Modelling Adaptive Systems with Nets-Within-Nets in Maude

Lorenzo Capra1 and Michael Köhler-Bussmeier2

1 Dipartimento di Informatica, Università degli Studi di Milano, Via Celoria 18, Milan, Italy
2 University of Applied Science Hamburg, Berliner Tor 7, Hamburg, Germany

Keywords: Adaptive Systems, Maude, Nets-Within-Nets.

Abstract: Systems able to dynamically adapt their behaviour gain growing attention to raising service quality by reducing development costs. On the other hand, adaptation is a major source of complexity and calls for suitable methodologies during the whole system life cycle. A challenging point is the system’s structural reconfiguration in front of particular events like component failure/congestion. This solution is so common in modern distributed systems that it has led to defining ad-hoc extensions of known formal models (e.g., the π-calculus). But even with syntactic sugar, these formalisms differ enough from daily programming languages. This work aims to bridge the gap between theory and practice by introducing an abstract machine for the “nets-within-nets” paradigm. Our encoding is in the well-known Maude language, whose rewriting logic semantics ensures the mathematical soundness needed for analysis and an intuitive operational perspective.

1 INTRODUCTION

Systems with the ability to dynamically adapt/reorganise in front of varying execution environments are becoming increasingly popular, to raise service quality without increasing development costs. On the other hand, system adaptation is complex to manage and therefore requires suitable models/methodologies both at design time and runtime. In this context, formal methods can play an important role. (Weyns et al., 2012), (Hachicha et al., 2019) present a survey of formal methods for modelling/analysis of self-adaptive systems. Significant examples are (Iglesia and Weyns, 2015), specifying MAPE-K templates for a family of self-adaptive systems, and (Weyns and Ifikhar, 2022), presenting an end-to-end approach for engineering self-adaptive systems during the life cycle of a feedback loop.

In this paper, we focus on a challenging point, namely, the structural reconfiguration of a system against particular events like component failure/congestion. This solution has become so common in modern distributed systems, that it has led to a specialisation of known formalism, e.g. the π-calculus or nets-within-nets. But even with syntactic sugar, these formalisms are far enough from ‘daily’ programming languages, in which reconfiguration often boils down to add- and delete operations.

This work aims to bridge the gap between theory and practice by presenting an abstract machine for the nets-within-nets approach, used in the last decade to model adaptive and mobile processes/agents. Our formalisation uses the well-known Maude language since specifications given in rewriting logic enjoy the mathematical soundness needed for analysis and provide developers with an intuitive operational perspective (in (Bruni et al., 2012) Maude has been used to formalize adaptive strategies). We focus on the base class of nets-within-nets, namely, Elementary Object Systems (EOS) since they make a reasonable trade-off between expressivity and analysis complexity. This initial formalisation, however, fosters further interesting extensions.

We first briefly introduce PT nets and Nets within Nets in Section 2. Section 3 presents (after a short background) the Maude formalization of EOS, focusing on the key aspects. Section 4 describes our running example: the specification of an adaptable production-line system. We sketch the Maude encoding (which is annexed) and present a few significant examples of formal verification. Section 5 summarises the lessons learned so far and gives insights into current research.
2 BACKGROUND

Petri Nets. PN are a reference model for concurrent systems. We hereafter focus on enriched Place-Transitions nets (PT). A PT net is a kind of bipartite multi-graph \( N := (P, T, I, O, H) \), where: \( P \) and \( T \) are finite, disjoint sets holding the places–state variables, drawn as circles– and the transitions–events changing the state, drawn as bars, respectively; Letting \( \text{Bag}[P] \) be the set of multisets on \( P \), \( \{I, O, H\} : T \to \text{Bag}[P] \) represent the input/output/inhibitor edges, respectively. The distributed state of a PT, or marking, is, in turn, \( m \in \text{Bag}[P] \). PT have intuitive semantics (in the sequel, we use the operator natural extension (e.g., a further extension with an arbitrary nesting of nets is natural) and analysis (e.g., we can use the same structural techniques). In the current implementation, however, net-tokens do not allow a further nesting of nets, i.e., they are PT systems.

In the example given in Figure 1 there are two different types of net-tokens (that we call \( \text{net}_1, \text{net}_2 \) corresponding to colours yellow and grey; The system-net and the net-tokens consist of single transitions; Only weight-one input/output edges are present.

EOS events are nested, accordingly. There may be three different kinds of events:

1. System-autonomous: A system-net transition (assume \( t \)) fires by consistently “moving” net tokens from its preset places \( (p_i, i: 1 \ldots 3) \) to its postset \( (p_j, j: 4 \ldots 6) \), without changing the net tokens marking.

2. Object autonomous: A net-token (assume that of type \( \text{net}_1 \) in the system-net place \( p_2 \) transition (e.g., \( t_1 \)) fires by moving (“black”) tokens from its preset \( (a_1) \) to its postset \( (b_1) \). The net token doesn’t move.

3. Synchronisation (illustrated in Figure 1): Whenever we add matching synchronisation inscriptions, e.g., between the system-net transition \( t \) and the nested transitions \( t_1, t_2 \), then they fire in a fully coordinated way: The net-tokens move from the preset to the postset of \( t \); at the same time, the black tokens inside nested nets move from the pre-set to the post-set of nested transitions \( (t_1, t_2) \). Transitions involved in a synchronisation cannot fire autonomously.

Notice that there may be several independent firing instances for a system-net transition: If many net-tokens of a certain type are in the transition’s pre/post-set, their cumulative marking is computed and distributed on postset places of the same type (possibly after a nested firing step, in the case of a synchronisation). Each possible way of distributing the net-tokens (cumulative) marking results in a separate transition instance. Therefore, there is another possible instance of transition \( t \) (other than that illustrated in Figure 1) in which the net-tokens in \( p_5 \) and \( p_6 \) are swapped. We give further details in Section 3.

Nets-Within-Nets. In recent years, adaptable distributed systems have been deeply studied in the context of the object-net formalism (Köhler-Bußmeier and Rölke, 2004; Köhler-Bußmeier and Heitmann, 2009), which follows the pioneering nets-within-nets model proposed by Valk (Valk, 2003). With object-nets, we mean PN where tokens (graphically denoting a PN marking) are nets in turn, i.e., we have nested markings. The Maude formalisation we are going to present refers to elementary object-net systems (EOS) (Köhler-Bußmeier and Rölke, 2004), the two-level specialisation of object-nets (Köhler-Bußmeier and Rölke, 2004; Köhler-Bußmeier and Heitmann, 2009). HORNets (Köhler-Bußmeier, 2009) are a further extension with algebraic operators making it possible to modify the structure of net tokens through transition firing. For complexity studies, elementary HORNets (Köhler-Bußmeier and Heitmann, 2013) have been introduced, which have a two-level nesting in analogy to EOS. It turns out that elementary HORNets have greater complexity than EOS, though more expressive: On the one hand, most problems (including reachability and liveness) for safe EOS are PSPACE-complete (Köhler-Bußmeier and Heitmann, 2011; Köhler-Bußmeier, 2014). Namely, safe EOS are no more complex than PT nets for this kind of problem. On the other hand, the reachability problem requires exponential space for safe, elementary HORNets (Köhler-Bußmeier and Heitmann, 2016).
Maude. Maude (Clavel et al., 2007) is an expressive, purely declarative language with a rewriting logic semantics (Bruni and Meseguer, 2003). Statements are (conditional) equations (eq) and rules (rl). Both sides of a rule/equation are terms of a given kind that may contain variables. Rules and equations have simple rewriting semantics in which instances of the left-hand side are replaced by corresponding instances of the right-hand side. Maude’s expressivity is achieved through matching modulo associativity; user-definable operator syntax/evaluation strategy; sub-typing; generic types; reflection.

A Maude functional module (fmod) contains only equations and is a functional program defining one or more operations through equations, used as simplification rules. A functional module (with all the imported modules) specifies an equational theory in membership equational logic (Bhouhola et al., 2000). Formally, such a theory is a pair \((\Sigma, E \cup A)\), where \(\Sigma\) is the signature, that is, the specification of all the (sub)sort, kind\(^1\), and operator declarations; \(E\) is the set of (conditional) equations and membership axioms, and \(A\) is the set of operator equational attributes (assoc, comm, ...). The model of \((\Sigma, E \cup A)\) is the initial algebra (denoted \(T_{E \cup A}\)), which is both junk- and confusion-free and mathematically corresponds to the quotient of the ground term algebra. Under certain conditions on \(E\) and \(A\), the final values (canonical forms) of all ground terms form an algebra isomorphic to the initial algebra, i.e., the denotational and operational semantics coincide.

A Maude system module (mod) contains rewrite rules and possibly equations. Rules represent local transitions in a concurrent system. Formally, a system module specifies a generalized rewrite theory (Bruni and Meseguer, 2003), a four-tuple \(R = (\Sigma, E \cup A, \phi, R)\) where \((\Sigma, E \cup A)\) is a membership equational theory; \(\phi\) specifies, for each operator in \(\Sigma\), the frozen arguments; and \(R\) is a set of rewrite rules A rewrite theory specifies a concurrent system. \((\Sigma, E \cup A)\) defines the algebraic structure of the states. \(R\) and \(\phi\) specify the system’s concurrent transitions. The initial model of \(R\) associates to each kind \(k\) a labeled transition system (category) whose states are \(T_{E \cup A,k}\), and whose transitions take the form: \([t] \xrightarrow{[\alpha]} [t']\), with \([t], [t'] \in T_{E \cup A,k}\), and \([\alpha]\) an equivalence class of rewrites modulo the equational theory of proof-equivalence. Executability of system modules maps to ground coherence, which ensures that a rewriting strategy in which terms are first reduced to the canonical form and then rewritten according to the rules is both sound and complete.

3 MAUDE IMPLEMENTATION OF EOS

In this section, we describe the Maude formalization of EOS. It relies on and extends that given in (Capra, 2021; Capra, 2022b), which provides an efficient operational semantics for rewritable PT nets. According to EOS definition, however, dynamic adaptation comes down to net-tokens manipulation. Reconfiguration of the system net is ongoing work. We use a few Maude code excerpts and refer to Figure 1 as a starting example of encoding. The full list of Maude source files is available at github.com/lgcapra/rewpt/new/EOS. We here focus on the non-deterministic firing rule of EOS.

The PT formalization in (Capra, 2021) relies on three generic functional modules, \(\text{BAG}(X)\), \(\text{MAP}+(X,Y)\), \(\text{SET}(X)\) (the last two extensions of built-in modules). They may be arbitrarily nested using a flexible mechanism of parameterized views instantiating the modules’ type parameters. Differently from other Maude formalizations of PNs (Stehr et al., 2001; Padberg and Schulz, 2016), bags are not merely represented as free monoids on sets. Specific bag operators guarantee much more abstraction and

---

\(^1\)A kind is an equivalence class grouping sorts directly or indirectly related by subset ordering; terms in a kind without a specific sort are undefined or error terms.
efficiency: \_._, \_+_, \[_\] \_-_, \_<=_, \>'_, set, \_*_. The first two are constructors, i.e., appear in canonical forms. We thus conveniently represent a bag as a commutative/associative weighted sum, e.g., \(3 \cdot a + 1 \cdot b\). The module MAP+ defines a map term as a “set” of entries built using the associative/commutative juxtaposition \[_\]. Sub-sort Entry of Map has as a unique constructor \[_\rightarrow\_\]. Module MAP+ supplies, among others, a predicate verifying the uniqueness of a map’s keys which is widely exploited in membership equations implementing consistency checks.

PT nodes are denoted by indexed-labelled terms, e.g., \(p(2,"net1"), t(1,"sys")\). A transition’s incidence matrix is a triplet (constructor \[_\rightarrow\_]\) defined in module IMATRIX of terms of sort Bag\{Place\}. The modules PT-NET and PT-SYS hold the signature of a PT net/system. A net is a term of sort Map\{Tran,Imatrix\} (renamed Net), i.e., a semicolon-separated set of entries \(t(k,"lab") \rightarrow [i,o,h]\), each entry being a term in the sub-sort ImatrixT < Net. A PT system is the juxta-position \(\_\rightarrow\_\) of Bag\{Place\} -\(\rightarrow\) (System)) of a Net term and Bag\{Place\} term representing the marking. Using a \textit{kind} as the operator’s range means that it defines a partial function: the net sub-term must be a consistent, non-empty map. A membership axiom characterizes well-formed System terms. This solution is a trade-off between rewriting efficiency and coding elegance.

A system module (PT-EMU) formalizes the operational semantics of PT systems by exploiting the algebraic representation of PT nets. The conditional rule firing implements the PT firing rule. All the involved operators are bag-operators. The matching equation \((t \leftarrow t')\) in rule’s condition ensures compactness.

\begin{verbatim}
mod PT-EMU is
  pr PT-SYS.
  var T:Tran.
  var I O H:S:Bag\{Place\}.
  var N\':Net.
  crl[ firing]: NS\rightarrow NS+O-I if T \rightarrow [I,O,H]; N':= N\cap T \rightarrow [I,S\cap H']\cap S.
endm
\end{verbatim}

The model-specific part consists of a system module importing PT-EMU and PT-SYS and containing two zero-arity operators of range Net and Bag\{Place\}.

### 3.1 EOS Specification

The EOS specification extends and reflects in part that of (rewritable) PT systems. A few functional modules specify the EOS algebraic structure. A system module (EOS-EMU) specifies the (non-trivial) EOS operational semantics. Some auxiliary operators are needed to mimic the inner steps of transition firing, in particular, the computation of firing instances of a system-net transition and the consequent (non-deterministic) distribution of net-tokens on its post-set. Finally, a specific system module instantiates a given EOS model. For the reader’s convenience, we mix the textual description with a few code excerpts. We use Figure 1 to illustrate the main concepts.

#### EOS Net

A term describing an EOS net (module EOS-EMU) is the empty juxta-position of three sub-terms of sorts Net, Map\{String,Net\} (renamed NeTypeS) and Map\{Tran,Map\{String,Bag\{Tran\}\}\} (renamed Syncmap). The resulting whole term is of \textit{kind} (Sysnet) (because of possible inconsistencies among its components). The 2nd and 3rd sub-terms specify the types of net tokens and the synchronization between system- and object-net transitions (for each object-net type), respectively. These sub-terms are equipped with ad-hoc operators and separately defined (modules NET-TYPES and SYNCHRO). Note that, according to EOS definition, a system-net transition may synchronize with multiple occurrences of object-net transitions.

The type of net-tokens a system-net place may hold is associated with the place’s label in NeTypeS sub-term. If there is no association, the system-net place may only hold “black-tokens”. Instead, the three categories of events possible in a EOS meet the following conventions.

2. Object autonomous: nested transitions for which the predicate \textit{op synchronized} : Syncmap Tran \rightarrow Bool evaluates to false (this predicate checks that a given transition appears among the values that are maps, in turn- of Syncmap sub-term).

A membership-axiom connotes well-formed EOS as terms of sort Sysnet. Predicate coherent\((Sy, Ty)\) checks that every nested transition occurring in Syncmap belongs to the corresponding net-type.

\begin{verbatim}
cmb N Ty Sy : Sysnet if welldef(N) and-then welldef(Ty) and-then not(repeatedKeys(Sy)) and-then coherent(Sy, Ty)
\end{verbatim}
Using a uniform syntax for the system-net and net-tokens is convenient in terms of description/algebraic manipulation and significantly enhances EOS expressivity. Furthermore, the adopted signature may be easily adapted to support an arbitrary nesting of nets.

**EOS System.** The EOS dynamics is mimed using a structured state representation, in which the basic generic types are reciprocally nested. A term of sort $\text{Map}\{\text{Place}, \text{Bag}\{\text{Bag}\{\text{Place}}\}\}$ specifies an EOS marking as a map from system-net places to nested multisets of net-token places: a term of sort $\text{Bag}\{\text{Place}\}$, indeed, represents a marking of the net-token type associated to a certain system-net place, its multiplicity in the top multisets the number of net-tokens in the system-net place with that marking. The nil ground term (representing the empty $\text{Bag}\{\text{Place}\}$) may also denote -without any ambiguity- anonymous tokens in untyped system-net places. For example, the EOS marking in Figure 1 is described by the term $(\text{net1}^*, \text{net2}^*)$ representing the empty juxtaposition of a net-token type and a certain system-net place, its multiplicity in the top multisets the number of net-tokens in the system-net place with that marking.

A marked EOS is formally described by the empty juxtaposition of a $\text{Sysnet}$ sub-term and a $\text{Map}\{\text{Place}, \text{Bag}\{\text{Bag}\{\text{Place}}\}\}$ sub-term (module EOSYS). Due to possible inconsistencies, the operator’s arity is the kind $[\text{Eosystem}]$. As usual, we use a membership axiom to connote terms of sort $\text{Eosystem}$ as those in which every system-net place is a key in the EOS marking sub-term. No check is done on net-token places, for the sake of efficiency and coherence with the fact that (in a mutable context) they may contain isolated places.

\begin{align*}
\text{var S: Sysnet. var M: Map\{Place, Bag\{Bag\{Place}\}\}.}
\text{c mb S M: Eosystem if not(repeatedKeySTM()) and -then places(net(S)) subset keySet(M);}
\end{align*}

In the perspective of allowing an arbitrary nesting of nets, generalized multisets (whose elements may be bags in turn) should be used instead of nested bags.

**EOS Operational Semantics.** According to the architecture of EOS, two rewrite rules specify the firing of system-net transitions —taking account of synchronizations— and autonomous transitions in nested nets. Both rely on the PT firing rule.

The former has some trickiness due to the intrinsic nondeterminism: Similarly to High-Level PN transitions, a system-net transition $t$ may fire in different instances in a EOS marking $m$ (inhibitor edges currently have a merely numerical interpretation). An instance of $t$ has the same algebraic representation as $m$, i.e., it is a $\text{Map}\{\text{Place}, \text{Bag}\{\text{Bag}\{\text{Place}}\}\}$ term whose map’s keys are the $t$’s preset. In other words, an instance of $t$ is a sub-multiset of $m$.

The system-net firing rule exploits two main operators (module EOSYS):

\begin{align*}
\text{op firingmap: Eosystem} & \rightarrow \text{Map\{Tran, Set\{Map\{Place, Bag\{Bag\{Place}\}\}\}\}}. \\
\text{op firings: ImatrixTMap\{Place, Bag\{Bag\{Place\}\}\} Synmap} & \rightarrow \text{Set\{Map\{Place, Bag\{Bag\{Place\}\}\}\}}.
\end{align*}

firingmap computes the enabled firing instances for every system-net transition, taking account of synchronizations. It builds on a few auxiliary operators defined in generic modules $\text{BAG-}\text{SPLIT, MAP-}\text{PROD}$, in particular:

\begin{align*}
\text{op split: Bag\{X\} Nat} & \rightarrow \text{Set\{Bag\{X\}\}}. \\
\text{op prod: Map\{X, Set\{Y\}\}} & \rightarrow \text{Set\{Map\{Place, Bag\{Bag\{Place\}\}\}\}}.
\end{align*}

The former divides a bag into sub-bags of given cardinality, the latter performs a kind of Cartesian product of maps having sets as values. In the event of synchronization, $t$’s instances are filtered according to the enabled synchronized nested transitions.

The firing of an instance of $t$ may be nondeterministic, i.e., result in alternative multisets of net-tokens to distribute on $t$’s post-set (Sect. 1). Operator firings calculates this set for a system-net transition instance (the transition’s incidence matrix and the synchronization map are the other arguments). It builds on:

\begin{align*}
\text{op distribute: Bag\{Place\} Bag\{Place\} } & \rightarrow \text{Set\{Map\{Place, Bag\{Bag\{Place\}\}\}\}};
\end{align*}

that distributes the (pre-calculated) cumulative marking obtained by an instance of $t$ (1st arg) to $t$’s post-set (2nd arg), assuming that the two arguments refer to system-net places of the same type. This operation is tricky and reduces to enumerate the partitions of a multiset.

The system module $\text{EOS-EMU}$ formalizes the EOS operation semantics.

\begin{align*}
\text{mod EOS-EMU is}
\text{pr EOSYS.}
\text{inc PT-EMU.}
\text{var FM: Map\{Tran, Set\{Map\{Place, Bag\{Bag\{Place\}\}\}\}\}\}}. \text{*** whole firing map}
\text{var TI: Entry\{Tran, Set\{Map\{Place, Bag\{Bag\{Place\}\}\}\}\}\}}. \text{*** transition instance}
\text{var NeFS: NeSet\{Map\{Place, Bag\{Bag\{Place\}\}\}\}\}}. \\
\text{var FS: Set\{Map\{Place, Bag\{Bag\{Place\}\}\}\}\}}. \text{*** output firing set}
\text{vars I O MM’: Map\{Place, Bag\{Bag\{Place\}\}\}\}}. \text{*** firing instance/EOS marking}
\end{align*}
vars N N' : Net . var T : Tran . var Ty : NeTypeS . var Sy : Syncmap . vars B' : Bag{Place} . var B2 : Bag{Bag{Place}} .

We refer to the EOS with the term eosnet m0 using simple aliasing. By running the inline reduce command of the Maude interpreter

Maude> red firinginstances eosnet m0, t(1,"sys")

we get two enabled instances for the system-net transition t(1,"sys"), in accordance with the two net-tokens of type "net1" in place p(1,"net1") (Figure 1).

The following command searches for reachable final (!) states.

Maude> search eosnet m0 => X:Eosystem .

The command has four matches which express the non-determinism of EOS transition firing: For each instance of t(1,"sys") there are indeed two possible output-markings, as the ways to distribute the net-token of type "net2" on places p5, p6.

### 4 THE PRODUCTION LINE

We consider a FMS with two production lines as an example. This scenario has been used as a case study for Rewritable PT Nets in (Capra, 2021). Here, we will model the scenario in terms of nets-within-nets.

In the scenario we have raw material and two operations, t1 and t2, working on pieces of those. We have two production lines (i.e. robots), both being capable of performing t1 and t2. We assume that the two lines have different qualities for these operations, i.e., it is better to execute t1 in line 1 and t2 in line 2. This is the standard production plan. It is possible that during the execution one of the two lines gets faulty. A double failure has a negligible probability and is not modelled here.

The scenario is well suited for a EOS-model as we have two obvious levels in our model: the production site and the production plans. The model is specified in the syntax of the RENEW tool (Kummer et al., 2004).

The system level shown in Figure 2 describes the two production lines, the execution life cycle of the production plan, and the dropout of lines.

In the system net, the place p0 indicates normal operation. In this mode the transition f0 takes two tokens (i.e. the raw material) from place p1 and activates the normal production plan, i.e. it generates a net-token of type plan via the inscription x:new plan. The two production lines are shown as blocks. Their transitions are side-conditions for the place production plan. They synchronise via the channel inscriptions of the form x:line1,t1().

These channels have a counterpart in the net-tokens (e.g. the net plan – shown in Figure 3 – has
the corresponding inscription \textit{line1.t1()}. Therefore, we have a synchronous firing of the transition in the production line and in the production plan.

Net-tokens describe the different production plans: That for normal operation is given as the net plan shown in Figure 3, while we also have two fall back plans \textit{fallback 1} and \textit{fallback 2} for the case of dropouts. Specifically, the given production plan from Figure 3 specifies a synchronisation via \textit{line1.t1()} and via \textit{line2.t2()}, so we will execute \textit{t1} in line 1 and \textit{t2} in line 2. When the production plan is finished we synchronise via \textit{do.t3()} and delete the net token, i.e. the plan. For simplicity, the scenario restarts this production via the transition \textit{t4} which regenerates two raw materials again on place \textit{p1}. This reset makes the main loop of the model live.

Now, we will look at the adaption part and how the model preserves liveness even in the event of dropouts. On the left part of Figure 2 (the yellow nodes), we have the adaption part: Transitions \textit{t5} and \textit{t6} model the dropout of one production line. Whenever we have a drop the standard production plan is not executable anymore. Therefore, the transitions \textit{fall back to line 1/2} withdraws them. After a dropout, the places \textit{p7} and \textit{p8} indicate which of the two lines is down. According to this information, we will select the appropriate fallback plan, i.e. in the case of a dropout in line one (\textit{p7} is marked) we switch to the fallback plan 2 and vice versa. The two fallback blocks have almost the same structure as the original block. The only difference is that we generate a different net-token via \textit{x:new fallback1} in the case of the side condition \textit{p8} and via \textit{x:new fallback2} in the case of the side condition \textit{p7}. To avoid lots of crossing arcs we use the RENEW feature of so-called virtual places, i.e., places with a double outer line. They denote a reference to the original place.

The two fallback plans \textit{fallback 1} and \textit{fallback 2} are not shown here as they look almost identical to the basic plan in Figure 2. The only difference are the used channels: The plan \textit{fallback 1} works only with the production line 1, which is expressed by using the channels \textit{line1.t1()} and via \textit{line1.t2()}. Analogously for \textit{fallback 2}.

The modular \textit{Maude} specification of the adaptable Production Line may be found in github.com/lgcapra/rewpt/tree/main/new/EOS. We got it by composing atomic sub-nets through the associative/commutative net-juxtaposition operator (‘;’).

4.1 Model’s Analysis

Below we give some evidence of formal verification with the only intent of showing the potential of an approach based on \textit{Maude}. Since we specify adaptation in a way to preserve the liveness of the production process, we focus on this kind of property. We use the two basic analysis tools, namely the \texttt{reduce} command, which prints out the canonical form of a ground term, and the \texttt{search} state-space explorer (both available inline).

As usual, we manage wordy terms using intuitive aliasing. For example, the \texttt{net} term describes the system-net component of the EOS, whereas the \texttt{eosnet} term includes the net-token description and synchronizations. Terms \texttt{m0} and \texttt{eosm0} denote the initial marking of the net (seen as a PT system) and of the EOS.

We can use the \texttt{reduce} command both to unfold these aliases and to perform a preliminary formal validation. If it assigns the canonical form a specific sort (i.e., \texttt{Eosystem}), it means that the initial term represents a well-defined EOS specification:

\begin{verbatim}
reduce in PLINE-EOS : eosnet eosm0.
result Eosystem: {... unfolded term}
\end{verbatim}

The next two \texttt{searches} operate on the PT-system we obtain from the system-net by replacing net-tokens with anonymous tokens. One of the advan-
Figure 3: The Object Net plan modelling the standard Production Plan.

The advantages of EOS, indeed, is that we can separately consider and analyse the two EOS levels.

The first one verifies a state invariant which characterizes the production plan’s life-cycle, in a configuration where place $p_1$ initially holds four tokens. The search has indeed no matches (solutions), consistently with the fact that we search for a counterexample. It is worth noticing two things. An implicit outcome of the command is that the PT system derived from the system net is bounded because the state space turns out to be finite. The invariant is actually structural, i.e., it holds for any configuration with $2 \times K$ tokens initially in place $p_1$ ($K$ being the model’s parameter). We cannot prove parametric invariants using state-space exploration.

The second search, instead, checks the absence of final (dead) states. Again, there are no matches.


Maude> search in pline-SYS : net m0 =>! X: System .

Analogously, we can start searching from an EoSystem term specifying the whole EOS. The following (unmatched) search is much more significant than the previous ones because it formally verifies that the whole EOS (including the nested nets) is deadlock-free (and implicitly, bounded). Both the state space and the execution time grow up.

Maude> search in pline-EOS : eosnet eosm0 =>! E:EoSystem.

Table 1 reports the performances of the last search, as the system’s parameter varies. The data refer to an Intel Core i7-6700 equipped with 32 GB of RAM.

Model-checking techniques suffer from the possible state-space explosion, which may be only alleviated using some heuristics, e.g., by carrying out bounded searches (the search command has a number of options) or making some abstractions (as we did in the first two searches). The canonization technique for rewritable PT systems recently presented in (Capra, 2022a) might represent an effective approach to scale the model’s complexity. Using this technique (which requires a slight adaption to work with EOS) the state sizes reported in Table 1 would approximately halve, due to the PL symmetry. Overall, it would be possible to model-check more realistic configurations with many PL working in parallel and synchronizing at some point, which would be otherwise intractable. Another drawback of model-checking is that, in general, we cannot infer parametric outcomes (not depending on the initial state).

PT structural analysis, which considers the PT graph structure, is an effective alternative (complementary) to state-space inspection. We may use it to infer parametric outcomes, e.g., structural state-invariants (semiflows). Structural analysis of EOS looks promising because it may take advantage of the fact that the types of net tokens flowing through the system net places are finite.

5 SUMMARY AND OUTLOOK

In this paper we have defined a Maude representation of nets-within-nets, more concretely: EOS.

What are the Strengths of the Approach? The Maude formalization presented here is an extension of (rewritable) PT specification (Capra, 2021). Therefore a lot of code could be reused, which is beneficial for the implementation’s reliability and efficiency as well.

The formalization preserves central design issues of EOS by supporting a uniform view: The system-net and net-tokens have the same structure in Maude.

<table>
<thead>
<tr>
<th>$K$</th>
<th># states</th>
<th># rewrites</th>
<th>time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>262</td>
<td>19801</td>
<td>38</td>
</tr>
<tr>
<td>5</td>
<td>2802</td>
<td>180539</td>
<td>453</td>
</tr>
<tr>
<td>10</td>
<td>13932</td>
<td>995724</td>
<td>3192</td>
</tr>
<tr>
<td>20</td>
<td>104007</td>
<td>19737494</td>
<td>56090</td>
</tr>
<tr>
<td>50</td>
<td>4186132</td>
<td>111564504</td>
<td>906253</td>
</tr>
</tbody>
</table>
which is essential as the same is true for EOS. This aspect is relevant when the architecture is extended from the two-level case to an unbounded nesting of net-tokens in a marking (Köhler-Büßmeier and Heitmann, 2009) or HORNETS (Köhler-Büßmeier, 2009). Standard Maude facilities for formal verification (e.g., state-space search and model-checking) may be used with no additional costs.

Additionally, the EOS firing rule is defined in a way that moving net-tokens around cannot be distinguished from moving ordinary tokens in PT. Therefore, we can easily define abstractions on the system’s state (e.g., forgetting about the marking of net-tokens), which is essential to perform state space exploration efficiently.

Our approach fosters model extensions. Passing to Object-nets should be almost for free (using general Bags, with an arbitrary nesting). While the extension from EOS to HORNETS is a rather large step, which involves new constructs like net-algebra operators, the formalizing of HORNETS in Maude seems very simple. We may easily go one step forward, towards “rewritable” EOS, where the structure of both the system net and of net-tokens may change over time.

Advantages of the Proposed Approach.

• The approach builds on an existing Maude formalization of (rewritable) PT nets: massive code-reuse, implementation reliability, efficiency, uniform view (system-nets and net-tokens have the same structure), easy abstractions (the system-net may be mimed as an ordinary net)
• The built-in Maude facilities for formal verification (e.g., state-space search and model-checking) may be exploited.
• The approach promotes updates and extensions. A natural extension is the use of PT nets (with inhibitor arcs both) at system-net level and at net-token level. Passing to Object-nets should be almost for free (using general Bags, with an arbitrary nesting level, would be required). Formalizing Hornets should be simple as well, even if the system-net definition would be closer to an algebraic PN.
• We may easily go one step forward, towards “rewritable” EOS where both the structure of the system net and that of net-tokens may change over time.

What Was Complex? The most challenging aspect of the formalisation was the integration of the so-called firing modes. Roughly speaking, the firing rule of a system-autonomous event in an EOS collects the tokens of all net-tokens coming from the system net places in the preset. When the system net transition fires it distributes all these tokens on freshly generated net-tokens in the postset. The firing rule allows any of these possible distributions – an aspect which requires some tricky handling of the binding in Maude. In summary:

• Formalizing the EOS firing mechanism shows some trickiness.
• The use of nested generic types in Maude requires some expertise.

Limitations. The current formalisation fulfils the requirement that it provides a link to the world of programming. But we have to admit that like in any algebraic specification, terms describing EOS may be wordy, structurally complex and (consequently) difficult to read and manage. A systematic aliasing mechanism might greatly help a modeller. Also, syntactic sugar would sweeten the approach. An automated translation from a high-level (graphical) description of EOS to the corresponding Maude module would be highly desirable.

Ongoing Work. In this paper, we were concerned with the Maude encoding of EOS. Our main motivation for this is to obtain a representation closer to the usual programming language world. Our intention is also to benefit from the advantages of a formal specification, i.e. the possibility to apply analysis techniques more easily. In the case of Maude the first idea is to apply state-space techniques, like LTL model checking. We also like to integrate structural PN techniques for EOS (Köhler-Büßmeier and Moldt, 2009).

For the analysis of EOS, we need to struggle with scalability as the state space of EOS grows even worse than that of PT nets. Possible approaches are the canonization of net-tokens and the use of abstractions to obtain condensed state spaces. Canonization of net-tokens allows us to capture symmetries in their behaviour. For that purpose, we have slightly adapted the canonization technique for rewritable PT systems defined in (Capra, 2022a), fully integrated into Maude. According to this technique, a PT system is seen as a coloured graph and put into a “minimal” isomorphic form by incrementally permuting node indices. The latter approach (state abstraction) can be expressed quite easily in Maude by adding additional equations on markings.
REFERENCES


Köhler-Bußmeier, M. and Heitmann, F. (2011). Live-