Symmetry Preservation in Modular Rewritable Muliformalism Models

Lorenzo Capra[1], Federico Bruzzone[1]

[1]Dipartimento di Informatica, Università degli Studi di Milano, Via Celoria 18, Milano, Italy

Abstract

A recent Maude formalization of multiformalism models for adaptive distributed systems makes it possible to build a symmetry-aware quotient transition system (TS) by means of compositional operators and a systematic node-labeling strategy that reflects the model's hierarchical modular organization. In the stochastic setting, strong bisimilarity on the TS coincides with exact lumpability of the underlying Markov process. We address the challenge of preserving symmetry when Maude rewrite rules perform non-trivial transformations and, using a running example, derive conditions under which such transformations still preserve symmetry.

Keywords

Adaptive multiformalism models, Compositionality, Graph automorphism, Maude

1. Introduction

As systems grow more complex, single-formalism models become either too detailed or too abstract. Multiformalism modeling [1, 2] addresses this by using multiple heterogeneous techniques to analyze quantitative characteristics, choosing the most suitable formalism for each component and often combining them with multi-solution approaches. However, existing approaches offer limited support for system adaptation. Maude [3] is a declarative, expressive, high-performance rewriting-logic language with built-in model checking. Widely used as a logical framework and recently extended with probabilistic analysis features [4], it is a promising multiformalism framework.

We extend this framework with a compositional technique originally developed for modular Rewritable PT nets [5, 6]. Using algebraic operators, we build a symmetry-aware model whose node labels explicitly encode the system's modular structure. This yields a quotient transition system (TS), aligned with Maude's operational semantics, that is strongly bisimilar to the original and, in the stochastic case, coincides with a lumped Markov process. A central challenge is preserving symmetry when Maude rewrite rules perform non-trivial transformations; using a running example, we identify conditions under which these transformations still preserve symmetric structure.

2. Encoding an Heterogeneous Network in Maude

The Maude-based multiformalism framework is defined by a compact hierarchical specification available at https://github.com/lgcapra/rewpt/tree/main/multiformalism. A multiformalism model [7][8] combines multiple components, each described by a (colored) graph-based formalism. Supported formalisms include Petri nets (PNs), automata, activity diagrams, fault trees, queuing networks, and routing or scheduling protocols (e.g., Join-the-Shortest-Queue, JSQ). Vertices of the global graph—the basic building blocks of each component—carry essential structured labels and are called *places*.

Components interact through a distributed state modeled as a commutative monoid. Both the state and place labels are parametric; here, the state is a multiset of places (a PN marking). The hierarchy root is the parameterized module `NETWORK{S :: C-MONOID}`, which specifies the abstract syntax of a multiformalism model by declaring sort `Network`, subsort `Node`, and the associative-commutative juxtaposition operator `_,_` on `Network` elements. A heterogeneous network is thus a multiset of

PNAS'26, International Workshop on Petri Nets for Adaptive Systems, 2026

0000-0002-0877-7063 (L. Capra); 0009-0004-6086-8810 (F. Bruzzone)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-08-03

CEUR-WS.org/Vol-4236/poster2.pdf

distinct nodes interacting via shared state components. Concrete node classes refine `Network` through a standard subsort relation (e.g., `Queue < Node`), allowing systematic specialization of node types while reusing existing component signatures. For example, a P/T `Net` is the juxtaposition of "transitions" (each made up of a label containing, e.g., the stochastic parameters and its adjacency lists defined by multisets of places) [6]; a (multiclass) `Queue` is a list of individual servers; it can have enabling conditions as a prefix. The component (timed) semantics are defined by rewrite rules in Maude system modules.

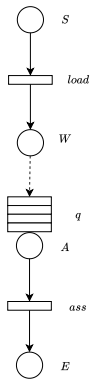
`t("a", 1.5, 0) |-> [1 . p(1) + 2 . p(2), 1 . p(3)] ; t("b", 2.0, 1) |-> [1 . p(3), 1 . p(2) + 1 . p(4)]` --- P/T term
`[2 . p(1), nilP] p(1) @ 1.0 p(2) @ 1.5 > p(3)` --- Queue term

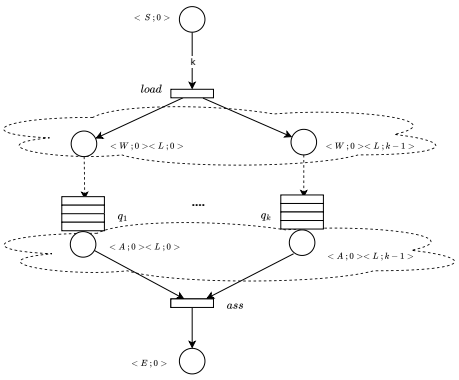
3. Facing the Analytical Complexity: a Modular Approach

We extend to the multiformalism framework [7] a methodology that was initially devised solely for P/T nets [5]. It relies on compositional operators that, through structured node labels, implicitly encode model symmetries (colored graph automorphisms). Here are the key points summarized; we refer to https://github.com/lgcapra/rewpt/tree/main/multiformalism/MODmulti for the formalization

- A `Network`, together with each of its subcomponents, is modeled as a (recursive) monoid structure
- We use place parameterization over the label type to capture and represent symmetries.
- Fundamental structures such as lists, sets, and multisets are monoids; this allows the approach to be applied homomorphically (for instance, the main subcomponent of a JSQ is a set of places).

We illustrate the proposed approach by incrementally building a hybrid manufacturing model that combines P/T nets, multiclass queueing systems, and JSQ. The basic unit, the *line* in Fig. 1-left, has two P/T transitions and one queue. The main structural element, the *production line* (PL) in Fig. 1-right, is obtained by replicating this unit K times and merging selected nodes with algebraic operators that progressively enrich place labels. Labels are lists of (tag, index) pairs. The overall architecture in Fig. 2 is then built by replicating N synchronized PLs, interconnected and controlled by a JSQ component.



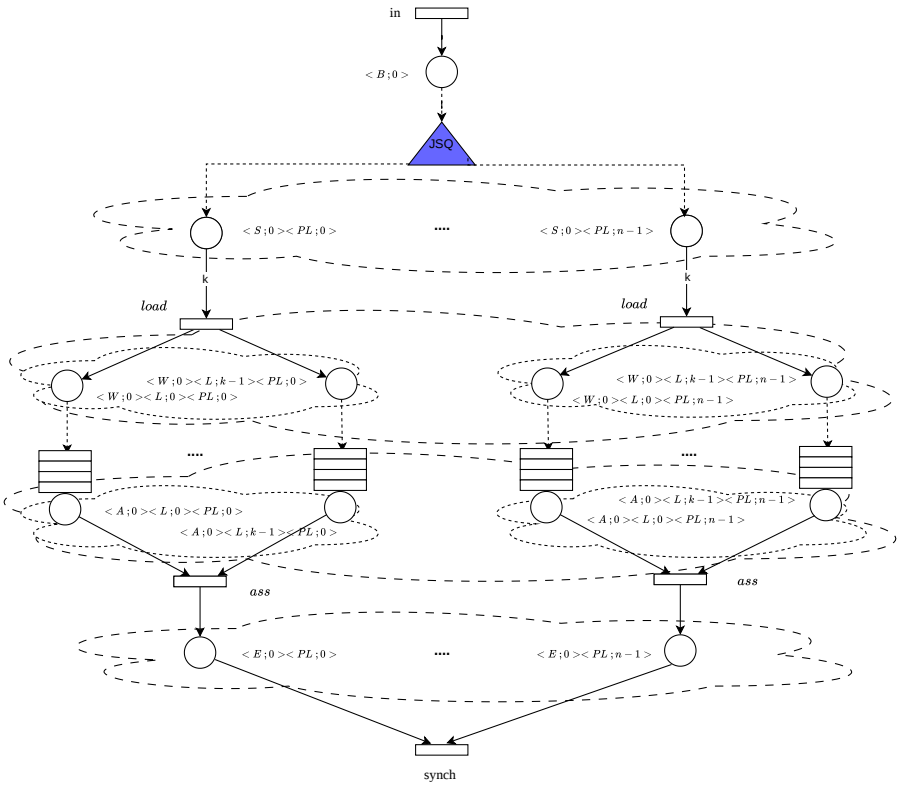



```
mod HNET is
 inc QUEUE-NODE-SYS-MOD .
 inc SPN-NODE-SYS-MOD{String} .
 inc JSQ-NODE-SYS-MOD .
 ops S W A E : -> Place [memo] .
 eq S = place("S") . eq W = place("W") .
 eq A = place("A") . eq W = place("E") .
 ops load, ass : -> Tran [memo] .
 op q : -> Queue [memo] .
 op line : -> Network [memo] .
 eq load = t("load", 0.5, 0) |-> [1 . S, 1 . W] .
 eq ass = t("ass", 0.5, 0) |-> [1 . A, 1 . E] .
 eq q = newElQueue(W, A, 0.1) .
 eq line = load, q, ass .
```

```
...
vars K N : Nat .
ops PL SL : NzNat -> Network .
*** creates K similar lines sharing S and E
eq SL(K) = repl&share(line, K, "L", S U E) .
*** merges transitions with the same label
eq PL(K) = mergeTran(SL(K)) .
...
```

Figure 1: Production line model composed of SPNs and multi-class queues

The structured labels partition the vertex set into subsets (graph-theoretic orbits) that are equivalent under graph automorphisms: any two vertices in the same subset can be mapped to each other by some automorphism. Multiple levels of local symmetry (orbits of smaller subgroups) reflect the model's modular organization. These symmetries are used to build a quotient transition system that is strongly




```
op NPL : NzNat NzNat -> Network .
eq NPL(N, K) = synch(replica(PL(K), N, "PL"), "E") .
op network : NzNat NzNat -> Network [memo] .
eq network(N, K) = newTout(place("B"), "in", 0.25), connect&link(newJsq(place("B")), NPL(N, K)) .
```

Figure 2: A (sub)system consisting of N PLs interconnected using a JSQ (Join the Shortest Queue) routing policy

bisimilar to the original. The construction relies on a simple normalization procedure—combining name abstraction and state minimization—by exchanging label indices (at the same level) within orbits. This method avoids backtracking and is much more efficient than traditional graph canonization.

Definition 1. *Let* $G(\mathrm{n})$ *denote the colored graph associated with a ground term* n *of sort* $\mathrm{Network}$. *The term* n *is said to be well-labeled if every maximal subset of vertices of* $G(\mathrm{n})$ *whose labels share a common (possibly empty) suffix and whose remaining prefix consists of an identical sequence of text tags constitutes an orbit under some automorphism (sub)group of* G.

Network rewrites retaining symmetries The compositional operators introduced in [5] and employed in the example guaranty the well-labeling property for `Network` terms. For instance, the "clouds" depicted in Fig. 2 encompass the orbits of the corresponding graph; in this situation, the label suffix is empty (see Def. 1). More importantly, in Fig. 2, we observe that sub-orbits are embedded within global orbits. This reveals a two-tier structure of symmetry: the global symmetry is associated with the empty suffix, whereas the local symmetry is exemplified by labels carrying the suffix `<PL ; 0>`. Sub-orbits are the orbits of the restricted action of a subgroup of the automorphism group, where all other elements of the original orbit are held fixed.

In a dynamic setting, the rewrite rules that encode structural transformations should preserve the well-labeling property. This central problem is by no means straightforward and is treated only briefly in [5]. Let us instantiate this issue and propose potential solutions based on the use of ad hoc operators within the rewrite rules. Consider the task of removing a single line from each PL. This symmetric operation is defined by a rewrite rule whose right-hand side (enclosed within the normalization operator)

uses a symmetric-removal construct. This operator takes a (list of) tag(s) for the suffix and an ordered pair that uniquely identify the nodes of the line to be removed.

A more likely operation entails the removal of a specific line from a given PL, for instance, in response to a localized fault. Naively removing elements can violate the well-labeling property of nodes: for example, deleting line $k-1$ from the PL with index 0 would cause all nodes in that PL to leave the global orbits in Fig. 2. A remedy is to relabel these nodes by changing their suffix from "PL" to, say, "FPL", which is exactly what the `remove&relab` operator does. Alternatively, one could delete the entire component `< "PL" ; 0 >` and replace it with a new component with one fewer line.

```
op remove : Network Elab List{String} -> Network .
var NET : Network .
crl NET => normalize(remove(NET, < "L" ; k - 1 >, "PL")) if cond(NET) . *** unsafe node removal
op remove&relab : Network NeLab String -> Network . *** safe node removal along with suffix relabeling
crl [rem&rlab] : NET => normalize(remove&relab(NET, < "L" ; k - 1 > < "PL" ; 0 >, "FPL")) if faulty(NET, 0, k - 1) .
```

4. Conclusion

We proposed a Maude-based framework in which heterogeneous models communicate via a distributed state. By using compositional operators and structured node labels, we automatically detect symmetries and construct a lumped CTMC. This paper focuses on the challenge of preserving hierarchical symmetries in compositionally built models when applying adaptive rewrite rules, aiming to formalize the method within a stochastic setting using lumpability-preserving abstractions.

Declaration on Generative AI

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

References

[1] M. Gribaudo, M. Iacono, Theory and application of multi-formalism modeling, 2013. doi:10.4018/978-1-4666-4659-9.

[2] W. Sanders, Integrated frameworks for multi-level and multi-formalism modeling, in: Petri Nets and Performance Models, 1999. Proc. 8th Int. Work. on, 1999, pp. 2–9.

[3] 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.

[4] 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. doi:10.5220/0013567900003970.

[5] L. Capra, M. Köhler-Bußmeier, Modular rewritable petri nets: An efficient model for dynamic distributed systems, Theoretical Computer Science 990 (2024) 114397. doi:10.1016/j.tcs.2024.114397.

[6] L. Capra, Modular stochastic rewritable petri nets, in: P. Cabalar, F. Fabiano, M. Gebser, G. Gupta, T. Swift (Eds.), Proceedings 40th International Conference on Logic Programming, ICLP 2024, University of Texas at Dallas, Dallas Texas, USA, October 14-17 2024, volume 416 of *EPTCS*, 2024, pp. 128–134. doi:10.4204/EPTCS.416.11.

[7] L. Capra, M. Gribaudo, M. Iacono, Reconfigurable multi-formalism models for performance analysis of distributed systems, in: B. Chatterjee, K. Kothapalli, N. Mittal, A. M. Natarajan, D. Singh (Eds.), Distributed Computing and Intelligent Technology, Springer Nature Switzerland, Cham, 2026, pp. 154–170. doi:10.1007/978-3-032-16632-6_10.

[8] E. Barbierato, L. Capra, M. Gribaudo, M. Iacono, Enhancing multiformalism models with rewrite engines, in: L. Carnevali, J. Doncel (Eds.), Computer Performance Engineering, Springer Nature Switzerland, Cham, 2026, pp. 1–16. doi:10.1007/978-3-032-16345-5_1.