THE OsMoSys/DrawNET Xe! LANGUAGES SYSTEM: A NOVEL INFRASTRUCTURE FOR MULTI-FORMALISM OBJECT-ORIENTED MODELLING

Marco Gribaudo
Dipartimento di Informatica

Università di Torino
C.so Svizzera 185
marcog@di.unito.it

Mauro Iacono
Nicola Mazzocca
Dipartimento di Ingegneria dell'Informazione
Seconda Università di Napoli
via Roma 29 Aversa (CE) Italy
{mauro.iacono, nicola.mazzocca}@unina2.it

Valeria Vittorini
Dipartimento di Informatica e Sistemistica
Università di Napoli "Federico II"
via Claudio 21 Napoli Italy
vittorin@unina.it

KEYWORDS
Modelling languages, multi-formalism, object orientation, XML.

ABSTRACT

Complex systems present a big challenge to the modeller: different subsystems need different modelling techniques, because they have different purposes and different kind of specifications. A multi-formalism modelling methodology can be useful to unify the different aspects of a model. In this paper we propose a system of languages which constitutes the foundation of OsMoSys, a multi-formalism, multi-solution, object-oriented modelling framework. The presented languages system supports automatic generation of GUI for every modelling formalism through the integration of the DrawNET Xe! tool, as well as OO submodel reuse and inheritance, flexible multi-solver solution and result analysis through the OsMoSys solving architecture.

INTRODUCTION

Modelling complex systems is a challenge for designers. Several modelling techniques have been developed and can be found in literature. These techniques are able to pursue performance prediction, verification or analysis of systems in both the design phase and validation phase. Complexity (in terms of number of subsystems, or behaviours, or heterogeneity of subsystems) exasperates the task of modellers, because different subsystems must meet different kinds of requirements, either functional (e.g. correctness) or non-functional (e.g. dependability), or both. Different submodels need to be developed in order to design and verify the whole system, with the consequent growth of the number of available system views. These views, each one independent from the others, should be kept synchronized during the design cycle of the system. In addition, different models, written in different formal languages, can not automatically share parameters or exchange results, because they have not been thought to interoperate. In order to obtain a comprehensive, flexible, interoperable, composeable modelling technique, different languages/techniques must be integrated in an unique canvas.

Syntactical integration between formal languages is the first step on the path to a multi-formalism, multi-solver modelling technique. Every formal modelling technique is founded on a formal language. Syntactical integration relies on the definition of common rules behind different grammars which define different formal languages. A second step is semantic integration. Semantic integration defines the meaning of inter-model communication and parameter exchange, and, at the best of our knowledge, should be examined case by case, with some exceptions for formal languages that have common roots (e.g., various kind of Petri Nets evolutions share the basic elements of the languages). Semantic integration also introduces the problem of describing (and retrieving, as a side effect) the results of the evaluation of a submodel, because interoperation between models includes hierarchical composition or dependences in the model evaluation process. The inclusion (or the dependence) of a submodel in (by) another one, written in a different formal language, usually requires the evaluation of some parameters of one component and the transformation of these values into parameters for the other submodel. Automating this evaluation process requires the definition of additional formal languages suitable for the description of results and for the specification of a solution process.

This paper is part of a more complex research project whose purpose is the development of an integrated framework for multi-formalism, multi-solution modelling. Some research results have already been published on this topic and on the methodology behind the framework in literature (Gribaudo and Valente 2000b, Gribaudo and Sessi 2001, Vittorini et al. 2002, Franceschinis et al. 2002a, Franceschinis et al. 2002b, Gribaudo and Valente 2000a, Vittorini et al., Baravalle et al. 2003), as well as applications (Franceschinis et al. 2003, Gilmore and Gribaudo 2003). In this paper we show how the OsMoSys/DrawNET Xe! (OsMoSys in the following) languages family allows the designer to easily develop multi-formalism, object oriented models

Proceedings 15th European Simulation Symposium
Alexander Verbraeck, Vlatka Hlupic (Eds.)

ISBN 3-936150-28-1 (book) / 3-936150-29-X (CD)

together with a general specification of a complete multi-solver resolution process, from the graphical specification of models to the extraction of the results after their evaluation. In this work we will only describe the syntactic integration and resolution process specification: the semantic integration is out of the scope of this paper.
In the following section we will describe the OsMoSys methodology and the overall features of the OsMoSys languages family, their integration in the framework and in the DrawNET Xe! tool. The general organization in levels and layers is then presented, followed by an in-depth analysis of both the syntactic integration facilities and the modelling paradigm; then a brief description of the OsMoSys architecture is given, to show how the results definition and retrieval languages are used together with the resolution process definition languages.

OsMoSys OVERVIEW

The OsMoSys methodology is the result of a collaboration between research groups at the *Univ. di Napoli* and the *Dip. di Informatica* of the *Univ. del Piemonte Orientale* on the themes of compositional model construction with sub-model reuse and multi-formalism modelling. The methodology is the evolving result of the work done by the two research groups to define formalisms integration and composition strategies and to develop an integration strategy for external solvers. In particular, ongoing research is aimed at applying workflow principles to cope with the problem of multi-solution when analysing/simulating multi-formalism models. A principal role in the work done is played by DrawNET, a multi-formalism GUI developed at the *Dip. di Informatica* of the *Univ. di Torino*. The third version of DrawNET (namely DrawNET Xe!) is currently under release. Its first release is at the origin of the first version of the languages family now used to support the OsMoSys methodology. The evolution of the two projects is now tightly connected because DrawNET Xe! has been adopted as the presentation level of the OsMoSys framework, which in turn is the implementation of the ideas of the OsMoSys methodology.
The principal guideline of the modelling paradigm in OsMoSys is flexibility: submodels in a model can belong to different formalisms and must speak to each other in order to allow the modeller to exploit his/her proficiency to cope at its best with any different problem in the design phase. In addition, the model can be obtained by composition of submodels and submodels can be reused and refined. In order to achieve this goal, the OsMoSys modelling languages family implements some concepts of Object Oriented Development (OOD), like classes, inheritance, aggregation and instances. Submodels in OsMoSys are classes, which can be developed, stored in a class library and instantiated to build models. Non instantiated submodels are named Model Classes (MCs) in the OsMoSys terminology. MCs can be obtained by scratch, building them with a proper formal language, suitable for the evaluation process they are designed for. New MCs can be also derived from existing classes or can be assembled by aggregation (and partial instantiation) of existing classes. A model is finally a fully instantiated class. All these ideas are supported by DrawNET Xe!: in Figure 1, the tool screenshot shows a model together with a component submodel, interface and parameter definition facilities in order to build model classes, the model class library and the instantiation of a class through parameter specification.
Going behind the modelling level, model classes must be written following a proper syntax. This syntax belongs to a proper formal language suitable for some kind of analysis, like Petri Nets (PN) and extensions, or Fault Trees (FT) and extensions et cetera. A syntax is defined by a grammar. In the OsMoSys terminology a grammar is named Model Metaclass (MM). The word *metaclass* defines the two main characteristics of it: as first, it's a class, so it can be obtained by derivation from another metaclass; besides, it is a meta-description, that is a description of a description (a MC is the description of an instance). This technique, known as *meta-modelling*, allows the definition of formal languages and of their syntactical interaction to develop the foundation of multi-formalism models.
These issues were already implemented in the OsMoSys framework (Vittorini et al.) before the introduction of the new languages family presented in this paper (and obviously in DrawNET++, the predecessor of DrawNET Xe!): but the new family exceeds the limitations of its predecessor and completely redefines the modelling and meta-modeling languages improving their clearness and their expression power, as well as DrawNET Xe! is a totally new tool rebuilt on the new languages paradigm.
Brand new features included in the new versions are the ability of describing and retrieving results in the solving architecture, and to show them through the GUI. Three different problems have been solved through the definition of proper languages for results specification: first, a language has been introduced in the family to describe the results themselves; another language has been introduced to define queries that can be sent to the solving subsystem in order to produce results; a third language has been defined to return the values of the results back to the GUI. In addition, proper languages have been introduced to support the specification and the enactment of solution strategies.

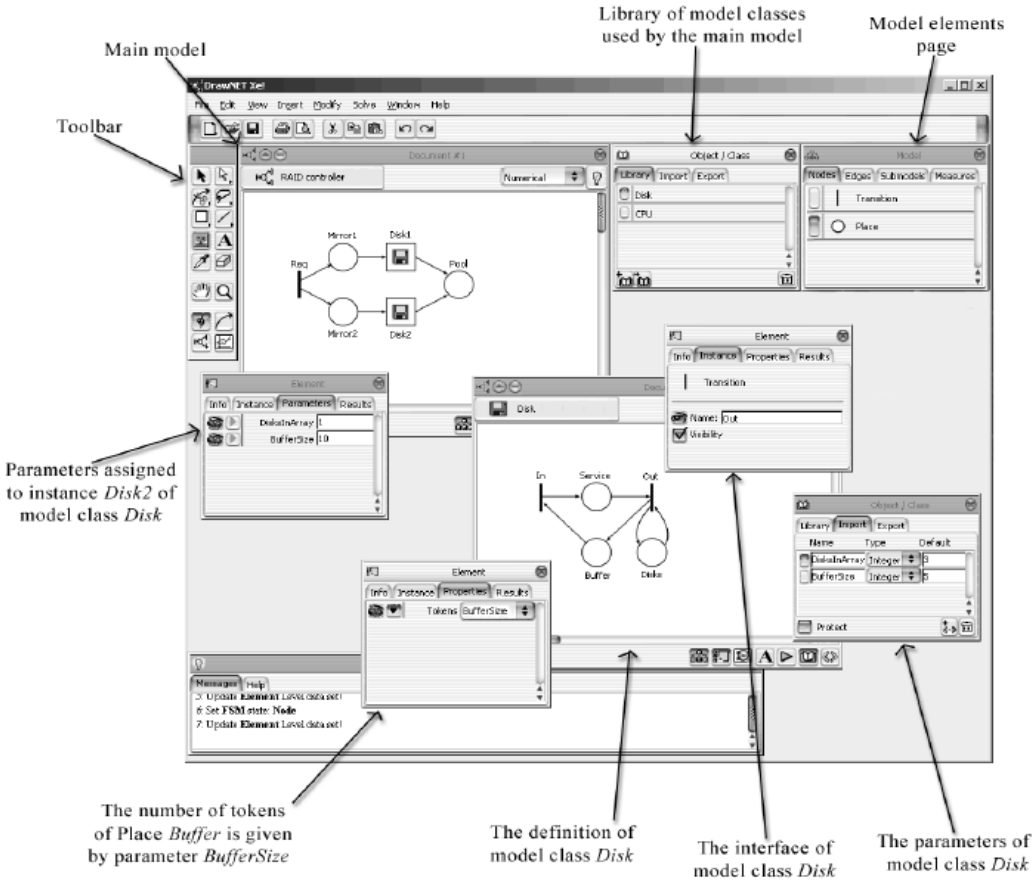

Figure 1: the DrawNET Xe! interface with visual representation of model classes and the library

LAYERS AND LEVELS

A more in-depth view on the OsMoSys modelling technique requires a description of the conceptual structure of an OsMoSys model. An OsMoSys model can be viewed by two different points: from the modeller's one or from the structural one.

The modeller can consider an OsMoSys model as the result of the coexistence of three different layers: the conceptual graph layer, the visual representation layer and the solved model layer. The structural view of models is orthogonal to the layers view and consists of two levels: the description level and the metadescription level.

OsMoSys layers

The conceptual graph layer is the core part of a model and constitutes its abstract representation. A synthetic view of the languages used in this layer is in Figure 2, which will be better examined in the following. This layer contains all the structural and quantitative information that form the model itself. OsMoSys has been designed and exploited for formal languages which models can be represented by a graph. This policy, the only available in the first version of the framework languages family, is adopted because of the presence, as a main model development tool, of the DrawNET Xe! user interface. The DrawNET Xe! user interface in fact implements syntactical validation on models and aids the modeller in many tasks, as choosing between already allowed multi-formalism techniques as well as selecting the most appropriate multi-solver analysis. This choice is not a limitation for the expression power of the framework, because it is possible to design and implement graph-based representations for usually text-based formal languages (Gilmore and Gribaudo 2003).

A model is in the conceptual graph layer a graph of (nested) graphs: every submodel is represented by a graph itself and submodels can be nested by aggregation with no limitations in the depth level. The nodes of inner graphs are constituted by atomic components of a formal language (integrated into the framework). The OsMoSys implementation of a formal language will be named *formalism* in the following to better highlight the context. At each level, a graph is a submodel expressed in a single formalism. Each graph/submodel is included in another graph/submodel or connected to other graphs/submodels through a special kind of arc: these graphs/submodels can be written in the same or different formalisms. A closer view to this organization will be given in the next section.

The visual representation layer is in charge of the visual presentation of a model: each element of each formalisms has a graphical version, able to give the user an iconic view of itself. The graphical representation is used by DrawNET Xe! in order to visualize a model and all its parts (Figure 1). The visual description of a model is kept separated from the model abstract description. This is done for two reasons: first, the two different

layers are independent representations of the same reality, both needed for different and disjoint purposes; second, while the visual representation layer is only exploited by DrawNET Xe!, conceptual graph layer is a bridge between DrawNET Xe! and the OsMoSys modelling framework and carries only the information required to store, to solve and to evaluate the model. A useful consequence of this organization is the independence of models from their representation (which can be modified whenever needed).

The solved model layer enriches the model with the results of its evaluation. A synthetic view of the languages used in this layer is in Figure 4, which is commented in the following. This layer contains an abstract description of the results of the pursued analysis. This description is flexible enough to incorporate structured information from any kind of (unknown when a model is written) solving process the model is sent to. This layer is responsible of:

- defining which kind of analysis can be pursued on a certain multi-formalism model;
- defining the structure of the analysis process for every allowed analysis, in terms of algorithms and solvers;
- defining how the allowed analysis can be communicated to the solving subsystem by the DrawNET Xe! user interface;
- defining the format of the results;
- describing the results.

These functionalities constitute a bouquet of services which is the warp of the solving subsystem of OsMoSys. The weft, as we will show later, is constituted by several software objects properly driven and coordinated.

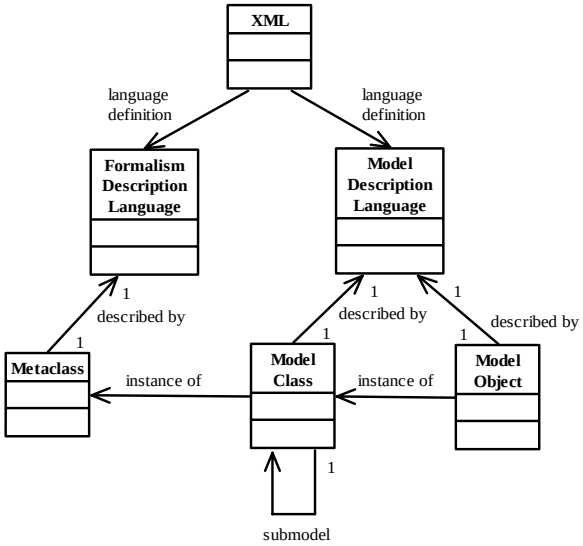

Figure 2: the UML class diagram of the language system for conceptual graph layer

OsMoSys levels

The structural view of models is composed of two levels: the description level and the metadescription level. These levels are present in each layer of the OsMoSys model layer organization. The description level is the part of the model the user can see, directly or by the DrawNET Xe! interface, including the conceptual graph, the visual representation, the result presentation and the model results queries. All these information are written following proper formalisms (modelling formalisms, visual formalisms, result structuring formalisms and model query languages, respectively). These informations are instances of the formalisms in which they are represented. One of the strengths of OsMoSys since the first version, is that those information are expressed in a language whose keyword are specified by an upper level specification. This composed description of the model is supported by the metadescription level, which defines the formalism grammars and the facilities to implement syntactical connections between submodels written in different formalisms. Figure 2 and Figure 4 respectively show the UML description of the set of languages used for models definition and results definition: the relationship between languages and other elements will be detailed in the following. For example, with reference to Figure 2 and Figure 4, the Formalism Description Language (FDL) is used to specify model metaclasses (formalism grammars), which allows writing model classes (submodels) by the Model Description Language (MDL); the Model Query Language (MQL) is used to write queries to be evaluated on models, with the support of results specifications, written by the Request Definition Language (RDL). This two-levels architecture gives OsMoSys the possibility of completely customizing the modelling formalisms and to express and retrieve results.

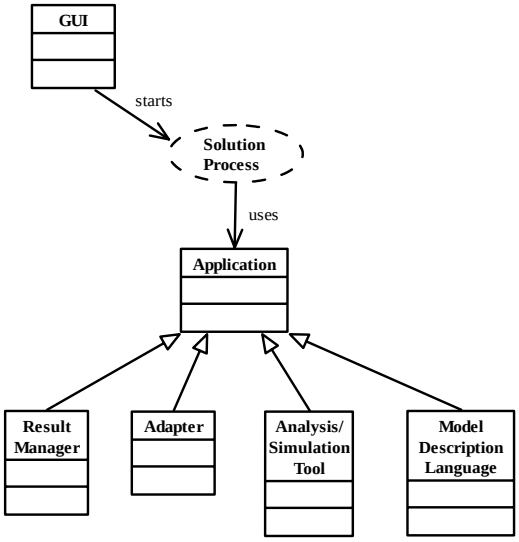

Figure 3: the UML class diagram of the components participating in a solving process

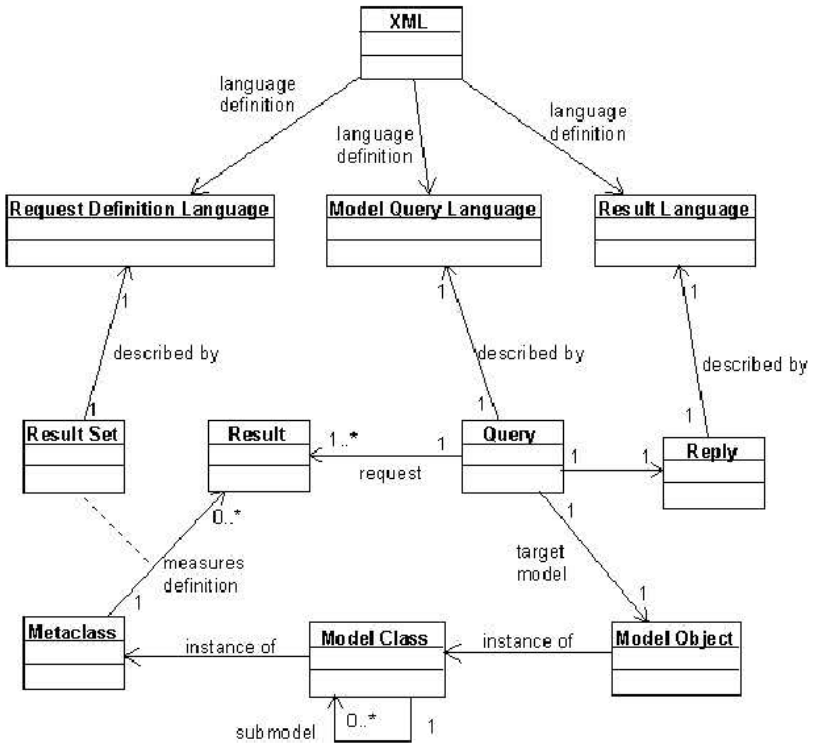

Figure 4: the UML class diagram of the language system for solved model layer

MODELLING FORMAL LANGUAGES

According to the structural view, we will in brief describe the family of languages of OsMoSys by levels. All languages are based on XML because of its flexibility, portability and universality, as shown in Figure 2 and Figure 4. Three languages are used in the metadescription level of OsMoSys: they are the Formalism Description Language (FDL), the Result Description Language (RDL) and the Formalism Representation Language (FRL), each one used in a different layer of the modelling paradigm.

The FDL is the foundation of the conceptual graph layer. The FDL is used to specify formalism syntax for both the definition of models and the automatic customisation of the DrawNET Xe! interface. FDL is a brand new evolution of the previous version, already existing in OsMoSys/DrawNET++ with all the main features but completely redesigned and redefined. The FDL allows a formalism designer to describe all the elements of a formalism in terms of nodes (*elements* in OsMoSys), edges (*arcs* in OsMoSys) and *constraints*. The FDL grammar itself is defined by an XML dtd in order to keep coherence in the framework.

The central point of FDL is the element. An element is defined by an elementType definition and is characterized by properties (which are now typed in the new version). An element can contain other elements, references to other elementTypes statements or elementPointers to other elements. References can be used to introduce type restrictions in the syntax of a model: elementPointers, as well as common software pointers, are handles which reference other elements in the model and in addition are able to add new properties to the target. Pointers are a new extension to OsMoSys, as well as references. Constraints define limitation on other elements in the formalism: now they are independent entities while in OsMoSys they were owned by other elements. Constraints have been enhanced: while in the previous version they only were able to limit the kind and/or the number of elementTypes participating in a relationship, they can also now express checks or include mathematical/boolean expressions to be evaluated on the constrained elements.

The FDL is object oriented. Every element can be inherited from an existing one and can extend it with new properties or new constraints. Elements can be defined concrete or abstract and visibility qualifiers can be used in models. Constraints can be defined as valid on the original elementType or on it and every derived elementType. FDL definitions (formalisms) can inherit one from another as well, by adding new elements or constraints and by inclusion of other formalisms. Inclusion is another new feature now available at FDL level. The following is an example of the FDL definition of Petri Nets (PN).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE fdl SYSTEM "fdl.dtd">
<fdl main="PN">
  <include src="base/GraphBased.fdl" />
  <include src="base/Instantiable.fdl" />
  <elementType name="PN" type="private">
    <parent ref="GraphBased" />
    <parent ref="Instantiable" />
    <elementType name="Place">
      <parent ref="Node" />
```

```
    <propertyType name="Tokens" type="integer" default="0" />
   </elementType>
   <elementType name="Transition">
    <parent ref="Node" />
    <propertyType name="Weigth" type="float" default="1.0" />
    <propertyType name="Priority" type="integer" default="1" />
   </elementType>
   <elementType name="Arc">
    <parent ref="Edge" />
    <propertyType name="Weight" type="integer" default="1" />
    <constraint>
     <check op="isOfKind" ref="from" kind="Transition" />
     <check op="isOfKind" ref="to" kind="Place" />
    </constraint>
    <constraint>
     <check op="isOfKind" ref="from" kind="Place" />
     <check op="isOfKind" ref="to" kind="Transition" />
    </constraint>
   </elementType>
   <elementType name="InhibitorArc">
    <parent ref="Edge" />
    <propertyType name="Weight" type="integer" default="1" />
    <constraint>
     <check op="isOfKind" ref="from" kind="Place" />
     <check op="isOfKind" ref="to" kind="Transition" />
    </constraint>
   </elementType>
   <elementTypeRef ref="PN" />
  </elementType>
</fdl>
```

As seen, FDL definitions are classes themselves. Two special abstract FDL definitions actually form the kernel of our formalism library: GraphBased.fdl and Instantiable.fdl. The first one defines (exploiting the FDL language itself) an abstract graph formalism to be extended by concrete formalisms. The second one defines a syntax to implement (exploiting the FDL language itself) instantiability of submodels within other submodels: it defines an *interface* and a *parameter* elements, as well as an *use* construct, which allows derived formalisms to export and import elements when instantiating models and to include submodels.

The RDL is used to define feasible results from the analysis of models of a certain formalism. RDL is a new feature of OsMoSys. RDL is similar to FDL: it allows to define elements, which have resultTypes that describe the quantities a result query can refer to. ResultTypes are typed and can be structured or contain other resultTypes. An RDL formalism should be designed with respect to the solution engine able to obtain the desired results from the evaluation of a model: special hints can be added to resultType definitions in order to modify the solver evaluation process evolution (e.g., to obtain optimisations). RDL is object oriented too: inheritance and aggregation are supported as well as in FDL. The following is a simple example of the RDL definition of Petri Nets (PN).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE rdl SYSTEM "rdl.dtd">
<rdl main="PN">
 <elementType name="PN">
  <elementType name="Place">
   <resultType name="MeanTokens"/>
  </elementType>
  <elementType name="Transition">
   <resultType name="Trhoughput" defaultCompute="yes"/>
  </elementType>
  <elementType name="Arc">
   <resultType name="TokenFlow"/>
  </elementType>
  <aggregateType name="MeanValue">
   <propertyType name="Expression" type="string" default="" />
   <resultType name="Mean"/>
  </aggregateType>
 </elementType>
</rdl>
```

The FRL is used to describe the visual representation of the elements of the formalism. FRL is a new feature of OsMoSys. It associates a set of graphical representations to the element of a formalism. The language used to describe the appearance of each element is a subset of SVG (Scalable Vector Graphic, a w3 standard for vector graphic representation (Eisenberg 2002)). Each element may have more than one graphical representation: the used one is chosen depending on the property of the associated element. In this way, for example, a PN place may have different representations depending on the property that holds the number of tokens: each representation adds to the circle commonly used to visualize a place, a number of small filled circles corresponding to the marking.

The fact of having the visualization appearance of a formalism in separate layer allows the possibility to have several different representation for a single formalism: a FRL document can be thought as a style sheet for the corresponding Model Representation Language (MRL) document. This can be useful for example for formalisms for which more than a single standard representation exists. For example for Fluid Stochastic Petri Nets two different representation exist: one in which the arcs used to transfer fluid are represented as bold arrows (Ciardo et al. 1999), and another where they are represented as double arrows (Horton et al. 1998).

MODELLING MODELS AND RESULTS

As in the metadescription level, three languages are used in the description level of OsMoSys: they are the Model Description Language (MDL), the Model Query Language (MQL) and the Model Representation Language (MRL), each one used in a different layer of the modelling paradigm.

The MDL is used to represent the conceptual graph layer. MDL models are automatically generated by DrawNET Xe! from the graphical inputs of the modeller (Figure 1). MDL is used for the definition of both model classes and model instances. A MDL document refers to a FDL description that states the admitted model elements and eventually some additional characteristics, like the instantiability feature. In the following we will assume the general case of instantiable graph-based models. Although object oriented capability (inheritance, aggregation, information hiding) were already present in the previous version, a completely new approach has been adopted in OsMoSys for three reasons: to support class libraries, to enhance the instantiation mechanism and to improve the information

hiding features. An MDL model defines now the structure of a model class, e.g. the various elements present for each elementType defined in the corresponding FDL, their connection and the included (sub)model objects, if any. In the last case, references to their description are included pointing to the same document or to external documents, e.g. from a MDL library. The instantiation of included model classes is pursued through the instantiation of their interface and parameters or a further export of them. A model class can also export some parameters and some interfaces: a completely instantiated model class is a model instance (the main model). Included classes are not necessarily based on the same FDL, thus implementing multi-formalism. The definition of inter-formalism connections should be defined in the container model class FDL.
The following is a simple example of the MDL definition of a PN model class including two nested classes, one internal and one external, and exports a parameter and an interface. Notice that PN.fdl should include Instantiable.fdl in order to include and instantiate model classes.

```
<mdl fdl="PN.fdl" main="Net1">
  <PN name="PNClass">
      <interface add="#T0"/>
      <parameter name="K" default="1">
            <assign obj="#P0" property="token"/>
      </parameter>
      <Transition name="T0" rate="1.0"/>
      <Place name="P0" token="2"/>
  </PN>
  <PN name="Net1">
    <Place name="P0" token="1"/>
    <use class="#PNClass" name="Inst1">
      <parameter name="K" value="7"/>
    </use>
    <arc name="arc0" from="P0" to="Inst1.T0"/>
    <!-- Includes an extern object -->
    <use class="library.mdl#MachineClass" name="Inst2">
      <parameter name="Speed" value="1.0">
    </use>
    <!-- Defers K and exports Inst1.T0 from the nested object -->
    <interface add="#Inst1.T0"/>
    <parameter name="K" default="1">
        <assign obj="#Inst1.K" property="value"/>
    </parameter>
  </PN>
</mdl>
```

The MQL is used to define queries to be solved by the solution subsystem. An MQL document refers to a RDL document which defines the feasible results the MQL can ask for. Aggregate measures can be included, as well as derived measures through calculations.
The following is a simple example of a MQL query for a PN model.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE fdl SYSTEM "mql.dtd">
<mql rdlref="PN-GreatSPN.rdl">
   <element name="APetriNet" type="PN">
    <result name="HasPSemiFlow"/>
    <element name="P0" type="Place">
      <result name="MeanTokenNum"/>
    </element>
    <element name="P1" type="Place">
      <result name="MeanTokenNum"/>
      <result name="MeanTokenDistrib"/>
    </element>
    <element name="T0" type="Transition">
      <result name="Trhoughput"/>
    </element>
    <aggregate name="M1" type="Mean">
            <property name="Expression" val="P0+P1" />
      <result name="Value"/>
    </aggregate>
    <aggregate name="M2" type="Mean">
            <property name="Expression" val="T0 * P0 / (P0+P1)" />
      <result name="Value"/>
    </aggregate>
  </element>
</mql>
```

The MRL is used to describe the visual representation of a specific model expressed in a given formalism. In order to make the tool more extensible and allow a deeper integration in a graphic environment, MRL files are *SVG compatible*. MRL are structured in a way that any SVG viewer (that are currently integrated in most web browser and available for most platforms) can show the model they describe. In this way the models drawn in DrawNET can be immediately imported and printed by most existing graphical editing packages. The MRL files do not only hold the information regarding the representation of the models, but also all the information required by the GUI to define the visual appearance of a model, such as layers, texts and the other features implemented in the interface.
MRL files are connected to their corresponding MDL file using the ID attribute of the various SVG primitives that composes the MRL description. Using a special syntax, each ID of a graphic primitive that represents a model element, can be used as a pointer to connect it with the corresponding element in the MDL file.

LANGUAGES TO SUPPORT THE SOLVING ARCHITECTURE

The semantic level of OsMoSys also requires the support of proper languages in order to let the modeller take advantage of the multi-solving architecture. The semantic level is in charge of giving a meaning to the syntactic interactions between heterogeneous communicating submodels. The semantic level is implemented in the OsMoSys solving architecture, which is a system of software objects formed by many components. Figure 5 shows a general description of the software architecture of the solving subsystem. We will not investigate in depth this software architecture, but we will give a view on it in order to introduce the remaining languages of the OsMoSys family. Figure 5 shows a three-level architecture: a client interface, representing in this paper the DrawNET Xe! interface; a workflow management level, which implements the logic of the solution process; and a components level, in which several components can be found. The workflow management is under the control of a workflow engine: it uses a process repository to store the known solving processes, written in another XML-based language

which description is out of the aims of this paper. The available components (see Figure 3 for an UML description), coordinated by the workflow engine, are:

- adapters, used to interface OsMoSys with legacy solvers;
- external solvers, which are used to obtain partial evaluations on the model;
- pre-postprocessors, which implement the semantic layer of OsMoSys;
- the request manager, which is responsible of processing MQL documents and producing answers.

The client level supplies the substanding architecture with a MDL document and a MQL document and waits for results to show them back to the user. In order to compute a solution, more information should be supplied by the user: which is the preferred solution process, which documents are involved in the description of the model and which results must be computed for that model. A proper database structure has been developed to manage the set of XML-based languages.

The set of formalisms that the DrawNET GUI may have to cope with can grow quite rapidly. Since every formalism is described by several file (at least an FDL, FRL and RDL, but it may also have several different results definitions and graphical representations), it is clear that the number of files that must be organized can be quite large. The database structure helps with the formalism files organization. This database is also stored in a XML file and can be accessed by the GUI and by the various solution components to locate the required resources.

The GUI uses this database to present to the user a list of formalism from which he may choose which modelling language / solution component he may use to describe his models. The database is also accessed by the solution components to find the formalism definition files (which may be required to compute the solution of the model) and to find parameters that may be required to guide the solution process.

The database is composed by four tables: *formalisms*, *results*, *styles* and *solution processes*.

The *formalism* table holds the list of the available formalisms. For each formalisms it holds a textual description of its name, and the URL of the corresponding FDL file. The *results* table associates to each formalism a set of solvers. This is required since, has we have pointed out before, each formalism may have more than solver that can handle it, and each solver may be able to compute different results. Each row of this table contains the URL of the corresponding RDL file and a textual description of the solver. The *styles* table is parallel to the results table, in the sense that it holds the definitions of the visualization style that can be associated to a formalism. Each row of this table holds the textual description of the style and the URL of the corresponding FRL file. We must point out that in most of the cases, both the results and the styles table will have just one entry for each formalism. The *solution processes* table holds several entries for each formalism-result pair. Each solver can be able to perform different solutions, using different techniques. For example a Petri Net steady state solution can be computed either using analytical techniques or by simulation. Each row of the solution process table describes a specific solution. It contains a textual description of the solution, an URL for the process that carries out the solution, and a set of parameters. Those parameters are passed to the solver process, together with the MDL and MQL files when that particular solution technique is requested. The GUI uses this table to present the user the list of the possible solutions that can be invoked for a specific formalism/solver pair. Figure 6 shows the ER diagram for the proposed database.

Once the desired solution process has been selected and sent to the workflow management level, documents are routed from one tool to another by the engine, as well as the components of the solving subsystem are activated in turn as needed. After the solution process computes the results, they have to be conveniently represented and sent back to the DrawNET Xe! interface. For this purpose another language has been designed, namely ReSult Language (RSL). The RSL is designed to be a bridge between the solving subsystem, the result definition mechanism and the result presentation logic of DrawNET Xe!: the organization of this document reflects the presentation needs. Results are described by model element name as well as by the measure type and by the graphical organization: they can be organized in frames, in order to show temporal series, as well as in single values to be showed near the corresponding element or in series, to obtain plottable curves.

The following is a simple example of a RSL document for a PN model.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE fdl SYSTEM "mql.dtd">
<rsl rdlref="PN-GreatSPN.rdl">
 <frame label="Control" base="true">
  <element name="ReteDiPetriDiProva" type="PN">
   <result name="HasPSemiFlow" value="yes"/>
   <element name="P0" type="Place">
    <result name="MeanTokenNum">
     <value val="3.7777"/>
    </result>
   </element>
   <element name="P1" type="Place">
    <result name="MeanTokenNum" format="single">
     <value val="3.7777" />
    </result>
    <result name="MeanTokenDistrib" format="table">
     <value index="1.0" val="3.7777" />
     <value index="1.1" val="4.7777" />
     <value index="1.2" val="5.7777" />
     <value index="1.3" val="4.7777" />
    </result>
   </element>
   <element name="T0" type="Transition">
    <result name="Trhoughput">
     <value val="237.34"/>
    </result>
   </element>
   <aggregate name="M1" type="Mean">
    <result name="Value">
```

```
      <value val="46"/>
    </result>
   </aggregate>
   <aggregate name="M2" type="Mean">
    <result name="Value">
     <value val="0.000004534"/>
    </result>
   </aggregate>
  </element>
</frame>
```

```
</mrl>
```

The RSL is thus the final brick of the languages system and close the circular data path from/to DrawNET Xe!. As a matter of fact, it can be considered another description level built on the metadescription level of the RDL, as well as MQL (see Figure 4).

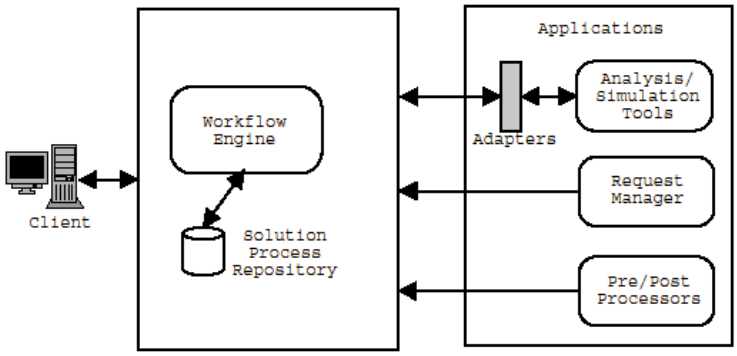

Figure 5: the structure of the OsMoSys solving architecture

CONCLUSIONS

In this paper we analysed the language infrastructure used in the OsMoSys framework to specify the various different aspects of a model. We showed how a multi-formalism paradigm can be supported by exploiting object oriented developing techniques through a family of description languages enforced by a metadescription level. The metamodelling approach allows the framework to be flexible and easily extensible and gives it a canvas to glue together the conceptual modelling view, the graphical representation view and the solution process view. Moreover, the layer/level structure of the languages family gives a comprehensive organization to the framework. The intrinsic graph structure for the models is showed to be not a limitation for the expression power of the OsMoSys framework: on the other hand, it allows an easy hierarchical representation of models.

The modelling framework is supported as well by a GUI interface as by an extensible and open solution subsystem: the latter itself is parameterised and driven by proper languages in the framework and the solution process itself is described by a special purpose language.

XML is behind all the language systems: the modelling language implements advanced syntax constructs, like pointers or class parameterisation, thanks to the flexibility of XML. The language system is an infrastructure which connects the various levels and elements of the whole framework and constitutes its foundations.

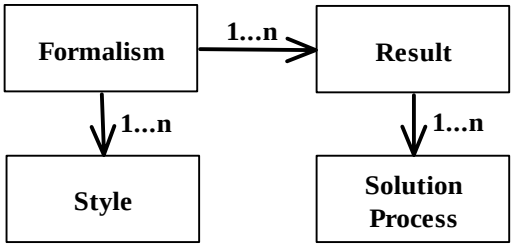

Figure 6: the ER diagram of the formalism database

ACKNOWLEDGEMENTS

This paper is partially supported by Centro Regionale di Competenza sulle Tecnologie dell'Informazione e della Comunicazione of Regione Campania.

REFERENCES

Baravalle, A., Franceschinis G., Gribaudo , M., Lanfranchi, V., Iacono , M., Mazzocca, N., Vittorini, V. 2003. "DrawNET Xe: GUI and Formalism Definition Language". To be published in *Proceedings of Performance TOOLS 2003 Conference, part of the 2003 Illinois International Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems* (Urbana, Illinois, USA Sep. 2-5).

Ciardo, G., Nicol, D. M., Trivedi, K. S. 1999. "Discrete-event Simulation of Fluid Stochastic Petri Nets" *IEEE Transactions on Software Engineering* 25, Vol.2, 207-217.

Eisenberg, J. D. 2002. *SVG essentials*, O'Reilly.

Franceschinis, G., Gribaudo, M., Iacono, M., Mazzocca, N., Vittorini, V. 2002. "DrawNET++: Model Objects to Support Performance Analysis and Simulation of Complex Systems". In *Lecture Notes in Computer Science 2324, Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and*

Communication System Performance Evaluation (TOOLS'02). Springer-Verlag, London, 233–238.

Franceschinis, G., Gribaudo, M., Iacono, M., Mazzocca, N., Vittorini, V. 2002. "Towards an Object Based Multi-Formalism Multi-Solution Modeling Approach". In *Proceedings of the Second Workshop on Modelling of Objects, Components and Agents (MOCA'02)* (Aarhus, DK, Aug. 26-27). 47–65.

Franceschinis, G., Marrone, S., Mazzocca, N., Vittorini, V. 2003. "SWN Client-Server Composition Operators in the OsMoSys framework". In *Proceedings of 10th Int. Workshop on Petri Nets and Performance Models (PNPM'03)* (IL, USA, Sep.). IEEE Soc. Press.

Gilmore, S. and Gribaudo, M. 2003. "Graphical Modelling of Process Algebras with DrawNET". In *Tools presentation at the Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems* (IL, USA, Sep.).

Gribaudo, M., Valente, A. 2000. "Framework for Graph-based Formalisms". In *Proceedings of the 1st International Conference on Software Engineering Applied to Networking and Parallel Distributed Computing (SNPD'00)* (Reims, France, May). 233–236.

Gribaudo, M. , Valente, A. 2000. "Two levels interchange format in XML for Petri Nets and other graph-based formalisms". In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets* (June), 22-29.

Gribaudo, M. , Sessi, D. 2001. "A Multiparadigm Simulation Framework". In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'01)* (June), 1647-1653.

Gribaudo, M. 2001. "FSPNEdit: a Fluid Stochastic Petri Net Modeling and Analysis Tool". In *Proceedings of Tools of 2001 International Conference on Measuring, Modeling and Evaluation of Computer and Communication Systems* (Aachen, Sep.). 24-28.

Horton, G., Kulkarni, V. G., Nicol, D. M., Trivedi, K. S. 1998. "Fluid stochastic Petri Nets: Theory, Application, and Solution Techniques" *European Journal of Operations Research* 105, No.1 (Feb), 184-201.

Vittorini, V., Franceschinis, G., Gribaudo, M., Iacono, M., Bertoncello, C. 2002. "DrawNET++: a Flexible Framework for Building Dependability Models". In *Proceedings of Tools presentations, Proc. of the International Conference on Dependable Systems and Networks (DSN'02)* (June).

Vittorini, V., Franceschinis, G., Iacono, M., Mazzocca, N.. "OsMoSys: a new approach to multi-formalism modeling of systems". To be published in *SoSyM, Journal on Software and System Modeling*, Springer.

AUTHOR BIOGRAPHIES

MARCO GRIBAUDO has currently a researcher position in the University of Torino, where he teaches Computer Graphics at the course of Arts, Music and Cinema. He obtained his PhD from the same university in 2002, with a thesis on Hybrid System for performance evaluation. His current research interests are: performance evaluation using hybrid formalisms, tool development and different formalism integration, communication networks and multimedia systems, computer graphic and virtual realty applied to learning.

MAURO IACONO is adjunct professor and research assistant at the Second University of Naples (SUN). He obtained his Laurea degree in Ingegneria Informatica in 1999 from the University of Naples "Federico II". After that, he joined the SUN where he received his PhD in Electronic Engineering in 2002. His current research interests are: performance evaluation by multi-formalism modelling techniques, complex systems and reactive systems engineering. He is a consultant of the Italian Ministry for Innovation and Technology (MIT) within the e-government national plan.

NICOLA MAZZOCCA is full professor of Calcolatori Elettronici at the Second University of Naples (SUN). He graduated in electronic engineering from the University of Naples, Italy, in 1987, and received his PhD from the same university. His scientific activity involves methodologies and tools for performance evaluation of computing systems, computer networks, communication protocols, general and special purpose parallel architectures and applications. Since 1998 he participated in various research projects as coordinator.

VALERIA VITTORINI is assistant professor at the Department of Computer Science and Systems of the University of Naples "Federico II", Italy. She graduated in Mathematics at the University of Naples in 1990 where she received her PhD degree in Computer Science in 1996. Her research interests include distributed systems, systems modelling and formal methods in system specification and design.