wedge member(F, FL) @ prolog(basics) \wedge F : firing[sourceMarking \rightarrow M_S; firesStep \rightarrow S] \\
&\wedge S : step[modes \rightarrow TML] \wedge P : place[hlpn \rightarrow HN] \wedge sum_input_values(P, TML, SI) \\
&\quad \wedge M_S..placeMarkings[place \rightarrow P; tokens \rightarrow SMTL]
\end{aligned}$$

In this definition the *sum_input_values(P, TML, SI)* predicate obtains a list, *SI*, with the values of the evaluations of the arc annotations between the place *P* and transitions defined in the transition mode list (*TML*). This list is applied to the *ms_minus(SMTL, SI, S1)* predicate which subtracts this list to the multiset of tokens, *SMTL*, defined for the place marking of place *P*. The same strategy has been followed to define the transition rule.

7 Conclusions and Future work

The ontology presented in this work is based on the International Standard ISO/IEC 15909-1, which is the current result of the consensus of the Petri net community on what the semantics and representation of the HLPN is. This is one of the main contributions of our work: the HLPN ontology models the shared knowledge of the PN community, and it could be *reused* in applications that follow the International Standard ISO/IEC 15909-1.

On the other hand, the HLPN ontology describes the semantics of the static and dynamic models of the HLPN. Particularly, the set of axioms that guarantee that Petri nets are correctly created and its execution follow the semantics defined in the International Standard ISO/IEC 15909-1. This is the other contribution of the work: the HLPN ontology incorporates all the semantics needed to define and execute a HLPN. Other approaches such as the PN ontology presented in [14] or the PNML specification are compared with this ontological proposal.

As future work, a hierarchical ontology on the top of the HLNP ontology will be developed. This ontology will describe the hierarchical semantics of a HLPN on the basis of the composition mechanisms proposed in the literature. The static and dynamic models of this hierarchical HLPN ontology will extend those proposed in the HLPN.

References

1. M. Weber and E. Kindler: The Petri Net Markup Language. In: Petri Net Technology for Communication-Based Systems. Volume 2472 of Lecture Notes in Computer Science. Springer-Verlag (2003) 124–144
2. A. Gómez-Pérez, M. Fernández-López, and O. Corcho: Ontological Engineering. Advanced Information and Knowledge Processing. Springer-Verlag (2003)
3. K. Jensen: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin (1992)
4. W. Reisig: Petri Nets and Algebraic Specifications. Theoretical Computer Science **80**(1) (1991) 1–34
5. J. Billington, et al.: High-Level Petri Nets - Concepts, Definitions and Graphical Notation. Final Draft International Standard ISO/IEC 15909 (2002)
6. E. Kindler: High-level Petri Nets - Transfer Format. Working Draft of the International Standard ISO/IEC 15909-2 (2004)
7. J. Clark and M. Murata: RELAX NG Specification (2001) <http://www.oasis-open.org/committees/relax-ng>.
8. M. Kiefer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of ACM **42** (1995) 741–843
9. T. Gruber: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition **5**(2) (1993) 199–220
10. M. Dean and G. Schreiber: OWL Web Ontology Language Reference. (2004) <http://www.w3.org/TR/owl-ref>.
11. L. Hillah, F. Kordon, L. Petrucci, and N. Trves: Model engineering on Petri Nets for ISO/IEC 15909-2 : API Framework for Petri Net types metamodels. Petri Net Newsletter **69** (2005) 22–40
12. P. V. Biron and A. Malhotra: XML Schema Part 2: Datatypes. (2001)
13. E. Breton and J. Bézivin: Towards an understanding of model executability. In: FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems, Ogunquit, Maine, USA (2001) 70–80
14. D. Gasevic and V. Devedzic: Reusing Petri Nets through the Semantic Web. In: Proceedings of the 1 st European Semantic Web Symposium. Volume 3053 of Lecture Notes in Computer Science., Heraklion, Greece, Springer-Verlag (2004)