A Meta-Model Based Approach for Definition of a Multi-Formalism Modeling Framework

Hamid Mohammad Gholizadeh and Mohammad Abdollahi Azgomi

Abstract—**Many software tools and environments are developed for modeling discrete-event systems. Although, most of the existing tool are proposed for modeling with one or a few modeling languages. In this paper, we propose a meta-modeling approach for definition of a multi-formalism modeling framework for Petri nets and related formal modeling languages. The proposed framework and the related software tool facilitate the inclusion of formalisms in the framework in a unified manner. The proposed meta-modeling structure is designed in four layers such that the most abstract layer as the meta-formalism and the least abstract layer as the concrete model. The basic concepts of the meta-modeling structure and some examples of formalisms are presented based on the proposed structure. We explain the techniques and the architecture of the implemented tool based on the proposed meta-modeling structure for the framework.**

Index Terms—**Modeling, meta-modeling, modeling language, formalism, tool, discrete-event systems.**

I. Introduction

A model is an abstract representation of a system and a meta-model is an abstract description of a model. The abstraction helps to neglect the less important aspects of a system, while concentrating on favorable parts that are desired to a specific study. However, abstraction helps us to study more phenomena and systems in a unified manner while they may seem to be completely different at first.

Nowadays, there are many simulation and modeling tools supporting different kinds of modeling languages. The comprehensive list of the existing modeling tools is listed in [1], which most of them are dedicated to support only one or few modeling languages. Mostly, the models constructed by these tools are not interoperable, since this matter has not been the main concern of the tool developers. For example, a Petri net model constructed in SHARPE tool [2] cannot be used in CPN Tools [3] and vise-versa.

Usually, tool developers' main concerns are how to implement a tool for a new derived or proposed language and little work is done on how to develop modeling tools that are extensible enough to support different formalisms or solution techniques. Besides, systems and models are growing in complexity and mostly a single formalism is not suitable for modeling all parts of a complex system. Therefore, we need an approach to construct models that are composed of several sub-models of diverse language types. Sometimes, some parts of a model are previously created and the issue is how to compose them into a single model. But, this integration is rarely supported by the existing modeling tools.

The above mentioned concerns are motivations for exploiting meta-modeling in developing a new multi-formalism framework. The meta-models usage makes the framework flexible enough to support diverse modeling languages in an integrated and unified manner. We have used meta-modeling concepts in definition of formal modeling languages or formalisms, model-classes and models in a new modeling framework.

In this paper, we propose a meta-modeling approach for definition of a multi-formalism modeling framework for Petri nets [4] and related formal modeling languages that can be represented using graphs, such as extensions of Petri nets. The proposed framework and the related software tool facilitate the inclusion of a wide range of formalisms in the framework in a unified manner. We explain the meta-modeling structure and discuss its implementation using the extensible markup language (XML). We present detailed definitions of some important parts of the framework to help understanding the whole framework. The complete formal definitions are already published in [5]. In this paper, we use XML as a base language for all of the definition regarded to meta-models structure, since some useful features, such as XSD and XSLT, are available with it, which suitably match to our needs in meta-model definition. There are also many accessible programming components implemented for XML documents' manipulation, which is a considerable factor in developing a tool for the framework. We briefly discuss the architecture of the tool based on the meta-models defined in this paper.

The rest of this paper is organized as follows. In section II, a brief survey on related works is presented. In section III, the proposed four layered meta-model structure and the related definitions are presented. In section IV, some sample formalisms are defined based on the proposed meta-modeling structure. In section V, the architecture and techniques used for implementing a tool based on the meta-modeling structure are introduced. Finally, in section VI, some concluding remarks are mentioned.

II. Related Works

Nowadays, models and meta-models are widely used in the area of software engineering. There are many researches

Manuscript received July 21, 2009.

Hamid Mohammad Gholizadeh was with School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran (corresponding author: phone: +98-21-77240540-50 (ext. 3325); fax: +98-21-7724046; e-mail: hgholizadeh@comp.iust.ac.ir).

Mohammad Abdollahi Azgomi is with School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran. (e-mail: azgomi@iust.ac.ir).

working on the topics related to model-driven engineering (MDE). The Meta-Object Facility standard [6] was originated by OMG as a meta-modeling architecture for definition of the unified modeling language (UML).

Several powerful modeling tools are available for modeling with UML. Rational Rose Suite, PowerDesigner and ArgoUML [7] are some examples of commercial and non-commercial ones. However there are many modeling and simulation tools, which most of them simply use de facto application development techniques for creating a modeling environment. UML is a standard supported by several tools that has an XMI Language [8] interface for exchanging UML models between different tools.

The problem in the scope of formal modeling languages is that there is a variety of languages and a variety of tools supporting them, while there is not a comprehensive methodology and framework for composition of these models in an integrated environment.

Also, some trends exist in introducing new techniques in creating multi-model multi-formalism environment. Möbius [9], AToM^3 [10] and OsMoSys [11] are examples of multi-formalism modeling tools that are found in the literature. Among these works, OsMoSys is closer than the others to our work in using meta-models. It is intended to support multiple formalisms in a common framework. As the best of our knowledge, this framework has not a complete formal definition. In [10], the framework is defined in a semi-formal manner and it is not possible to precisely define new formalisms in the framework. OsMoSys model solution approach is based on a new formal language definition, named SPDL, which forces a modeler to learn its complex syntax.

The Möbius modeling tool is the result of another try to create a multi-formalism framework. Its idea is based on defining an abstract function interface (AFI), which is a common application programming interface (API) for adding new formalisms to the framework and using its feature [12]. Möbius has respectful features in model composition and solution techniques, but adding a new formalism to the framework is not an easy task. Since its first version, which natively supports stochastic activity networks (SANs) [13, 14], performance evaluation process algebra (PEPA) [15] and MoDeST [16] are the two only formalisms, which have been implemented in the framework.

The final tool we discuss is AToM^3 [10]. It uses meta-models to support modeling by different modeling languages. However, it does not offer model solution. For solving models, the modeler should transform them into DEVS [17], and then can apply DEVS solution techniques for evaluation of models. Hence, the modeler cannot use the original solution techniques for models, which may be more efficient and useful. Apart from the above mentioned tools, other famous tools, such as CPN Tools [3], do not support multiple formalisms and their extensibilities are mostly limited. Some multi-formalism tools, such as SHARPE [2], support a fixed set of models (ex. Markov models, queuing models, stochastic Petri nets, etc.), and some built-in steady-state or transient solvers and simulators.

III. The Proposed Meta-Model Structure

In this section, we define a meta-model structure, which is used as a base for the definition of our modeling framework. The meta-model definition for the formalism should support the vast variety of formalisms for discrete-event systems. Therefore, when we abstract the formalisms structure like Petri nets, SANs, CPNs [18] and etc., we reach to a simple graph including some node and some edges. Fig. 1 illustrates this concept. We separate the behavior and structure of the formalism in the framework to add flexibilities into the framework. The properties of each element of the formalism distinguish it from the other formalisms. These properties are annotated to each element in the meta-model definition. This method makes the framework compatible with a wide range of existing formalisms. Every formalism is defined based on the meta-formalism definition. For example, for Petri nets, the place and transition elements are defined as nodes and arcs are defined as edges of a graph in the framework. The graphical notations, captions and tokens are defined as properties for each of the Petri net elements.

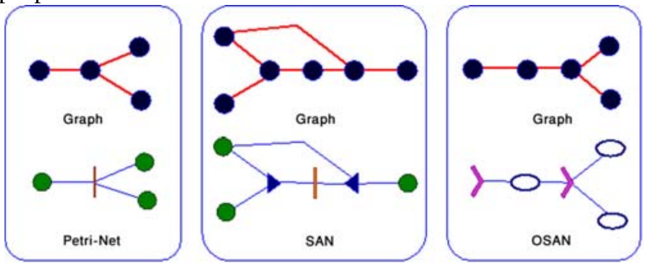


Figure1 Examples of mapping the formalism elements onto the graph elements

A model is an instance of the related formalism. It includes a number of elements with some valued properties. A model in the framework is considered as a model-class and is organized in separate layers in the meta-model structure. The model-classes are not solvable models and could be instantiated to make a solvable model.

The proposed meta-model structure is shown in Fig. 2. As shown in this figure, there are four layers in the meta-model structure of the framework that are as follows:

1) *Meta-formalism layer*, which is the top most abstract layer,
2) *Formalism layer*, which is based on meta-formalism,
3) *Model-class layer*, which should be defined based on a formalism that is defined in the formalism layer, and
4) *Model layer*, which includes the final solvable models.

Fig. 2, also depicts the analogies between the framework meta-model structure and MOF [8] structure. It demonstrates how these four layers are mapped onto the four layers that exist in the MOF meta-model structure. For example, solvable models in the framework's meta-model structure are similar to the object diagrams and the model-classes are also similar to the user-defined models.

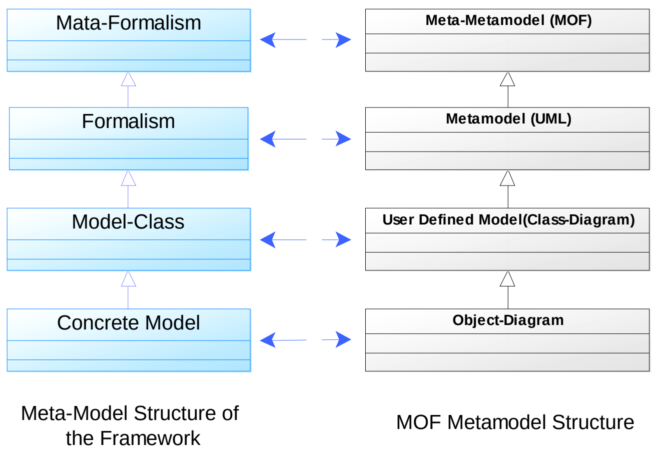


Figure2 The meta-model structure of the framework and its mapping onto the MOF meta-model structure

A. *Meta-Model Structure Definitions*

The first top layer in the meta-model structure is the most abstract layer. In this layer, models are considered as a collection of elements with some properties. The Element Type can be of type Node, Edge or Model. There are some properties for every element. The type of these properties may be ordinary, enumeration, class, object, function or a set of them. Class types are data structures defined inside the formalism definition. If we define a property as an object type, we mean that this property refers to a class type that its definition is postponed to model-class layer. Simply, we can consider it like pointers in programming languages. Object type feature in the framework definition is useful in implementing some formalisms like CPNs [18] and CSANs [19], where the modeler can define a new structure inside the model itself and then assign the type to the coloured places.

Now, we present two definitions:

Definition 1. An *element* of a formalism in the framework is consisted of the following properties:

1) A unique name of the element.
2) A graphical representation of the element (provided to the related tool as a file in a standard graphical format).
3) A type of the element.
4) The name of the related formalism.
5) A finite set of properties of the element.
6) An OCL expression defining the element's constraints.

In some formalism, such as SANs, CPNs and etc., there are some functions in the body of formalism's definitions. Definition of these functions as a property of type string is not precise. To clarify the subject, suppose the input function of input gates in SANs, which can only change the marking of connected places. It is required to have a way to express these constraints in formalism definitions in the framework. We consider such a function as a property of type FUNCTION and define its constraints using the object constraint language (OCL) [20]. The OCL expression in the framework is written based on components and relationships depicted as in Fig. 3.

After defining an element in the framework, we are ready to define the formalism or formal language definition, which is a collection of elements:

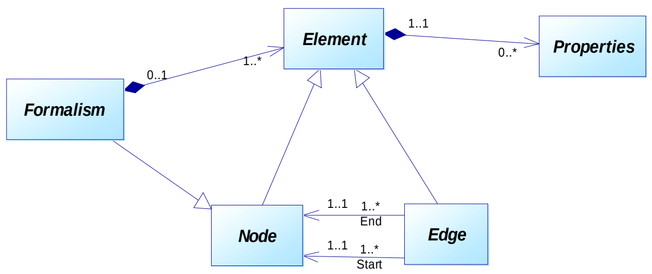


Figure3 Formalism meta-model illustrated in as a class diagram

Definition 2. A *formalism* in the meta-modeling structure is consisted of the following properties:

1) A unique name for the formalism.
2) A graphical representative of the formalism (provided to the related tool as a file in a standard graphical format).
3) A finite set of properties of the formalism.
4) A set of formalism's elements as in Definition 1.
5) References to other defined formalisms in the context, which we may want to use their elements as an ancestor in current formalism definition.
6) A collection of data structures defined inside the formalism.

Considering the above definition, we can summarize a formalism definition in the framework in a UML-like class diagram as shown in Fig. 3. In this figure, it is clearly shown that the formalism can also contain other formalisms. A formalism in the proposed framework is a collection of elements with some properties for each one.

A. Model-Class Definitions

The model-class layer is a layer just below the formalism layer in the meta-modeling structure. It is the models constructed based on previously defined formalisms in the framework. Now, we define this layer, but like before we need to firstly present some preliminary definitions. We define each element of model-class and next present a definition of the model-class itself.

Definition 3. An *element of a model-class* in the meta-modeling structure is consisted of the following properties:

1) The name of the model-class of the element.
2) The type of element. Only those types, which are already defined in the corresponding formalism of the model-class of the element, are permitted.
3) The properties of the element. Each property have the following information:
   - The name of the property.
   - The value of the property which should be compatible with the type of property defined in formalism definition.
   - The property visibility in model-class, which may be private or public.
   - The access modifier, which may be read-only or read-write.
4) The visibility of the element.
5) The access modifier of the element.

Now we continue the meta-modeling structure definitions by defining model-class:

Definition 4. A *model-class* is consisted of the following properties:

1) A name for the model-class.
2) A reference to the corresponding formalism of the model-class.
3) A set of elements, each one is defined as in Definition 3.
4) A set of model-class properties, each have the following information:
   - The name of the property.
   - The value of the property, which should be compatible with the type of property as defined in the formalism definition.
   - The visibility of property, which may be private or public.
   - The access modifier, which may be read-only or read-write.
5) A data structure.

According to the above definitions, it is clear that the model-class itself may have some properties just like its elements. It means that the extra information can easily be annotated to the elements of the model-class or the model-class itself.

At the time of constructing a model in the framework, the model is considered as a model-class and not a concrete model. An instantiated model-class makes a solvable model in the lower layer in the meta-modeling structure. This approach enhances the reusability of models in the framework. A model-class can be instantiated with different values while it is being used as a standalone concrete model or while it is being used as a sub-model in a composed model. At the former situation, the values may be provided by the user and at the latter situation, it may be provided by some elements in the container model. The properties of a model-class and its elements accept modifiers. These modifiers are as follows:

- *public*: The element can be reached inside and outside of the model.
- *private*: The element can only be accessed inside the model.
- *readonly*: The only reading the value of element is permitted outside of the model.
- *readwrite*: Both reading and modifying the element is permitted outside of the model.

A property can accept private or public and readonly or readwrite. The readonly and readwrite modifiers are meaningful when the property is public. We should mention that all of these modifiers are considered in solution stage, because it does not make any sense reading and writing a value when we are not in the solution stage. Since varieties of solvers may be used during the model solution stage by the solution manager of the framework, these modifiers will be quite useful. The solution manager is not discussed in this paper.

IV. Example of Formalisms Implemented in the Framework

Fig. 4 depicts how Petri net is defined in the framework based on meta-modeling structure. Each element has some properties for defining its position and image inside the model, including x, y, width, height and image, respectively. We have not shown these properties in Fig. 4 for conciseness. The OCL expression of the arc element imposes a limitation. The limitation is that it can only connect two elements of type place and transition together. The dollar prefix for the name in the definition means that it is a previously defined type in the context of the framework. All the elements in the framework are derived from the element type (as shown in Fig. 3). Therefore, *start* and *end* properties for the arc element are of type $*Element* in definition of Petri nets.

```
<?xml version="1.0" encoding="UTF-8"?>
<formalism image="petrinet.svg" name="Petrinet">
   <p name="eid" type="int"/>
   ……
   <p name="image" type="String"/>
   <element image="place.svg" name="Place"
     type="Node">
       <p name="eid" type="int"/>
       …
       <p name="token" type="int"/>
   </element>
   <element image="transition.svg"
     name="Transition" type="Node">
       ….
   </element>
   <element image="Arc.svg" name="Arc" type="Edge">
       <p name="eid" type="int"/>
       <p name="start" type="$Element"/>
       <p name="end" type="$Element"/>
       …….
       <ocl>
         context $Arc inv:(self.start=$Place
         implies self.end=$Transition) and
         (self.start=$Transition implies
           self.end=$Place)
       </ocl>
   </element>
</formalism>
```

Figure4 Definition of Petri nets formalism based on the meta-modeling structure

According to the definition of Petri nets, we define SANs to show the flexibility of the meta-modeling structure and its features in the formalism definitions. This definition is based on the formal definition in [13]. As shown in Fig. 5, the timed activity or Timed_Activity uses inheritance to extend a previously defined element in the framework. A previously defined formalism can be included by using ERef keyword in the current definition. The OCL expression for the rate, rateFun and reactivation properties of the timed activity elements imposes some limitations on the function. It indicates that it cannot change the state of the model after execution of the gate functions related to the activity. Also, there is similar OCL expression with a slight difference for the instantaneous activity or IActivity. The gate functions for input and output gates can only change the state according to formal definitions of SANs [14].

A formalism can include some other formalisms. In Fig. 6, definition of HSANs in the framework is shown, which is based on the definition of SANs. All the elements of HSANs formalism are just like SANs and only a sub-model definition is added. Furthermore, there is a new kind of arcs named SubSANArc, which extends Arc element and connects SAN sub-models to the container model elements. This kind of arc makes it possible to construct hierarchical models. The sub-model element is defined of type Model that is of type Node and Element in a hierarchical form. Also, it contains a property of type Objects, which means that it can contain any kind of data structures as a property. We have only shown the

```
<?xml version="1.0"?>
<formalism image="SAN.svg" name="SAN" ERef="$Petrinet" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="SANSchema.xsd">
    <p name="eid" type="int"/>
    <p name="x" type="int"/>
    <p name="y" type="int"/>
    <p name="width" type="int"/>
    <p name="height" type="int"/>
    <p name="caption" type="String"/>
    <p name="image" type="String"/>
   <element extends="$Petrinet.$Place" image="place.svg" name="Place" type="Node">
   </element>
   <element extends="$Petrinet.$Transition" image="Timed_Activity.svg"
      name="Timed_Activity" type="Node">
      <p name="rate" type="@Type1"/>
      <p name=" reactivation" type="@Type2"/>
      <p name="rateFun" type="@Type3"/>
    </element>
    <element extends="$Petrinet.$Transition" image="Iactivity.svg" name="Iactivity" type="Node">
         <p name="probability" type="@Type4"/>
    </element>
    <element extends="$Petrinet.$Transition" image="IGate.svg" name="IGate" type="Node">
         <p name="predicate" type="@Type5"/>
         <p name="fun" type="@Type6"/>
    </element>
    <element extends="$Petrinet.$Transition" image="OGate.svg" name="OGate" type="Node">
         <p name="fun" type="@Type7"/>
    </element>
    <element extends="$Petrinet.$Arc" image="Arc.svg" name="Arc" type="Edge">
         <p name="func" type="@Type2"/>
         <ocl>
           context $Arc inv:let st=self.start,en=self.end in
           st=$Place implies en=$IGate) and
           st=$IGate implies (en=$Iactivity or en=$Timed_Activity) and
           ((st=$Iactivity or en=$Timed_Activity) implies en=$OGate) and (st=$OGate implies en=$Place)
         </ocl>
    </element>
    <type name="Type1" type="Function">
         <p name="input" type="null"/>
         <p name="output" type="float"/>
         <ocl>
           context $Timed_transition post:self.$SPN->forAll($elements=$elements@pre)
         </ocl> </type>
    <type name="Type2" type="Function">
         <p name="input" type="null"/>
         <p name="output" type="boolean"/>
         <ocl>
           context $Timed_Activity post: self.$SPN->forAll(elements=elements@pre)
         </ocl>
     </type>
     <type name="Type3" type="Function">
         <p name="input" type="null"/>
         <p name="output" type="enum(Exponentioal|Bernoulli|Poisson|Geometric)"/>
         <ocl>
           context $Timed_Activity post: self.$SPN->forAll(elements=elements@pre)
         </ocl>
     </type>
     <type name="Type4" type="Function">
         <p name="input" type="null"/>
         <p name="output" type="boolean"/>
         <ocl>
           context $Timed_Activity post: self.$SPN->forAll($elements=$elements@pre)
         </ocl>
     </type>
     <type name="Type5" type="Function">
         <p name="input" type="null"/>
         <p name="output" type="boolean"/>
         <ocl>
           context $IGate post: self.$SPN->forAll($elements=$elements@pre)
         </ocl>
     </type>
     <type name="Type6" type="Function">
         <p name="input" type="null"/>
         <p name="output" type="null"/>
```

Includes Petri nets definition in the context

Place extends the place element of Petri nets

"@Type1, 2,…, 7 are defined later in the definition

```
        <ocl>
          context $IGate post: let adjacentE:Set($element)=self.$SPN.$elements->select(e:$element|e.end=self),
          allToken:Set(int)=self.$SPN.$elements->collect($Place.token) in
          allToken->exclude(adjacentE->collect(P.token))=
          allToken->exclude(adjacentE->collect(P.token))@pre
        </ocl>
    </type>
    <type name="Type7" type="Function">
        <p name="input" type="null"/>
        <p name="output" type="null"/>
        <ocl>
          context $OGate post: let
          adjacentE:Set($element)=self.$SPN.$elements->select(e:$element|e.end=self),
          allToken:Set(int)=self.$SPN.$elements->collect($Place.token)in
          allToken->exclude(adjacentE->collect(P.token))=  allToken->exclude(adjacentE->collect(P.token))@pre
        </ocl>
    </type>
</formalism>
```

Figure5 Definition of SAN formalism based on the meta-modeling structure and the definition of Petri nets

important parts in HSAN definition and neglected the transitions, gates, places and corresponding arcs definition for brevity.

A. A Sample Model-Class in the Meta-Modeling Structure

Fig. 7(a) is the model-class of a well-known producer/consumer system modeled in Petri nets. It is based on Petri nets definition in Fig. 4 .The token property of place *P1* is defined as a parameter and its value is provided in initialization stage by the user or in solution stage by other connected models. The token property of place *P2* is defined public and read-only, which means that its value can be reached outside of these models, but can be modified only by the model solver, not others. If not explicitly specified, the default values for the visibility and access modifiers are private and readonly, respectively. The graphical representative of this model is depicted in Fig. 7(b).

V. Architecture and Techniques of a Tool for the Framework

Any framework needs a software tool to make it possible to define formalisms, construct the models and utilize the solution or simulation techniques available in the framework. We have implemented a modeling tool based on the proposed meta-modeling structure of the framework. XML is an infrastructure for storing all kinds of data, including the formalism definition, model-class definition, transient data between solvers and solver manager and so on. Using XML simplifies importing models that are not defined exclusively for the framework. For example by implementing a simple extensible stylesheet language transformation (XSLT) document, the models compatible with the Petri net markup language (PNML) [21] can easily be imported to the framework. However, we have used the *scalable vector graphics* (SVG), which is an XML-based image format for representing the images inside the tool.

We have used Java programming language and JavaEE [22] features in implementation of the tool. The architecture of the tool's packages is shown in Fig. 8. According to this figure, the components of the tool are organized in three packages: (1) framework package, (2) formalism package and (3) editor package. The framework package includes all of the framework related modules, which have not any graphical representative. It mostly contains some controlling classes. For example when a user defines a new framework using a wizard inside the framework (as in Fig. 9), the editor uses the *classbuild* package's components to construct the formalism related modules and compiles and deploys them on-the-fly. Using Java Reflection API makes it possible not to recompile the entire framework while the new formalism is defined.

The formalism package includes the modules related to the defined formalism in the framework. This package includes some base classes for the whole formalisms as well as on-the-fly created packages and classes produced by the framework packages, as mentioned earlier. According to each formalism definition, theses classes can differ (i.e. they may contain different properties and methods). The last package is the editor package. This package includes all the modules related to the graphical representation of the tool. For example the classes drawing the toolbox, property box and main window, exist in this package. They manipulate the dynamic change of view while the user switches between models. They also completely handle the model creation and design and use the formalism and the framework packages for completing their functions. Finally, this package handles all the interaction between the tool and the user as a modeler or formalism definer.

The communications between the three main packages are based on the model-view-controller (MVC) pattern [23]. The mapping between these patterns' components and the tool modules are depicted in Fig. 10. The MVC pattern provides flexibility in the tool construction and its functionality. When the user changes the graphical representation of the model, the change is reflected to the corresponding stored XML document according to the pattern mechanism. The controller imposes rules and constraints of the formalism definition during the formalism definition wizard and of model construction on models construction time, according to the meta-modeling defined structure. The formalism should conform to the meta-formalism layer and the model-classes should conform to formalism in the tool, respectively. All of these are controlled by the modules inside the framework package.

VI. Conclusion

In this paper, we introduced a meta-modeling approach used in definition of a new multi-formalism modeling framework. The proposed meta-modeling structure is consisted of four abstract layers and constructs the fundamental structure of the framework. It provides the

```
<?xml version="1.0"?>
<formalism image="HSAN.svg" name="HSAN"
  ERef="$SAN,$Petrinet" >
  ……
    <element extends="$SAN.$Place"
      image="place.svg" name="Place" type="Node">
    </element>
          ….
    <element image="SubSAN.svg" name="SubSAN"
      ERef="$SAN" type="Model">
      <p name="data" type="Object"/>
    </element>
    <element extends="$Petrinet.$Arc"
      image="SubSANArc.svg" name="SubSANArc"
      type="Edge">
      <p name="relFun" type="@Type1"/>
     </element>
     <type name="Type1" type="Function">
        <p name="input" type="null"/>
        <p name="output" type="null"/>
        <ocl>
          context $SubSANArc inv: let
          st:$element=self.start,en:
          $element=self.end in
          (st=$Place implies en=$SubSAN)
          and (st=$SubSAN implies en=$Place)
        </ocl>
      </type>
    </formalism>
```

HSANs models may contain some sub-models of type

Figure6 Definition of HSAN formalism based on the meta-modeling structure and the definition of SANs

framework with the flexibility in defining diverse formalisms. Therefore, the framework is adaptable by a large number of formalisms to use their features.

The OCL expressions are exploited in the meta-modeling structure to define formalisms efficiently and precisely. We illustrated the applicability of the meta-modeling structure by defining some sample formalisms using the features of the framework. The innovative approach in defining a framework for diverse formalisms provides an infrastructure for defining a tool for constructing atomic or composed models. The tool uses XML and its derivations like SVG as data storage format and uses the MVC pattern as its core interaction mechanism.

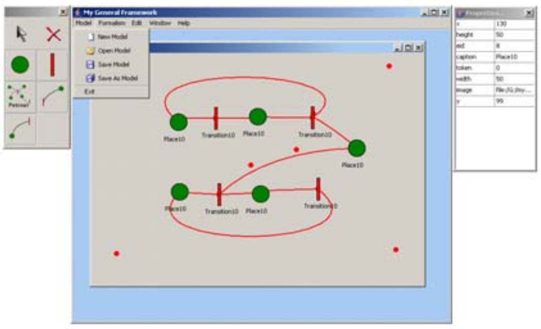
In the future, we intent to continue definition of model composition approach inside the framework based on the proposed meta-modeling structure. The solution strategy should also be completed in the future. Furthermore, we intent to develop some interchange formats for adapting model solvers or simulators inside the proposed framework.

```
<?xml version="1.0" encoding="UTF-8"?>
<model formalism="Petrinet">
    <height>500</height>
    <x>0</x>
    <caption>untitled01</caption>
    <eid>1</eid>
    <width>600</width>
    <image>null</image>
    <y>0</y>
    <element type="Place">
        <x>130</x>
        <height>50</height>
        <eid>8</eid>
        <caption>P1</caption>
        <token>$Param</token>
        <width>50</width>
        <image>/im/place.svg</image>
        <y>99</y>
   </element>
   <element type="Place">
        <x>271</x>
        <height>50</height>
        <eid>9</eid>
        <caption>P2</caption>
        <token visibility="public"
         access="readonly">0
      </token>
      <width>50</width>
       <image>/im/place.svg</image>
      <y>89</y>
  </element>
. . .
</model>
```

Token is defined as a parameter for P1 element

Place property is defined public and read-only

(a)



(b)

Figure7 A sample model-class based on meta-modeling structure: (a) Producer/consumer model-class XML file, (b) The graphical representation of the producer/consumer model

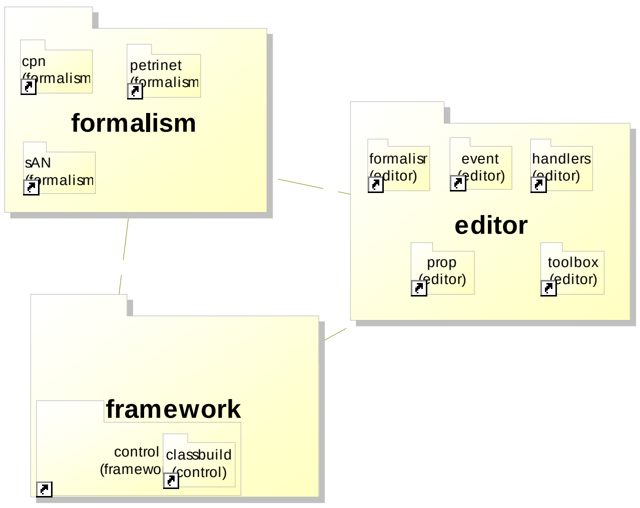
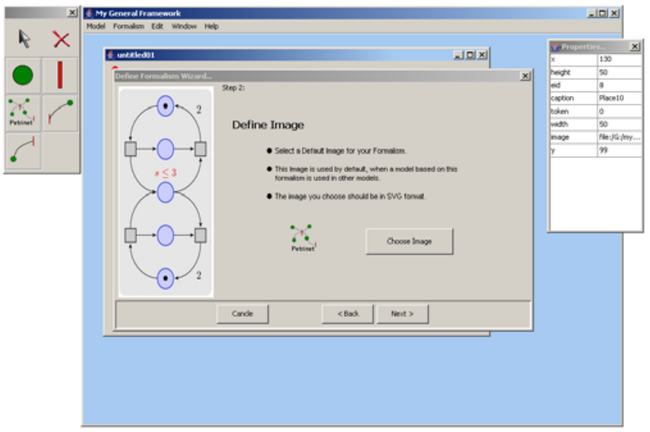


Figure8 The framework tool's packages structure



Define Image

- Select a Default Image for your Formalism.
- This image is used by default, when a model based on this formalism is used in other models.
- The image you choose should be in SVG format.

x	130
height	50
eid	8
caption	Place10
token	0
width	50
image	file:/G:/my....
y	99

Figure9 Formalism definition wizard inside the framework tool

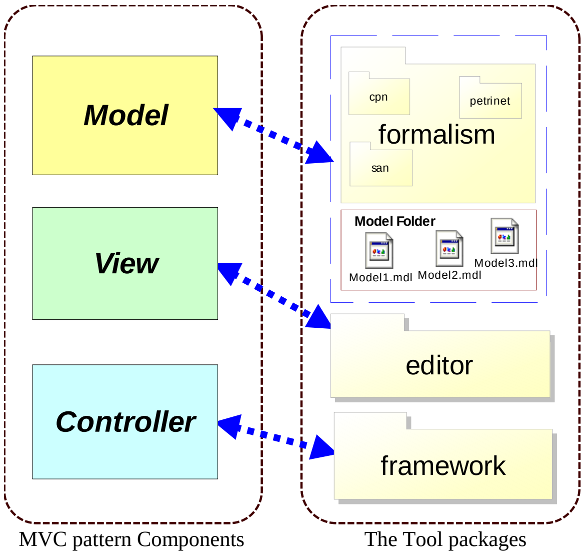


Figure10 Mapping between the tool packages and the MVC pattern components

References

[1] "Petri Nets Tool Database" University of Hamburg, Available: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html.
[2] R. A. Sahner, et al., An Example-Based Approach Using the SHARPE Software Package, Performance and Reliability Analysis of Computer Systems, Kluwer Academic Publisher, 1995.
[3] "CPN Tools: Computer Tool for CP-nets," University of Aarhus, Available: http://www.daimi.au.dk/CPNtools.
[4] J. L. Peterson, Petri Net Theory and the Modeling of Systems, Prentice-Hall, 1981.
[5] H. Mohammad Gholizadeh and M. Abdollahi Azgomi, "An Object-Oriented Modeling Framework for Petri Nets and Related Models," in Proc. of the 7th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'09), Rabat, Morocco, May 10-13, IEEE CS Press, 2009, pp. 546-549.
[6] "Meta Object Facility (MOF) Core Specification," OMG Available Specification, Version 2.0, 2006.
[7] "List of Unified Modeling Language Tools," Available: http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools.
[8] "MOF 2.0/XMI Mapping," OMG Available Specification, Version 2.1.1, 2007.
[9] J. Peccoud, et al., "Möbius: An Integrated Discrete-Event Modeling Environment" Bioinformatics, Vol. 23, No. 24, 2007, pp. 3412-3414.
[10] J. de Lara and H. L. Vangheluwe, "AToM3: A Tool for Multi-Formalism and Metamodelling," in Proc. of the European Joint Conference on Theory and Practice of Software (ETAPS'02), Springer-Verlag, 2002.
[11] V. Vittorini, et al., "The OsMoSys Approach to Multi-formalism Modeling of Systems," Software and Systems Modeling (SoSyM), Vol. 3, No. 1, Springer, 2004, pp. 68-81.
[12] "Möbuis Manual," PERFORM Group, University of Illinois at Urbana, Champaign, 2007.
[13] A. Movaghar, "Stochastic Activity Networks: A New Definition and Some Properties," Scientia Iranica, Vol. 8, No. 4, 2001, pp. 303-311.
[14] W. H. Sanders and J. F. Meyer, Stochastic Activity Networks: Formal Definitions and Concepts, Lecture Notes in Computer Science, Vol. 2090, Springer-Verlag, 2001, pp. 315-343.
[15] G. Clark and W. H. Sanders, "Implementing a Stochastic Process Algebra within the Möbius Modeling Framework," Lecture Notes in Computer Science, Vol. 2165, 2001, pp.200-215.
[16] H. Bohnenkamp, H. Hermanns, J. P. Katoen and R. Klaren, "The MoDeST Modeling Tool and its implementation," in Proc. of the Computer Performance Evaluation Modelling Techniques and Tools (TOOLS'03), Lecture Notes in Computer Science, Vol. 2794, Springer-Verlag, 2003, pp. 116-133.
[17] S. Palaniappan, A. Sawheny, H. S. Sarjoughian, "Application of DEVS Framework in Construction Simulation," in Proc. of Winter Simulation Conference, Monterey, CA, 2006.
[18] K. Jensen, "Coloured Petri Nets: A High Level Language for System Design and Analysis," Lecture Notes in Computer Science, Vol. 483 Springer-Verlag, 1990, pp. 342-416.
[19] M. Abdollahi Azgomi and A. Movaghar, "Towards an Object-Oriented Extension for Stochastic Activity Networks," in Proc. of the 10th Workshop on Algorithms and Tools for Petri Nets (AWPN'03), Eichstaett, Germany, Sept. 26-27, 2003, pp. 144-155.
[20] "OCL 2.0 Specification Version 2.0," The Object Management Group, 2005.
[21] E. Kindler, "High-level Petri Nets–Transfer Format," Working Draft of the International Standard ISO/IEC 15909, Part 2, University of Paderborn, 2005.
[22] "Java EE at a Glance," Available: http://java.sun.com/javaee/.

[23] S. Burbeck, Applications Programming in Smalltalk-80 (TM): How to Use Model-View-Controller, 1992.

Hamid Mohammad Gholizadeh received the B.Sc. degree in computer engineering (software) from Azad University of Tabriz (2004) and M.Sc. degree in information technology (design and development of software) from Iran University of Science and Technology (June 2008). Title of his M.Sc. thesis was "A New Framework Based on Petri Nets and Related Formalisms for Modeling and Analysis of Systems," which has been done under supervision of Dr. Abdollahi Azgomi.

Mr. Gholizadeh's professional activities include designing and developing enterprise applications based on Java EE framework and his research interests are in the area of software engineering, formal modeling and web technologies. He has published several papers in International conferences.

Mohammad Abdollahi Azgomi received the B.Sc., M.Sc. and Ph.D. degrees in computer engineering (software) (1991, 1996 and 2005, respectively) from Sharif University of Technology, Tehran, Iran. His research interests include performance and dependability modelling with high-level modelling formalisms such as stochastic Petri nets, tools for modelling and evaluation, verification and validation, object-oriented modelling, web services, network and web security. He has published several papers in International journals and conferences.

Dr. Abdollahi Azgomi is currently an assistant professor at the school of computer engineering, Iran University of Science and Technology, Tehran, Iran.