Formal comprehensiveness and uniformity and semantic intra and intermodel consistency in the representation of Discrete Event Dynamic System models

Wilson M. Arata
University of São Paulo, Escola Politécnica, Brazil
wimarata@usp.br

Paulo E. Miyagi
University of São Paulo, Escola Politécnica, Brazil
pemiyagi@usp.br

Abstract. *A remarkable characteristic in the research on Discrete Event Dynamic Systems, such as productive systems, manufacturing systems, and information systems, is the heterogeneity of models. Due to the specific nature of these models, there are cases where multiple and heterogeneous models are necessary so that an adequate coverage of system aspects and of analytic results can be provided. This work discusses issues on the computational representation of these models, more specifically, that an adequate treatment of the formal heterogeneity of models and of their integration, mainly the semantic aspects of it, can enhance the workflow of computationally supported modeling and analysis experiments.*

Keywords: *Discrete Event Dynamic Systems, tools, modeling, analysis*

1. Introduction

Under the perspective of Discrete Event Dynamic Systems (DEDS) (Cassandras,1993), an important class of system can be described and analysed, such as productive systems, manufacturing systems and information systems. A characteristic of DEDS is the multiplicity and diversity of modeling and analysis approaches, each one of them dealing with specific aspects of systems and dynamics and providing different types of information. If a wide range of system and dynamics characteristics and of analytical results is required, it is likely that one has to consider the use of multiple and heterogenous models. In this context, it is important that computational tools capable of dealing with this heterogeneity be available. This work discusses some important aspects of the infrastructure, mainly those related to the representation of information, that these tools should provide so that a fluid and efficient workflow in modeling and analysis experiments can be achieved.

This work focus on two sides of modeling and analysis experiments. The first one is on the need to deal with structurally different formal descriptions relative to each heterogeneous model, basically, due to the constraints determined by the mathematical formalisms used to analyze them. The other one is on the semantics of the models, that is, what they mean in terms of entities, facts and relationships observed in system and respective dynamics under examination. Specifically, it is shown how both parts should be implemented so that computationally supported modeling and analysis experiments can take full advantage of them.

Besides this section, there are a section on how to deal with the formal differences of heterogeneous models, another on the representation of the semantics of DEDS models, a section with an example of how to implement and use the descriptions treated on the previous sections, and conclusions.

2. Modeling and analysis environments

All computational modeling and analysis related to DEDS take place within Modeling and Analysis Environments (MAEs), that are the focus in this section. Ultimately, models are computationally handled in the form of numerical-symbolic constructs and the role of a MAE is to process numerical-symbolic constructs and generate numerical-symbolic constructs. The numerical-symbolic constructs are given the generic denomination of *structures* in the text that follows.

2.1 Mathematical description of MAEs and mapping models into them

Mathematically, a MAE A can be described by a triple $A = \langle L_C, L_B, C_A \rangle$.

$L_C = \langle T_C, E_C \rangle$ is a model composition language that provides the descriptive elements that comprise model representations, where T_C is the set of terms from which structures are made and E_C is the set of all valid (according to a certain criteria) structures — Figure 1(a) shows examples of T_C and E_C, the latter being a set of model representations expressed, in this example, by forms whose fields are correctly filled with the elements of the former.

$L_B = \langle P_B, E_B \rangle$ is a model building language, such that P_B is the set of operations on T_C (in L_C) provided by this language and E_B is the set of all structures that can be built using operations in P_B and terms in T_C — Figure 1(b) shows examples of P_B, E_B and the application of some operations in P_B in the construction of filled and semi-filled forms that

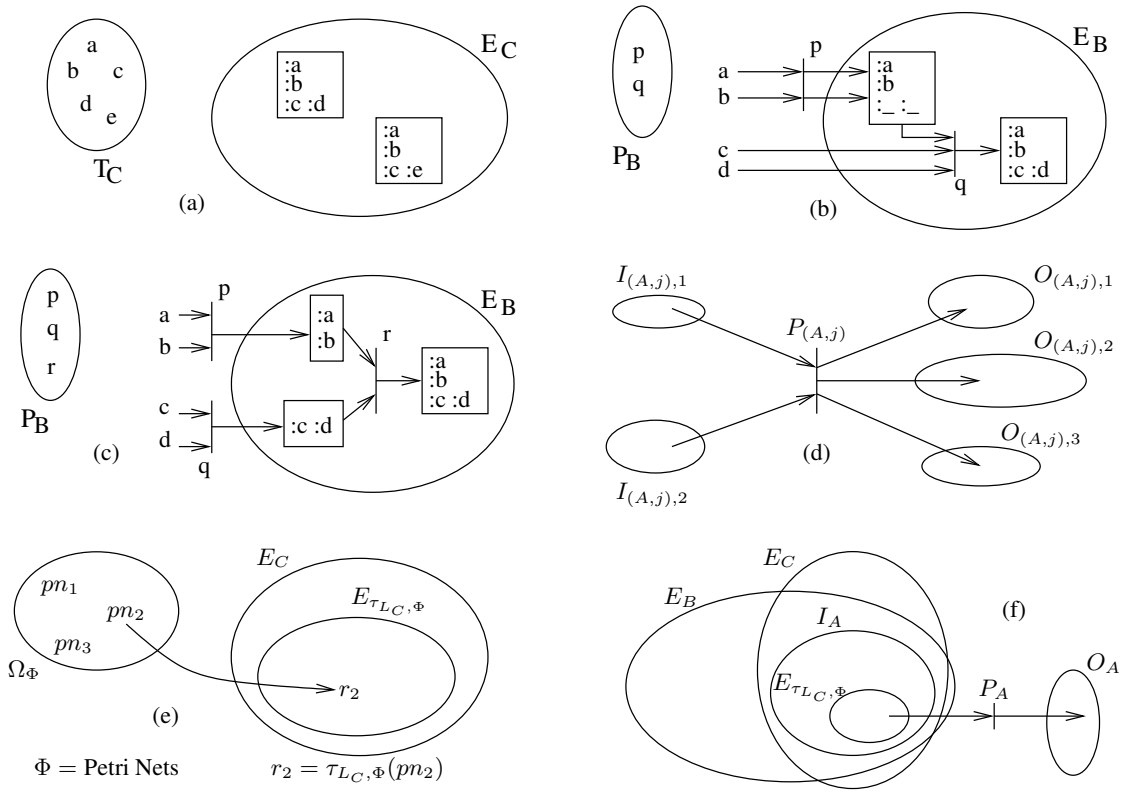

Figure 1. Elements of a MAE

belong to E_B; in Figure 1(c), another example shows a building language that provides operations to construct fragments of forms (operations p and q) and an operation r to compose complete forms from fragments.

$C_A = \{C_{(A,1)}, \ldots, C_{(A,n_A)}\}$ is a set representing the analytical capacity of the MAE, where $C_{(A,j)}$ is a triple $\langle P_{(A,j)}, I_{(A,j)}, O_{(A,j)}\rangle$, where $P_{(A,j)}$ is a *procedure* implementing an analysis, $I_{(A,j)} = (I_{(A,j),1}, \ldots, I_{(A,j),n_{(A,j),I}})$ and $O_{(A,j)} = (O_{(A,j),1}, \ldots, O_{(A,j),n_{(A,j),O}})$ are, respectively, the tuples whose elements are sets from which inputs are taken and to which the results from the procedure belong; so, an execution of $P_{(A,j)}$ takes $i_{(A,j)} = (i_{(A,j),1}, \ldots, i_{(A,j),n_{(A,j),I}})$ as input and generate $o_{(A,j)} = (o_{(A,j),1}, \ldots, o_{(A,j),n_{(A,j),O}})$ as results, where $i_{(A,j),k} \in I_{(A,j),k}, k \in [1, n_{(A,j),I}]$ and $o_{(A,j),k} \in O_{(A,j),k}, k \in [1, n_{(A,j),O}]$ — Figure 1(d) schematically illustrates a representation of a $C_{(A,j)}$, where a procedure take two structures as input and generate three structures as results.

Throughout the text, mentions to modeling language usually refer to the pair $\langle L_B, L_C\rangle$, what is clear by the context.

Along with mathematical models of MAEs, DEDS models are introduced by means of a transcription function $\tau_{L_C,\Phi} : \Omega_\Phi \longrightarrow E_C$, where L_C stands for a model composition language, Φ for a model type (like Petri Net (Murata,1989) or Markov Chain (Kulkarni,1995)), Ω_Φ for the set of all models of type Φ and E_C for the set of all valid models in L_C. Considering this function, there is a subset $E_{\tau_{L_C,\Phi}}$ of E_C such that $\forall \mu \in \Omega_\Phi, \tau_{L_C,\Phi}(\mu) \in E_{\tau_{L_C,\Phi}}$ and $\forall r \in E_{\tau_{L_C,\Phi}}, \exists \mu, \tau_{L_C,\Phi}(\mu) = r$. In other words, $E_{\tau_{L_C,\Phi}}$ is the set of all representations of models of type Φ in language L_C. Figure 1(e) presents an example involving Petri net.

2.2 Interpreting target sets

Important features of MAE can be visualized by means of some of the sets introduced in the previous subsections. They are given the denomination of *target sets*. The sets E_C, E_B, $I_{(A,j),k} (j \in [1, n_A], k \in [1, n_{(A,j),I}])$ and $E_{\tau_{L_C,\Phi}}$ are the target sets of the model composition language (L_C), the model building language (L_B), the inputs of the j-th procedures ($P_{(A,j)}$) and the transcription of models of type Φ onto a language L_C respectively. They should be interpreted as the expressive power of modeling language, the modeling language's capacity of building structures (in computational programs, it can be related to the pattern of interaction presented by their user interfaces to build model representations), the structures that can be analysed by a procedure, and (the models of) type Φ respectively.

As an example of how diagrams with target sets can be interpreted, considering Figure 1(f), with respect to a model type Φ, it is easy to visualize that a language L_C and a transcription $\tau_{L_C,\Phi}$ has been adequately provided so that models of Φ can be represented within L_C, as represented by $E_{\tau_{L_C,\Phi}} \subset E_C$; analogously, $E_{\tau_{L_C,\Phi}} \subset E_B$ indicates that L_B is able to generate the representations of Φ; in the same way, $E_{\tau_{L_C,\Phi}} \subset I_A$ indicate that the representations of Φ can be analyzed

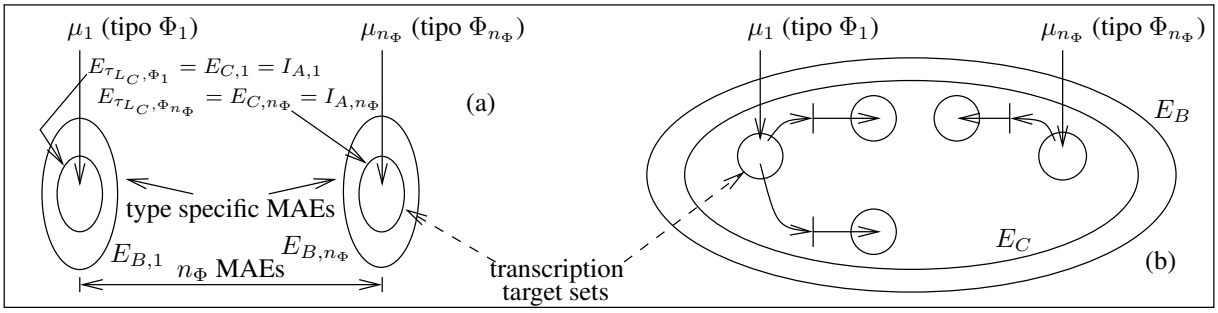

Figure 2. MAEs for heterogeneous models: (a) type specific (heterogeneous) MAEs and (b) comprehensive MAE

by P_A; for completeness purposes, it is shown that the results of P_A are structures not covered by any of the target sets.

Figure 2 illustrates two cases handling heterogeneous models. In Figure 2(a), each type of model is handled by a specific MAE, as indicated by the target sets. In Figure 2(b), all types of models are represented using a comprehensive model composition and building languages, including the results of the procedures implementing mathematical analysis. Both configurations are further discussed in the text.

Depending on model building language $L_B = \langle P_B, E_B \rangle$, if appropriate operations are provided in P_B, the process of building model representations can occur with the participation of several operations. In this case, besides the aimed model representation, the process generates intermediary structures, as it is the case of the already discussed Figures 1(b) and 1(c). In the same figures, it is easy to understand that it is possible to reuse substructures to build multiple model representations. This feature is interesting when many models with common parts have to be built in an experiment. Another characteristic is that, the greater the number of intermediary structures, more paths to the aimed model representations can be followed, as it can support diverse model representation building paths. Again, this can be used to support varied configurations of modeling and analysis experiments.

So, since this work is oriented to enhance the productivity of experiments, it assumes that stepwise operations are provided by the building language in the examples; also, the size of target set of the building language, that correlates with the amount of intermediary structures, can be seen as a measure of the flexibility of this language.

Regarding the composition language, consider the following definition:

Definition 21

An ideally comprehensive and uniform modeling language is one that, with the same set of terms and complex structure composition rules, is capable of building representations of models of an indefinite number of types.

A conclusion from this definition is that, in the case of uniform comprehensive modeling language, the inclusion of a new type of models is a matter of finding a suitable transcription function. The uniformity is specially important when a new type of model is to be considered: Figure 3(a) shows an example of a nonuniform model composition language L_C (with target set E_C and covering model type Φ_1) that, in order to expand its coverage to model type Φ_2, it must be replaced by a language L'_C (with target set E'_C); observe that, due to the nonuniformity, there is an impact on the model building language, that has to be extended (from E_B to E'_B) to deal with the representations of models of Φ_2.

Due to the flexibility of the model composition language, there are two problems that must be addressed: the ambiguity and the redundancy of representations.

The ambiguity of representations occurs if there are two (or more) transcription functions that map two (or more) models of different types into a same structure, as illustrated in Figure 3(b). For example, when analysing a set of models of type Φ_1, a problem arises if representations of model of type Φ_2 are inadvertently included in the assemble of structures to undergo the same analysis: the origin of the inconsistencies in the results can be difficult to track down. In programming language, a similar problem is addressed by type systems (Cardelli and Wegner, 1985) and, along this line, Arata and Miyagi(2003) describes an implementation that employs typed terms so that this confusion can be avoided.

The redundancy of representations occurs if there are two (or more) transcription functions for models of the same type (as illustrated in Figure 3(c)). The problem here is the possible duplication of cost in utilization of resources (like model storage costs) and in the management of these differences if both representation schemes are involved in a experiment. A metamodel-based reference scheme is presented in (Arata and Miyagi,2003) that supports the standardization needed to, at least, minimize this problem.

2.3 Analysis of configurations of MAE

Let the configuration in Figure 2(a) be called Heterogeneous MAE (HM) configuration and and that in Figure 2(b), Comprehensive MAE (CM) configuration. Observe, in the latter, that the languages L_C and L_B and the procedures implementing the analysis are conceived in a way that the results of the procedures are also expressed by L_C, easing the

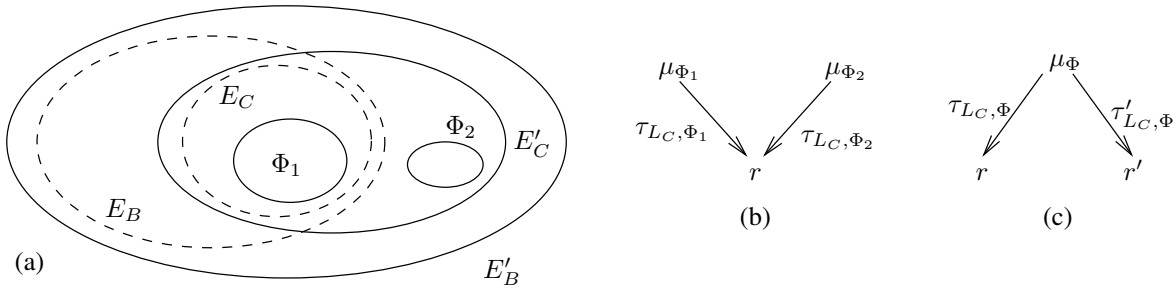

Figure 3. Issues of comprehensive modeling languages: (a) nonuniformity, (b) ambiguity and (c) redundancy

use of the analysis results as inputs to other analyses. Furthermore, it is considered that the model composition language in CM configuration is comprehensive and uniform.

Introducing costs to enable other comparisons, let us assume that the experiments that are considered are identical, with same models, analyses and operation sequence patterns and, thus, same storage and operational costs. The comprehensiveness and uniformity of the language in the CM configuration becomes evident, as the number of types of models increases, by constant learning (in training users) and implementation costs relative to the MAE; in the HM configuration, these two costs tends to increase as new MAEs are implemented for each new type of models.

Comparing, at first, the case where there is no integration of models, that is, experiments dealing with one type of model at a time, the costs to compare are those essentially related to the implementation and learning. In this case, one can conclude that, as the heterogeneity of model types increases, the situation becomes more favorable to the CM configuration.

When model integration occurs, the costs in switching contexts of work in the HM configuration, consequence of the handling of heterogeneous models in different MAEs, must be considered. So, as the degree of integration increases, there is a stronger tendency to the adoption of the CM configuration.

Therefore, there are two main factors that make the CM configuration a better choice: the heterogeneity of model types and a workflow that involves an integration of heterogeneous models.

Thus, at this point, it is possible to enumerate important features that an extensible MAE for heterogeneous models should present: the elements of MAE and the transcriptions of different types must be coordinated, comprehensive and uniform model composition language, flexible model building language, analytical extensibility (to incorporate new analysis procedures), ambiguity avoidance mechanisms, and redundance minimization mechanisms.

These guidelines are important to computationally support an effective workflow in modeling and analysis experiments involving heterogeneity, benefiting application and research projects as a wider coverage of aspects can achieved and more techniques can be developed and applied.

3. Representing the semantics of models

An important part of the modeling and analysis experiments has a formal orientation, like when ensuring the well-formedness of models and in the application of mathematical analysis techniques. However, the semantics of models is equally important. In this context, semantics refers to the meaning, that is, to what is being represented by them. So, the semantics of models corresponds to what is observed in the dynamics being treated, and the same must be true for the results of the analysis of those models. Normally, this must be ensured during the conception models.

This text proposes a structured and consistent representation scheme of the semantics involved in modeling and analysis experiments as a means to present the information provided by the models in a relatively uniform way: so, to obtain information about dynamics being modeled, instead of having to deal with diverse mathematical constructs (reflecting the heterogeneity of models), it can be derived from a set of semantics-oriented descriptions presented in a uniform manner. Thus, it is shown that an adequate representation of the semantics can bring different improvements to computationally supported modeling and analysis experiments, particularly when intra and intermodel consistency is observed. For that purpose, predicate-based representations of semantics are presented.

3.1 Representing with terms, functors and predicates

In the representations of semantics presented here, the most basic element is the term. Terms are used mainly to represent entities or in the formation of functors or predicates.

Functors are a form to represent objects with complex structures and they are similar to the functors adopted in logic programming languages like Prolog (Deransart et al., 1996). Functors have the following syntax:

<name>(<argument>,...,<argument>)

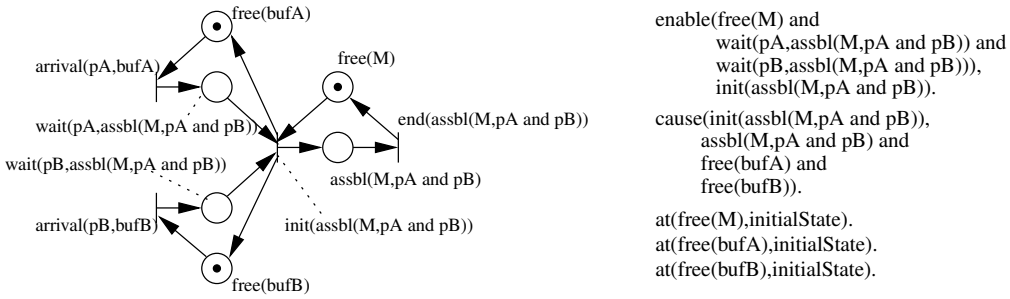

Figure 4. Annotated Petri net and predicates describing it

that is, it has, in the beginning, a name followed by a list of terms, other functors or logical expression involving functors within parentheses. Functors can be nested indefinitely.

The functors **process(M,p)**,**init(process(M,p))** and **end(process(M,p))** can be used to express "machine M process a part p", "Initiating the processing of a part p by machine M" and "Ending the processing of a part p by machine M".

Expressions involving functors can be constructed by the use of the conjunctive operator **and**. For example, the expression **init(process(M,p) and go(p,labA))** can be used to represent "start of processing by machine M on a part p that is destined to be sent to laboratory labA (for tests)". One important aspect of expression is the inferability: from this expression, it is possible to infer the occurrence of **process(Machine,part)**; the same can be said of **go(part,labA)**. This is important in the achievement of consistency in the representation of semantic information.

Predicates are used to state that a fact is true. Predicates have the same basic form as functors, that is, a name followed by, in parentheses, a list of arguments that can be terms or functors (or expressions with functors), except that there is a period at the end and predicates cannot be nested.

For example, the predicate **at(process(M,p),s12)** can be employed to indicate that the statement "at state s12, machine M is processing a part p" is true.

3.2 Representing facts described by a model

Basically, the approach presented here is to associate representations in the form of terms, functors or expressions to elements of models. That representations are denominated annotations. The descriptions of what the model as whole is expressing is represented by predicates.

An effort should be dedicated to the consistency in the representation of facts, so that, if a fact occurs in various situations, there is a way to infer that occurrence from the description of each of these situations. So, if a situation presents that fact, relationships involving the latter can be relevant to that situation.

Considering the Petri net in Figure 4, the *transitions* represents the beginning and the end of a condition (annotated by functor **assbl(M,pA and pB)**). Both the beginning and the end refers to the same condition, and it is reflected in the functors that annotate the *transitions* have the same argument. So annotations are useful in relating model componentes to the elements they correspond in the dynamics being modeled and to reproduce the relationships between those elements.

The idea here is to represent the semantic interpretations of facts described by a model. The pattern expressed by a model represents the relationships between what is represented by its elements. So, from the pattern that constitute a model and the representation of the semantic interpretation of its elements, one can expect that representations of the semantic interpretations of facts described by the model can be built. This facts can be expressed by means of the predicates involving terms and functors that annotates the elements of the model.

With annotations and the specific pattern described by the models, it is possible to derive the predicates describing semantic aspects os those models. For example, the Petri net in Figure 4 models an assembly operation by a machine M on a part of type A and another of type B; the parts must be in buffers bufA and bufB before the operation begins; the buffers can hold one part at most at a time. The predicates on the right side of the figure can be elaborated. The **enable** predicate represents the conditions that must be present to enable the begining of the assembly operation (**init(assbl(M,pA and pB))**), that is, machine M is free (**free(M)**) and parts of types A and B in the buffers (functors **wait**). The **cause** predicate indicates what happens as the assembly operation is initiated: the buffers are released (functors **free**) and assembly is executed (**assbl(M,pA and pB)**). The **at** predicates indicate that, in the initial state, machine M and buffers bufA and bufB are free. Note that these predicates can be constructed computationally.

3.3 Semantic bindings between different models

There are cases where elements of different models of the same system and dynamics are semantically bound. A notorious case is that of the isomorphism between stochastic Petri nets (SPN) and continuous-time Markov chains (CTMC)

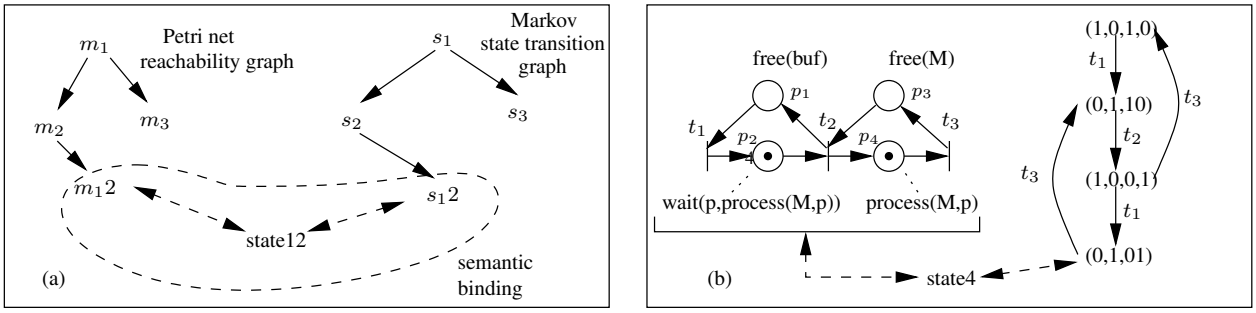

Figure 5. Semantic bindings: SPN-CTMC isomorphism and predicates involving PN annotations and markings

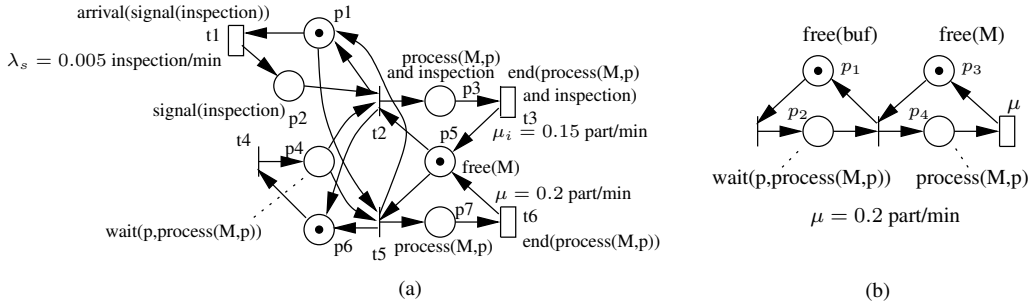

Figure 6. GSPN models

(Molloy, 1980), one consequence of which is that their state transition graphs are topologically identical and the stochastic timings associated to the transitions have the same exponential distributions. Considering an example of this case and the predicate elements, one form to represent this binding is to associate both a SPN *marking* m_{12} and the isomorphic CTMC state s_{12} to a same term, for example, **state12**, as pictorially described in Figure 5(a). So, references to the term **state12** refer to what is modeled by both the SPN *marking* and the CTMC state. The use of the same term is an expression of the bindings between these two models and it is a way to ensure the consistency of the representations of the semantic information relative to SPN *markings* and CTMC states.

Considering the example in Figure 5(b), a Petri net and its respective reachability graph are shown; also, following the guidelines given in the last paragraphs, a term, **state4**, is used to represent the marking displayed by the Petri net. With this term and the annotations in the places containing tokens, predicates like **at(state4,processing(M,p))** and **at(state4,waiting(p,processing(M,p)))** can be elaborated to denote that, in that state, machine M is processing a part p and a part p is waiting for processing in machine M, describing that particular *marking*. This is an example of propagation of elements of the semantic representations from one model to another, what can be seen as a display of intermodel consistency.

The consideration of semantic bindings is an important element in the representation of integration of models. Such bindings are also considered in, for example, the hierarchical model composition in the SHARPE system (Trivedi, 2002) and in the Möbius framework (Sanders et al., 2003) (as formalisms are described in terms of the components defined in the framework).

4. Examples

The generalized stochastic Petri net (GSPN) (Marsan, 1984) in Figure 6(a) models a service center where a processing machine M performs jobs on parts, with a buffer that can hold just one part to be processed. Part processing by M ocurr at rate $\mu = 0.2$ part/min. As soon the buffer is released, a new part gets into it. Inspections are made in the processing of parts from time to time; an inspection occurs as soon as a processing cycle begin after the inspection signal is set. The inspected processing has an execution rate of μ_i, that is smaller than the the uninspected processing rate μ (equal to the rate of the previous example), due to measurements to be made. The next inspection signal is issued, in average, after an interval of time given by $1/\lambda_s$ (according to an exponential distribution) since the beginning of the current inspection.

The analysis of this model is made by generating the timed reachability graph (that describes the tangible markings and the timed and imediate transtion firings between them), construction of the isomorphic Markov chain and the calculation of the stedy state probabilities from the chain. Just to simplify the examples, an assumption is made so that only examples of generalized stochastic Petri nets without conflicts between imediate transitions are considered.

The following examples consider the modeling language presented by Arata and Miyagi (2003). It is a language that can express the representations of many types of models using some basic (called atomic) types and two complex type

```
P is atomic; TT is atomic; IT is atomic;
MarkingType is setof (place is P,
                      tokens is Integer);
MarkedGSPNType is (
   Places is setof P,
   TimedTransitions is setof TT,
   ImediateTransitions is setof IT,
   ITT is setof (arc is (place is P,
                         transition is T),
                 weight is Integer),
   IIT is setof(arc is (place is P,
                        transition is T),
                weight is Integer),
   OTT is setof(arc is (transition is T,
                        place is P),
                weight is Integer),
   OIT is setof(arc is (transition is T,
                        place is P),
                weight is Integer),
   Marking is MarkingType
    );
```
(a)

```
TimedReachabilityGraphType is
   setof (origin is MarkingType,
          end is MarkingType,
          timedTranstion is TT,
          imediateTranstitions is
             setof (order is Integer,
                    transition is IT)
          );
```
(b)

```
S is atomic;
CTMCType is
   (states is setof S,
    transitions is
    setof (origin is S, end is S, rate is Real)
    );
```
(c)

```
SteadyStateDistribution is
   setof (state is S, probability is Real);
```
(d)

Figure 7. Metamodels of Marked GSPN, Timed Reachability Graphs, Continuous-Time Markov Chains and Steady-State Probability Distribution

```
# GSPN model representation
#
gspnmodel →
({p1,p2,p3,p4,p5,p6,p7},
 {t1,t2,t3,t4,t5,t6},
 {((p1,t1),1),((p3,t3),1),((p7,t6),1)},
 {((p2,t2),1),((p4,t2),1),((p4,t5),1),((p5,t2),1),
  ((p5,t5),1),((p6,t4),1),((p1,t5),1)},
 {((t1,p2),1),((t3,p5),1),((t6,p5),1)},
 {((t2,p1),1),((t2,p3),1),((t2,p6),1),((t5,p1),1),
  ((t4,p4),1),((t5,p6),1),((t5,p7),1)},
 {(p1,1),(p2,0),(p3,0),(p4,0),
  (p5,1),(p6,1),(p7,0)}
);
```

```
# marking m1
# m2, m3 and m4 omitted
m → {(p1,1),(p2,0),(p3,0),(p4,1),
     (p5,0),(p6,0),(p7,1)};
```

```
# timed reachability graph representation
#
trg → {(m1,m1,t6,{(1,t5)}),
       (m1,m2,t1,{}),
       (m2,m3,t6,{(1,t2),(2,t4)}),
       (m3,m1,t3,{(1,t5),(2,t4)}),
       (m3,m4,t1,{}),
       (m4,m3,t3,{(1,t2),(2,t4)})
      };
```

```
# isomorphic CTMC representation
#
isoctmc →
  ({s1,s2,s3,s4},
   {(s1,s2,0.005),(s2,s3,0.150),(s4,s3,0.200),
    (s3,s4,0.005),(s3,s1,0.200)}
  );
```

```
# steady state probabilities representation
#
distprob →
    {(s1,0.944),(s2,0.031),
     (s3,0.024),(s4,0.001)};
```

Figure 8. Representations of the models involved in the analysis of the GSPN in Figure 6(a)

building constructs (homogeneous typed sets and tuples). The basic procedure is to, first, define metamodels for the types to be represented, that can be used as reference of how representations of models of a certain type should be constructed and to well-formedness checking procedures. Figure 7 presents the metamodels of Petri Net, timed reachability graph, Markov Chains and steady state probabilities of Markov Chains. For example, Continous-Time Markov Chains are described by **CTMCType**, that is a two-element tuple where the first one is the set of states of the chain and the second one is the set of the state transitions, each represented by a tuple denoting the "**origin**" state, the "**end**" state, and the rate of the transition. In Figure 8, the representations of the models relative to the current example is presented. The representations of the models, despite their differences, are all expressed using the same lexical and syntatic elements; the differences in the representations are inherent to the heterogeneity and so they are to meet the well-formedness requirements set by the mathematical analysis techniques. Then, in Figure 9(a), the predicates relative to the models are presented; in this case, instead of the heterogeneity of the structures of the representations directed to mathematical analysis, a flatter description (i.e., without the need to understand the intricacies of each mathematical model) is provided; notice that the consistent representation of the semantic bindings is important to achieve this flatness.

Considering the model in Figure 6(b), it models the same system in a similar operation, except that there is no inspection. Making the same analysis and generating the same models, the predicates generated are presented in Figure 9(b). The use of the same predicates and terms as those in the previous example allows to easily visualize the relationships between different dynamics. In fact, it is also possible to define a common analysis procedure to execute computations on the both sets of predicates, since, for these examples, they both have the necessary elements to calculate the respective throughput of the completion of activity denoted by **process(M,p)**.

```
rate(process(M,p),.2).
rate(process(M,p) and inspecting,.15).
rate(arrival(signal(inspection)),.005).
in(state1,StateSpace_inspection).
in(state2,StateSpace_inspection).
in(state3,StateSpace_inspection).
in(state4,StateSpace_inspection).
at(process(M,p),state1).
at(process(M,p) and inspecting,state2).
at(process(M,p) and inspecting,state3).
at(process(M,p),state4).
prob(state1,.915).
prob(state2,.023).
prob(state3,.030).
prob(state4,.032).
```

(a)

```
rate(process(M,p),0.2)
in(state0,StateSpace)
at(process(M,p),state)
prob(state,1)
```

(b)

Figure 9. Two sets of predicates describing the models generated in experiments involving analysis of the GSPNs in Figures 6(a) and 6(b) respectively

5. Conclusions

From the set-based description of the interactions between model types and modeling languages, helpful in the visualization of the configurations that these elements can assume, plus the consideration of costs, this work concludes that comprehensiveness and uniformity are important features in the modeling languages when dealing with heterogeneity of DEDS models. While these aspects support a coexistence of heterogeneous models within a computational environment, this work presents descriptions based on the semantics of models as a means to represent (and, so, to take advantage of) the integration of models: in this case, the main feature to achive is the consistency in the representation of these descriptions. In short, this paper proposes that these approaches, as long uniformity and consistency are observed, must be seriously considered in computational environments supporting DEDS modeling and analysis experiments.

6. Acknowledgements

The authors gratefully acknowledge the financial support to the present project of the Brazilian Governmental Agencies CNPq, CAPES and FAPESP. Particularly, the authors would like to thank TIDIA/KyaTera program under which this work is developed.

7. References

Arata, W.M. and Miyagi, P.E.,2003, "Uniform computational treatment of heterogeneous discrete-event dynamic system models", Proceedings of 9th IEEE International Conference on Emerging Technologies and Factory Automation,Lisbon, Portugal, pp.47-53.

Cardelli, L. and Wegner, P., 1985, “On Understanding Types, Data Abstraction, and Polymorphism”, ACM Computing Surveys, Vol.17, No. 4, pp.471-522.

Cassandras, C.G., 1993, ‘ ‘Discrete Event Systems: Modeling and Performance Analysis”, Richard D. Irwin Inc., Burr Ridge, USA.

Deransart, P., Cervoni,L. and Ed-Dbali,A., 1996, “Prolog: the standard: reference manual”, Springer-Verlag, London, UK.

Kulkarni, V.G., 1995, “Modeling and Analysis of Stochastic Systems”, Chapman & Hall, London, UK.

Marsan, M.A., Conte, G. and Balbo, G., 1984, “A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems”, ACM Transactions on Computer Systems, Vol. 2, No. 2,pp. 93-122.

Molloy, M.K., 1980, “Performance Analysis Using Stochastic Petri Nets”, IEEE Transactions on Computers, Vol. C-31 , No. 9, pp. 913-917.

Murata, T., 1989, “Petri Nets: Properties, Analysis and Applications”, Proceedings of IEEE, Vol. 77, No. 4, pp. 541-580.

Sanders, W.H., Courtney, T., Deavours, D., Daly, D., Derisavi, S. and Lam, 2003, “Multi-formalism and Multi-solution-method Modeling Frameworks: The MÃűbius Approach”, Proceedings of the Symposium on Performance Evaluation - Stories and Perspectives, Vienna, Austria pp. 241-256.

Trivedi, K.S., 2002, “SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator”, Proceedings of2002 International Conference on Dependable Systems and Networks (DSN 2002), http://csdl.computer.org/comp/proceedings/ dsn/2002/1597/00/15970544.pdf.