# Using Rewriting Systems for Performance Analysis

Lorenzo Capra[1,*,†], Marco Gribaudo[2,*,†], Mauro Iacono[3,*,†] and Michael Köhler-Bußmeier[4,*,†]

[1]Università degli Studi di Milano, via Celoria 18, 20133 Milano, Italy
[2]Politecnico di Milano, via Ponzio 34/5, 20133 Milano, Italy
[3]Università degli Studi della Campania "L. Vanvitelli", viale Lincoln 5, 81100 Caserta, Italy
[4]Hamburg University of Applied Sciences, Berliner Tor 7, D-20099 Hamburg, Germany

**Abstract**

The modeling and analysis of adaptive distributed systems, specifically those that possess (self-)reconfiguration or modification capabilities, present a considerable challenge, necessitating the use of appropriate formalisms and techniques. Traditional quantitative analysis frameworks demonstrate constrained expressiveness and should be amalgamated with innovative methodologies. Rewriting-based frameworks appear to be more suitable, despite their predominant application in formal verification. This paper examines the use of Maude as a framework for performance or probabilistic analysis in adaptive distributed systems, highlighting recent advances.

**Keywords**

Performance evaluation, distributed systems, self-adaptation, rewriting systems, Petri nets, Maude

## 1. Introduction

Analyzing and modeling adaptive distributed systems, particularly those with capabilities for (self-) reconfiguration or modification, pose significant challenges and require the application of suitable formalisms and techniques. Traditional quantitative analysis frameworks, including stochastic process algebras, timed and stochastic Petri nets (PN) and timed/probabilistic automata exhibit limited expressiveness when the system undergoes big changes during its execution. These modeling frameworks lead to the merging of functional and non-functional elements within models, impacting scalability. Various noteworthy extensions have been suggested, such as the $\pi$-calculus, the Ambient calculus, and the Nets-within-Nets paradigm. However, these extensions have not been well-supported by tools and efficient analysis features. Systems based on rewriting, especially Graph Transformation Systems, are more suitable, but they are mainly used for formal verification. This paper explores the utilization of Maude as a framework for performance or probabilistic analysis in adaptive distributed systems, focusing on recent progress.

Maude [1] is a purely declarative language with high performance and sound rewriting logic semantics [2]. It achieves efficiency and expressiveness through pattern-matching modulo operator attributes, subtyping, partiality, generic types, and reflection. A Maude system module is an executable specification for a distributed system. The Maude runtime engine provides various facilities for model checking, verification of LTL formulae, infinite-state analysis, and symbolic reachability analysis. Furthermore, Maude has been used as a logical framework for other formalisms, such as Petri Nets (PN), Automata, and Process Algebra. These formalisms, though powerful, lack the necessary features for modeling adaptable systems intuitively.

Maude possesses intuitive rewriting semantics. Statements are classified as *equations* (utilized as simplifications) or *rewrite rules* (representing local concurrent transitions). A Maude specification comprises *Functional* modules (which contain only equations) and *System* modules (which include rules and potentially equations). A functional module characterizes a *theory* $(\Sigma, E \cup A)$ within the

*QualITA 2025: The Fourth Conference on System and Service Quality, June 25 and 27, 2025, Catania, Italy*
*Corresponding author.
†These authors contributed equally.
lorenzo.capra@unimi.it (L. Capra); marco.gribaudo@polimi.it (M. Gribaudo); mauro.iacono@unicampania.it (M. Iacono); michael.koehler-bussmeier@haw-hamburg.de (M. Köhler-Bußmeier)

framework of membership *equational* logic: $\Sigma$ constitutes the signature (operators, sorts, subsorts), $E$ comprises the set of axioms, and $A$ encapsulates the operator equational attributes (such as AC). The model of an equational theory is the initial algebra, specifically, the *quotient* term algebra $T_{\Sigma/E\cup A}$, which, assuming confluence and termination conditions are met, is isomorphic to the *canonical* term algebra: consequently, the mathematical and operational semantics align. A system module represents a *rewrite theory* $(\Sigma, E \cup A, R)$, $R$ being a nonempty set of rules. The corresponding model is set forth as a *labeled transition system* (TS) associated with each term: therein, the distributed states are represented by *canonical* terms, whereas state transitions illustrate *classes* of equivalent rewrites.

**Related Works** Several options exist for timed and probabilistic analysis using `Maude`. [3] presents a non-up-to-date survey. The framework presented in [4] enables deterministic time specifications to analyze real-time systems. A branching-time analysis framework for `Maude` specifications is described in [5]. The approach detailed in [6], based on probabilistic rewrite theories associated with actors, enables probabilistic discrete-event simulation. [7] presents a reflective framework in `Maude` for the quantitative analysis of self-adaptive systems. More recently, [8] introduced a comprehensive method for using `Maude` in stochastic analysis through a probabilistic extension of its strategy language. In particular, this strategy language operates at the "object" level rather than the meta-level.

## 2. Associating a Markov Process to `Maude` Executable Modules

In [9], we introduce a new general methodology to generate a (continuous- or discrete-time) Markov chain (MC) from user-defined `Maude` executable modules (including stochastic parameters) directly and systematically. Our approach and objectives differ significantly from the above mentioned works. Essentially, our goal is to equip any executable specification in `Maude` with probabilistic semantics directly, through (automatic) "preprocessing" the original modules, marking an important step toward automating the entire process.

Calculating the MC stochastic matrix exactly is challenging due to rewriting logic semantics, which obfuscates multiple state transitions. The methodology, which functions for any `Maude` executable specification, is illustrated through a challenging application domain: stochastic PN with a dynamically changing structure (see the next section). This application example outlines all the potential issues related to the accurate derivation of a Markov process from `Maude` executable modules.

The rewriting logic establishes a labeled transition system (TS) associated with ground terms of any type. However, deriving a consistent Markov chain for this TS presents challenges for three main reasons: TS state transitions correspond to equivalence classes of rewrites; equivalent rewrites may be logically indistinguishable and need to be united; and local rewrites of subterms within a specific term may occur. To our knowledge, none of the mentioned techniques addresses all these issues. Our method, which delineates a class of meta-operators at the object level, is more straightforward to implement and significantly more efficient than the predefined `Maude`'s meta-level modules when addressing extensive state spaces (comprising dozens or hundreds millions of states). In addition, it shows greater precision compared to using the `Maude`'s strategy language.

Stochastic parameters are first integrated into a `Maude` executable specification (a system module) flexibly and intuitively. Next, the challenge of accurately calculating state transition rates is tackled by methodically preprocessing executable modules so that they generate an enhanced description of states associated with terms, which contains all the information to calculate state-transition rates exactly. Thus, we obtain the corresponding MC generator matrix through fundamental text processing.

We will outline the method to obtain the MC generator matrix using the `Maude` representation of Stochastic Petri Nets (SPN) [10, 11]. The formalization of SPNs contains a straightforward hierarchy of modules accessible at https://github.com/lgcapra/rewpt. The module `SPN-SIG{TL :: TRIV, PL ::TRIV}` establishes the SPN signature and is parameterized by both place and transition labels. A parameterized module implements type parameters through (functional) *theories*. These theories define the module interfaces by specifying the syntactic and semantic properties

for the parameter modules. Theories have loose semantics, meaning they accept any algebra that satisfies the stipulated equations and membership axioms. In Maude, *views* connect a source theory to a target module or theory and specify the mapping of sorts and operators. The theory TRIV merely requires a sort. The SPN signature is predicated upon a concise definition of multisets as weighted sums, as facilitated by the predefined module BAG{X :: TRIV}.

This module is imported in the protecting mode, thus preserving the initial semantics within the module SPN-SIG, utilizing the formal parameter PL as the actual parameter for BAG. Specifically, the sort Pbag encompasses multisets of places. SPN transitions, which are terms of sort Tran, are characterized by labels associated with adjacency lists, conveyed through Pbag triples [I, O, H]. These labels consist of a descriptive tag (a String in our context), a Float representing the rate parameter of a negative exponential firing delay, and a Nat that specifies the firing policy. For example, the ground term:

```
t("a", 1.5, 0) |-> [1 . p(1) + 2 . p(2), 1 . p(1), 2 . p(1)]
```

delineates a transition identified by the label "a", an exponential firing rate $\mu = 1.5$, and typified by an infinite-server type. This transition requires the presence of exactly one token in place $p_1$ and a minimum of two tokens in place $p_2$ for the enabling. Upon firing, it removes two tokens from $p_2$. The PT net that underlies an SPN, classified as a term of type Net, is defined straightforwardly in a modular fashion through the utilization of the associative/commutative juxtaposition ; and the subsort relationship Tran < Net.

Using the predefined firingRate operator, we can define marking-dependent rates: The current definition is based on the enabling degree ($ed(t, m)$), which refers to the occurrences of a transition that are simultaneously enabled in a marking. Under the infinite server policy (0), the transition exponential rate is $\mu \cdot ed(t, m)$. Under the $k$-server policy, $k > 0$, it is $\mu \cdot min(ed(t, m), k)$.

The system module SPN-SYS{TL :: TRIV, PL ::TRIV} extends SPN-SIG by defining the SPN firing rule as a rewrite rule. This rule applies to the System terms, composed of a Net and a Pbag subterm.

In the Transition System (TS) generated in Maude, a single state transition may amalgamate multiple "equivalent" instances. For instance, consider various (SPN) transitions that are enabled in a given marking and upon firing reach the same target marking. This occurs frequently, for example, when generating a TS quotient using a canonical representative for markings [11]. To accurately determine the corresponding rates in the Markov chain's stochastic matrix, an approach grounded in automated preprocessing system modules is employed.

1. rewrite rules are translated into kinds of "meta-level" operators (at the object-level), which compute every distinguished rule match
2. because rewrite rules can apply locally to fragments of subject terms, they are encapsulated at the level of a subject term of in accordance with the terms' abstract syntax graph
3. upon preprocessing, an augmented TS is created wherein states include the exact state transition rates

Several experiments show tolerable overhead resulting from redundant state representation. Alternatives relying on Maude's predefined meta-level modules or strategy language tend to be unwieldy or imprecise and lack the capacity for full automation.

Focusing on point 1, which is critical in the process, the following excerpt shows the result of the *automated* encoding of the SPN firing rule in a corresponding firing match operator (subject to further optimization, here ignored): A term of sort StateTran{System} encompasses a target marking and the corresponding rate. The classic fixed-point iteration is used to calculate all possible matches (variable substitutions) of the rule. The same schema applies to any rewrite rule.

```
var T : Tran . vars M M' : Pbag . vars N N' : Net . var S : System .
var R : Float . var X : Set{StateTran{System}}. var XM : Match.

crl [firing] : N M => N fire(T, M) if T ; N' := N /\ enabled(T, M) /\ R := firingRate(T, M) .

*** rule's translation into an operator

op firing-match : System -> Set{StateTran{System}} .
eq firing-match(S) = $firing-match(S,noStateTranS) .
op $firing-match : System Set{StateTran{System}} -> Set{StateTran{System}} .
ceq $firing-match(S, X) = $firing-match(S, (XM --> S' : R) U X) if (T ; N) M := S /\
    enabled(T, M) /\ S' := (T ; N) firing(T, M) /\ R := firingRate(T, M) /\ XM := {N} & {T} & {M} /\
     (XM --> M' : R) in X = false .
eq $firing-match(S, X) = X [owise] .
```

## 3. Rewritable Stochastic PN

Rewritable Petri nets ([10, 12]) constitute a highly flexible model for distributed adaptive systems, as formalized in Maude. Within this framework, the PN firing rule, along with modifications to the net structure, are specified in a standardized manner. In [11], improvements have been achieved by integrating net algebra operators that identify symmetries, specifically Petri net automorphisms, through systematic and transparent node labeling. These node labels represent the modular and hierarchical configuration of the models, facilitating the construction of a quotient transition system via efficient normalization of states (canonical terms).

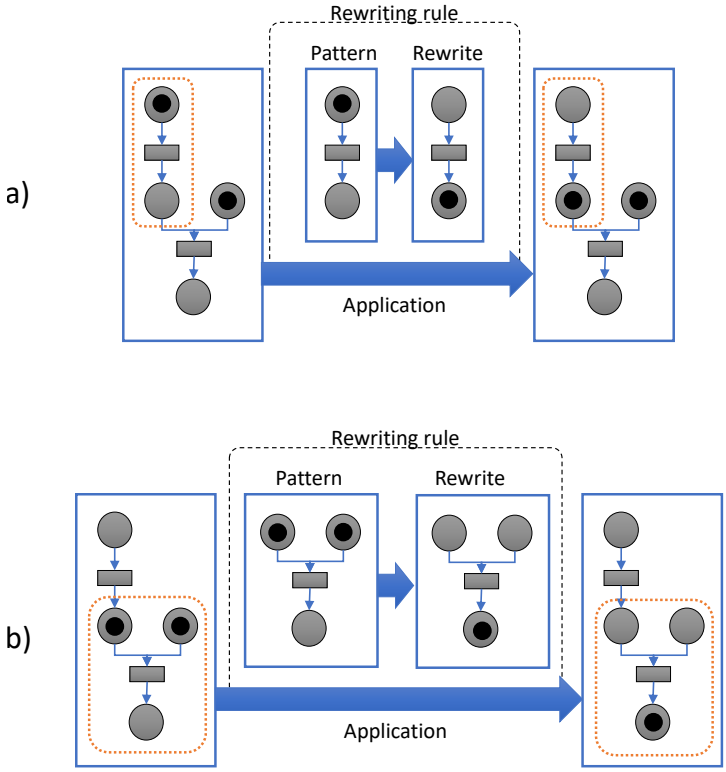


**Figure 1:** Following the semantics of a Petri Net as a graph rewriting system: a) simple transition, b) a more complex example.

Rewrite rules that rely on symmetric net transformations preserve this labeling strategy. Analyzing models modulo automorphisms is essential to maintain scalability across all frameworks predicated on graph transformation systems, of which rewritable PNs serve as an exemplar. In [13], we have extended rewritable PNs by incorporating stochastic parameters and outline a semi-automated procedure to extract a compact continuous-time Markov chain (CTMC) from the TS quotient.

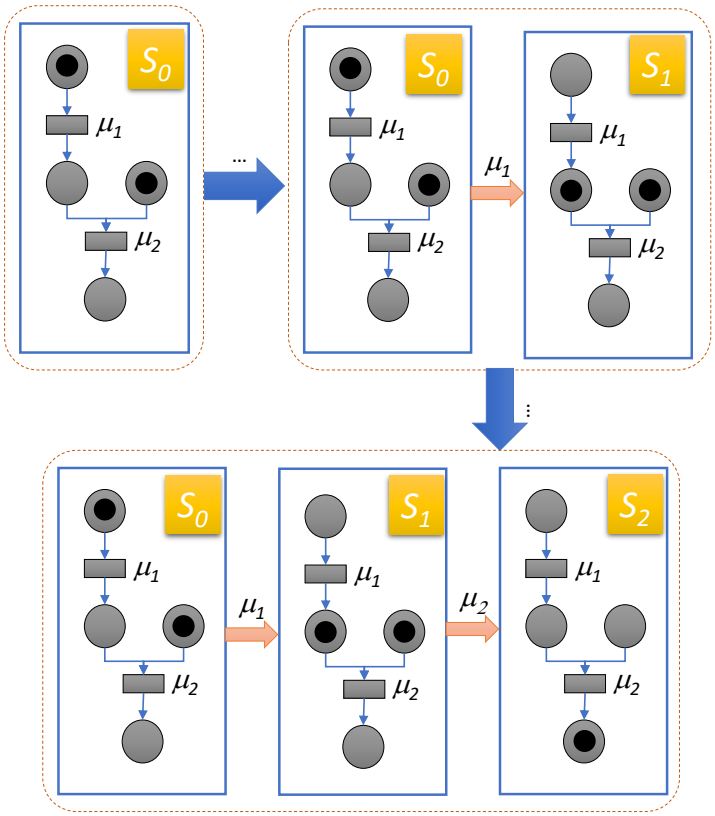


**Figure 2:** The generation of the infinitesimal generator as successive graph rewriting operations

This CTMC satisfies the exact lumpability property, which parallels the strong bisimilarity observed in the quotient transition system. The efficacy of this approach has been illustrated using a composite model of a gracefully degrading production system. We believe that by integrating the methodology described in [9], the process of generating lumped CTMC could be completely automated in the near future.

By creating proper rules, the semantics of a specific modeling formalism can be implemented. For example, Fig.1 shows how the semantics of a Petri net can be interpreted as a graph rewriting system. In particular, in Fig.1a), a rule matches a pattern consisting of a place marked with one token, connected to a transition, which is further connected to a place. The application of the rule rewrites the identified subnetwork, with one in which the token has been moved to the second place. Fig.1b) shows how these rewriting rules can be extended to a more complex case, where a pattern contains two tokens inside two input places, and the rewriting rule combines them and marks the third place. Please note that, generally, rules are specified using functional expressions, so one rule can match a large number of cases, making this process enumerable and feasible.

One of the most interesting properties of these types of rewriting system is that the nodes of the graphs can be graphs themselves. In this way, the rewriting can even produce the infinitesimal generator of a Stochastic Petri Net Model, as shown in Fig.2. In this case, the initial graph is composed of a single node, which is itself a graph that corresponds to the Petri Net in its initial state.

The first rewriting (whose formalization as a rule is omitted for simplicity) transforms this graph into another that adds a node, with the marking obtained by the first transition, and a connection, labeled with the transition rate ($\mu_1$) at which this event occurred. A second rewriting adds a third node to the

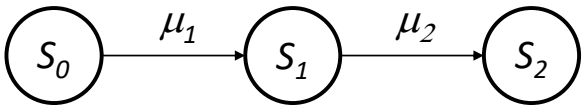


**Figure 3:** The CTMC created with graph rewriting

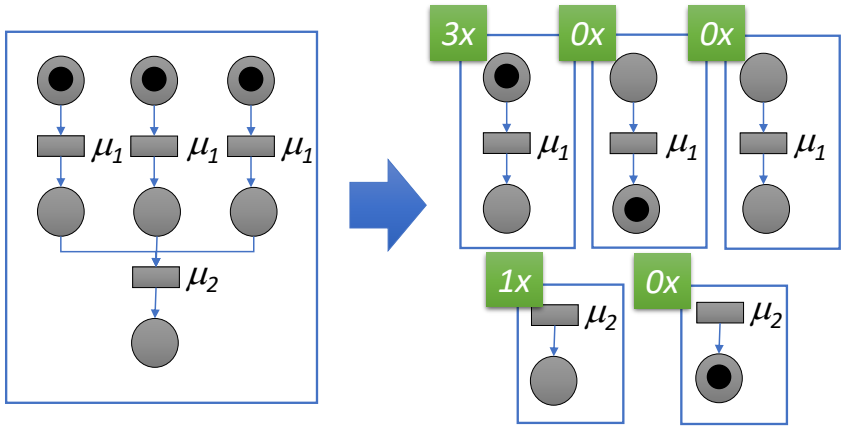


**Figure 4:** The encoding of a state as supported by the rewriting system.

outermost graph, which contains the final Petri net configuration, connected with an arc labeled with rate $\mu_2$. This corresponds to the CTMC that describe the evolution of the considered Stochastic Petri Nets, as shown in Fig.3

One of the advantages of creating the infinitesimal generator through graph rewriting is that the configurations of the model represented by the states of the CTMC are created in a compact way that natively performs lumping, as shown in Fig.4. This allows us to generate CTMCs with state spaces much smaller than in conventional techniques.

In this example, a model where three identical components are present, plus a second different one, is encoded by having the three possible evolution of each subcomponent exposed, and counting that we have three of them representing the initial state of the system are required. Also, the lower part of the model is encoded with an additional sub-network, where currently the version with the empty place is used once. This makes the classical state-space reduction techniques based on symmetries considered native, thus reducing the generated state spaces.

## 4. Probabilistic Nets-within-Nets

In [15], we examine Hornets enhanced by incorporating firing probabilities to represent multi-agent systems with capabilities for self-modification. The theory is explained in more detail in [14]. Hornets [16] embody a Nets-within-Nets formalism, a type of Petri net formalism characterized by tokens that themselves are Petri nets. Each net-token possesses its own firing rate, which operates independently from the firing rates of other net-tokens. Hornets furnish algebraic operations that enable modification of net tokens during firing. Within our probabilistic extension, these operators are capable of individually adjusting the net-token firing rate.

Our model is used to conduct a quantitative analysis of self-modifying systems. Hornets are particularly adept at modeling self-adaptive systems engaged in a MAPE-like loop (monitor-analyze-plan-execute). In this context, the system net represents the feedback loop, whereas the net-tokens illustrate the adapted model elements. We introduce a subclass of Hornets that can be readily formalized

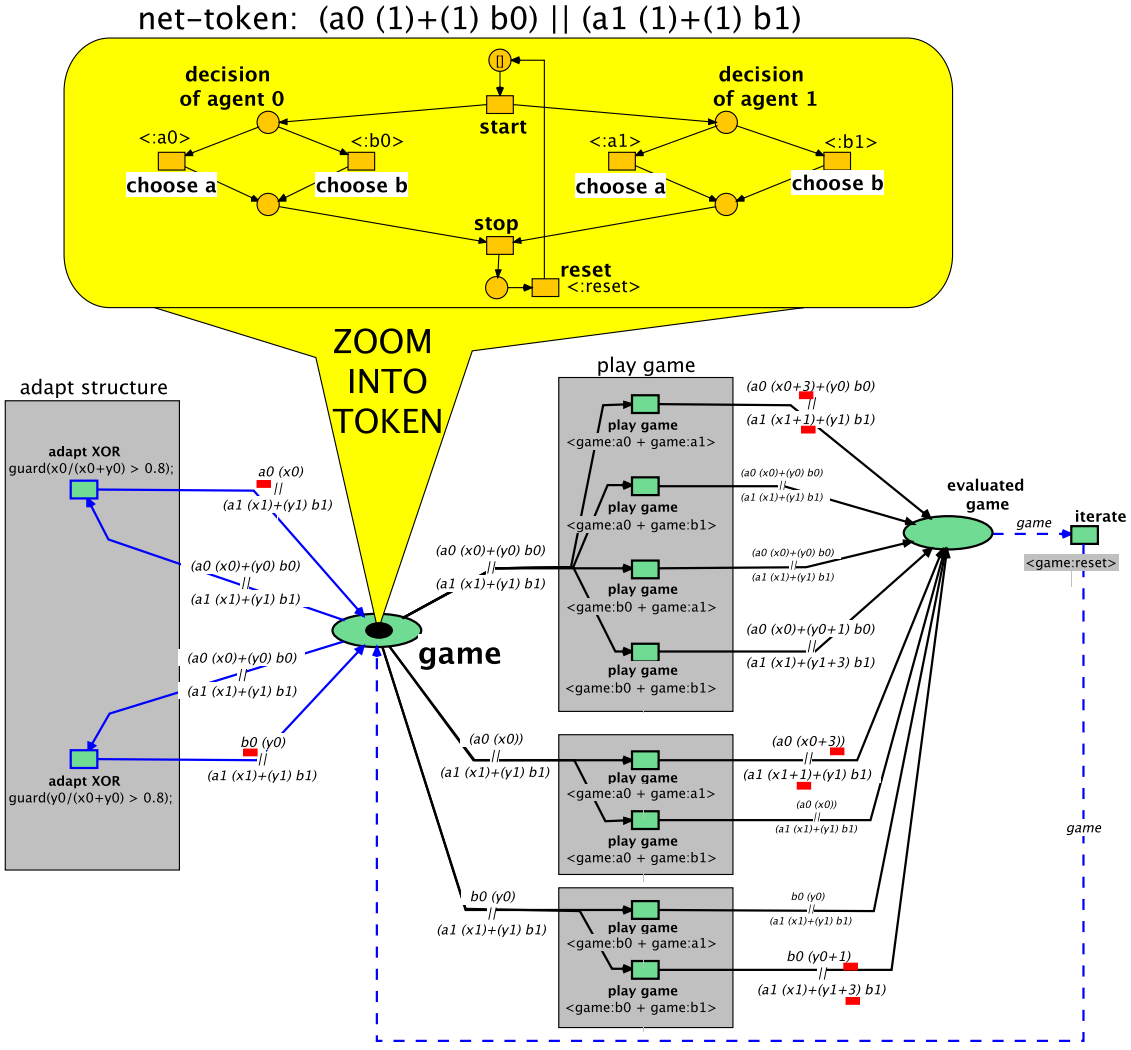


**Figure 5:** The Probabilistic eHornet: System-Net containing the Battle-of-Sexes Interaction (right) and the Structural Adaption Logic (left) (adapted from [14])

in Maude. Consequently, this allows for the utilization of additional tools for generating probabilistic state spaces, specifically, discrete Markov chains, in our stochastic framework.

We would like to illustrate our ideas by an idea first presented in [14]. Our self-adaptive system is based on the battle-of-sexes scenario, which is well known in game theory. Two agents, named 0 and 1, must choose between two actions, labeled $a_i$ and $b_i$, $i = 0, 1$. They receive a positive reward if they choose the same action (i.e. coordinate) and zero otherwise.

In this game, the first agent prefers action $a$, while the second prefers $b$. If we assume that the reward for the preferred outcome is three times higher than for the other, then the game is specified by the payoff matrix:

| | $a_1$ | $b_1$ |
|---|---|---|
| $a_0$ | $(3,1)$ | $(0,0)$ |
| $b_0$ | $(0,0)$ | $(1,3)$ |

Let $(a^{\langle x \rangle} \oplus^{\langle y \rangle} b)$ describe the probabilistic xor choice between action $a$ and $b$ where $\Lambda(a) = x$ and $\Lambda(b) = y$. The object net that models this game is shown as a net-token in Fig. 5; it is a parallel composition (denoted by _||_) of two choices (for some initial values of $x_0, y_0, x_1$, and $y_1$):

$$N_1^{\Lambda} = (a_0{}^{\langle x_0 \rangle} \oplus^{\langle y_0 \rangle} b_0) \parallel (a_1{}^{\langle x_1 \rangle} \oplus^{\langle y_1 \rangle} b_1) \tag{1}$$

The system net observes the decision history and adapts by modifying the rates (cf. eHornet in Fig. 5). We have four transitions named play game on the right side corresponding to the four different

ways of choosing the actions. We give the payoff as a reward signal to the agents. (There may be more appropriate ways to adapt, but for this simple example, we do not care about the efficiency of the learning process.) For example, when the agents play $(a_0, a_1)$, then we update the rates in the workflow by the payoff $(3, 1)$, and we obtain the following.

$$N_2^\Lambda = (a_0{}^{\langle x_0+3\rangle}\oplus^{\langle y_0\rangle} b_0) \parallel (a_1{}^{\langle x_1+1\rangle}\oplus^{\langle y_1\rangle} b_1) \tag{2}$$

We have another source of adaption in the system net: Choices that are chosen quite regularly over a longer time period are converted into fixed structures without choice by the two transitions named adapt XOR on the left-hand side. In this example, the transformation is allowed whenever $a_0$ is chosen in more than 80% of the time. This is expressed by the transition guard $\frac{x_0}{(x_0+y_0)} > 0.8$. Then, we obtain $N_3^\Lambda = a_0 \parallel (a_1{}^{\langle x_1\rangle}\oplus^{\langle y_1\rangle} b_1)$ as the modified net structure. Analogously, whenever $b_0$ dominates. (For simplicity, we omit modifications in the model whenever the second agent has a dominating option.)

In [14] we implemented Hornets in Maude to translate our game-theoretical model into a Discrete Time Markov Chain (DTMC) to establish guarantees for the occurrence of structural modifications within a given number of execution steps.

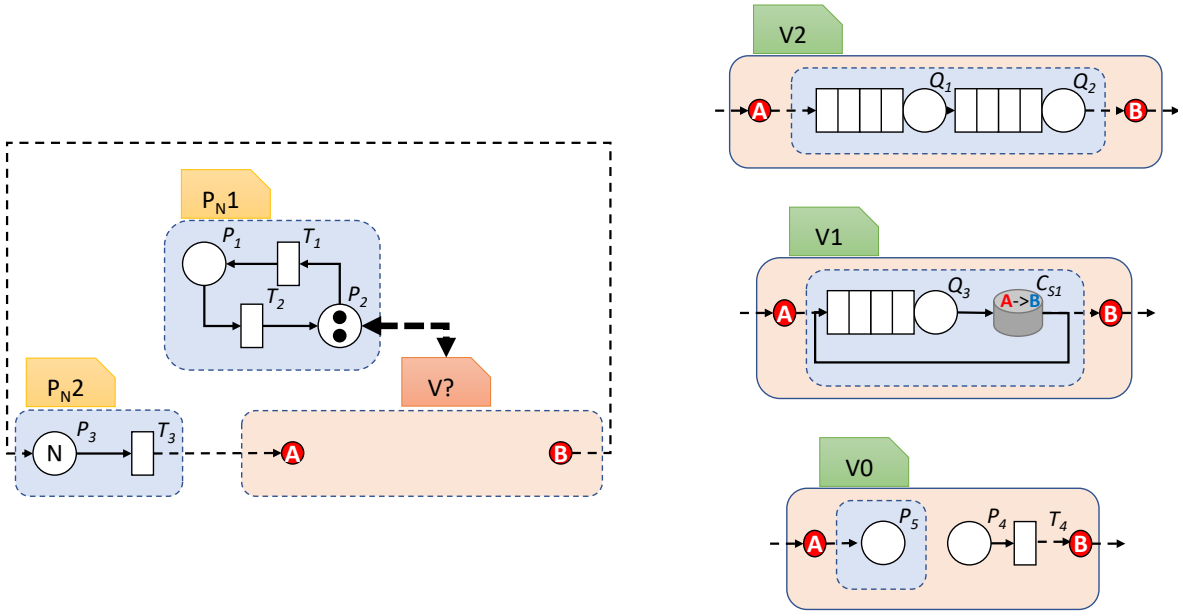


**Figure 6:** A rewritable multi-formalism model.

## 5. Adaptive Multi-formalism

Rewriting also provides solid and powerful support for the analysis of multiformalism models [17]. Multiformalism models typically exhibit complexity, modularity and heterogeneity in the formalisms which define modules: rewriting can be used to add dynamic solution-time features to multiformalism models, by allowing the dynamic generation or adaptation of modules to take into account partial results and ease model parameterization while lowering the complexity of the analysis process. For example, rewriting is being used to incorporate in SIMTHESys [18] the features obtained in OsMoSys [19] by means of a complex orchestration engine [20].

Fig.6 shows a re-writable multiformalism model, which is detailed in Fig.7.

The model represents a two-tier architecture that serves requests generated by $N$ users, whose "think time" is modeled by the place $P_3$ and the transition $T_3$ of the submodel $P_N2$. The two tiers can be deployed on two servers that might experience down periods. In particular, place $P_2$ of the submodel $P_N1$ contains as many tokens as the current number of active servers, $T_1$ models the possibility that a

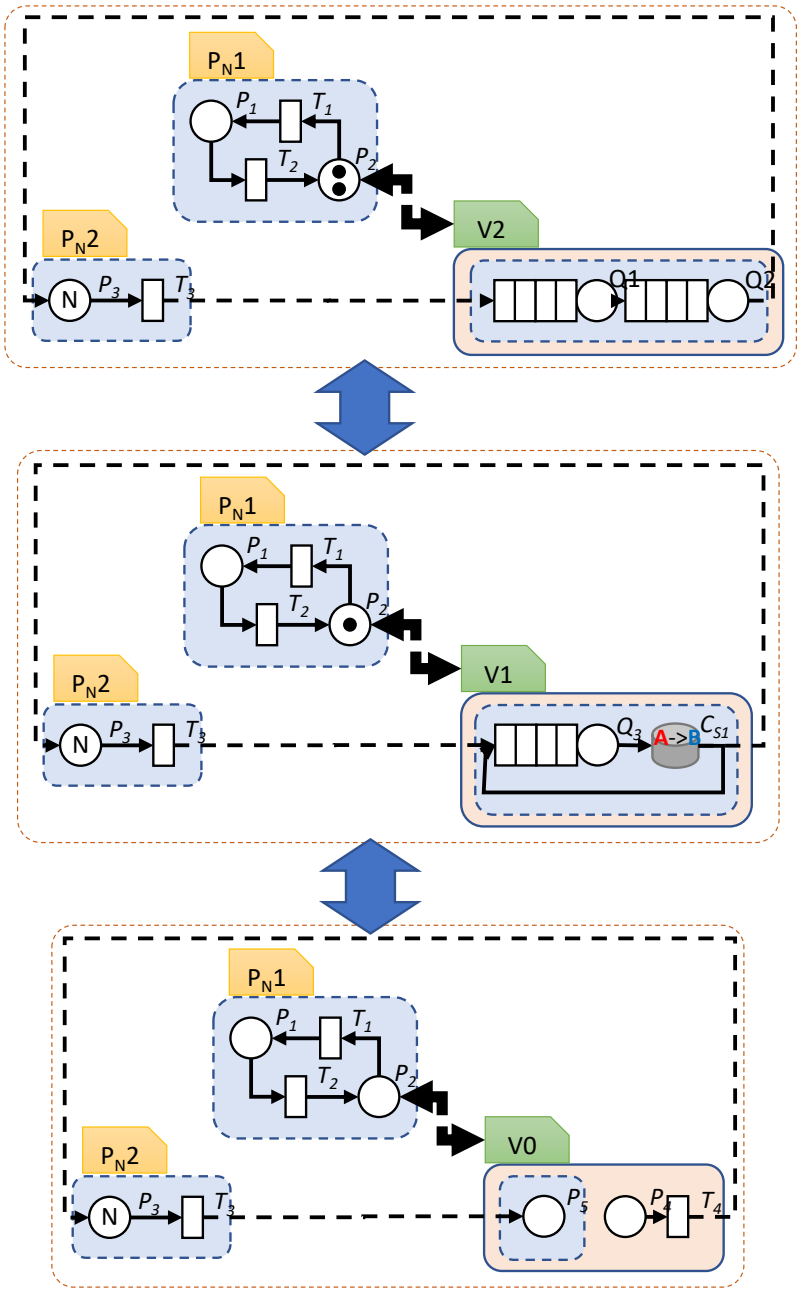


**Figure 7:** The evolution of the proposed rewritable model.

server goes down, $P_1$ accounts for the number of servers that are not available, and $T_2$ models the return in operation of one of the servers. Depending on the marking $P_2$, the submodel identified with $V?$ is replaced by a different implementation. In particular, when both servers are available, the submodel used is $V2$, a queueing network with two stations $Q_1$ and $Q_2$ that represents the two tiers. Each station is characterized by its own service time. When one server goes down, the submodel $V?$ becomes the multiclass queuing network represented in $V1$. In this case, a single multiclass queueing station $Q_3$ is used, where the current tier being executed is represented by the class of the job. When the first-tier jobs end, they change class due to the class switch $C_{S1}$ and re-enters $Q_3$ as costumers of the other class. When the second tier finishes, they can leave the subnetwork. If both servers are down simultaneously, $V?$ is replaced by the Colored Petri Net represented in $V0$. Here, jobs are tokens, colored by their current stage. The place $P_5$ collects interrupted jobs that wait for server availability. The place $P_4$ and the forever disabled transition $T_4$ illustrate the blockage of the system, preventing user activity.

## 6. Conclusions and Future Work

As briefly shown in this paper, rewriting techniques have a great potential as means to improve the efficiency of the analysis of complex stochastic models and to support advanced modeling approaches such as multiformalism modeling, on which future work will focus to verify at which extent this potential will manage to support the implementation of advanced dynamic features for the parametric analysis-time model adaptation and generation.

## Acknowledgments

This work was partially funded by the MUR project "T-LADIES" (PRIN 2020TL3X8X).

## Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

## References

[1] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. M. Oliet, J. Meseguer, C. Talcott, All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic, Lecture Notes in Computer Science, Springer, 2007. doi:10.1007/978-3-540-71999-1.

[2] R. Bruni, J. Meseguer, Generalized rewrite theories, in: J. C. M. Baeten, J. K. Lenstra, J. Parrow, G. J. Woeginger (Eds.), Automata, Languages and Programming, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 252–266. doi:10.1007/3-540-45061-0_22.

[3] J. Meseguer, Twenty years of rewriting logic, The Journal of Logic and Algebraic Programming 81 (2012) 721–781. Rewriting Logic and its Applications.

[4] P. C. Ölveczky, J. Meseguer, Specification of real-time and hybrid systems in rewriting logic, Theoretical Computer Science 285 (2002) 359–405. Rewriting Logic and its Applications.

[5] R. Rubio, N. Martí-Oliet, I. Pita, A. Verdejo, Strategies, model checking and branching-time properties in Maude, Journal of Logical and Algebraic Methods in Programming 123 (2021) 100700.

[6] G. Agha, J. Meseguer, K. Sen, Pmaude: Rewrite-based specification language for probabilistic object systems, Electronic Notes in Theoretical Computer Science 153 (2006) 213–239. Proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005).

[7] R. Bruni, A. Corradini, F. Gadducci, A. Lluch Lafuente, A. Vandin, Modelling and analyzing adaptive self-assembly strategies with Maude, in: F. Durán (Ed.), Rewriting Logic and Its Applications, Springer, Berlin, Heidelberg, 2012, pp. 118–138.

[8] R. Rubio, N. Martí-Oliet, I. Pita, A. Verdejo, Qmaude: Quantitative specification and verification in rewriting logic, in: M. Chechik, J.-P. Katoen, M. Leucker (Eds.), Formal Methods, Springer International Publishing, Cham, 2023, pp. 240–259.

[9] L. Capra, Associating a Markov process with Maude executable modules, in: Proceedings of the 15th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, SciTePress, 2025, pp. 106–116.

[10] L. Capra, Rewriting logic and Petri nets: A natural model for reconfigurable distributed systems, in: R. Bapi, S. Kulkarni, S. Mohalik, S. Peri (Eds.), Distributed Computing and Intelligent Technology, Springer International Pub., Cham, 2022, pp. 140–156.

[11] L. Capra, M. Köhler-Bußmeier, Modular rewritable Petri nets: An efficient model for dynamic distributed systems, Theoretical Computer Science 990 (2024) 114397.

[12] J. Padberg, L. Kahloul, Overview of reconfigurable petri nets, in: R. Heckel, G. Taentzer (Eds.), Graph Transformation, Specifications, and Nets: In Memory of Hartmut Ehrig, Springer, Cham, 2018, pp. 201–222. doi:10.1007/978-3-319-75396-6_11.

[13] L. Capra, M. Gribaudo, A lumped CTMC for modular rewritable PN, in: J. Doncel, A. Remke, D. Di Pompeo (Eds.), Computer Performance Engineering, Springer Nature Switzerland, Cham, 2025, pp. 106–120.

[14] M. Köhler-Bußmeier, L. Capra, Analysing probabilistic Hornets, in: E. Amparore, L. Mikulski (Eds.), Application and Theory of Petri Nets and Concurrency (PETRI NETS 2025), volume 15714 of *LNCS*, Springer, 2025, pp. 287–309. doi:10.1007/978-3-031-94634-9_14.

[15] M. Köhler-Bußmeier, L. Capra, Modelling and simulation of adaptive multi-agent systems with stochastic nets-within-nets, in: Proceedings of the 16th International Joint Conference on Computational Intelligence - Volume 1: ECTA, INSTICC, SciTePress, 2024, pp. 313–320.

[16] M. Köhler-Bußmeier, Hornets: Nets within nets combined with net algebra, in: K. Wolf, G. Franceschinis (Eds.), International Conference on Application and Theory of Petri Nets (ICATPN'2009), volume 5606 of *LNCS*, Springer, 2009, pp. 243–262. doi:10.1007/978-3-642-02424-5_15.

[17] M. Gribaudo, M. Iacono, Theory and application of multi-formalism modeling, 2013.

[18] E. Barbierato, M. Gribaudo, M. Iacono, Modeling hybrid systems in SIMTHESys, Electronic Notes in Theoretical Computer Science 327 (2016) 5 – 25.

[19] V. Vittorini, G. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, DrawNet++: Model Objects to Support Performance Analysis and Simulation of Complex Systems, in: Proc. of the 12th Int. Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation (TOOLS 2002), London, UK, 2002.

[20] F. Moscato, F. Flammini, G. Di Lorenzo, V. Vittorini, S. Marrone, M. Iacono, The software architecture of the OsMoSys multisolution framework, in: VALUETOOLS 2007 - 2nd International ICST Conference on Performance Evaluation Methodologies and Tools, 2007.