Process Refinement and Asynchronous Composition with Modalities

Dorsaf Elhog-Benzina[1], Serge Haddad[1], and Rolf Hennicker[2]

[1] LSV, CNRS & Ecole Normale Supérieure de Cachan
94235 CACHAN, 61 avenue du Président Wilson, France
{*elhog*, *haddad*}@lsv.ens-cachan.fr
[2] Institut für Informatik Universität München
Oettingenstraße 67 D-80538 München, Germany
hennicke@pst.ifi.lmu.de

Abstract. Modal transition system specifications has been first introduced in [6] as transition systems with modalities *may* and *must* that explicitly distinguish between allowed transitions and required ones. Use of modalities is a way to loose specifications definitions as a set of allowed transitions is specified and the decision is delayed to an advance step. Thus this model can be commonly used for both specifications and implementations: a possible implementation is a specification in which all decisions has been taken i.e all transitions are required. Thus a possible implementation is considered as a simulation of the original specification. Modal specifications can be combined together to model complex systems due to composition process. Existing works are limited to synchronous composition as they deal with only finite systems. Actually, asynchronous composition introduces a delay between the actions of sending and receiving a message between the communication partners which involves infinite systems. We propose in this paper a Petri net based approach to deal with asynchronous communication. We first syntactically define modal Petri nets to model infinite systems and introduce modal Petri net asynchronous composition. Then we propose a procedure to decide whether, in one hand, an implementation simulates a specification and on an another hand whether a composition specification simulates a third one.

Key words: Modal specification, implementation, simulation, Petri nets, infinite systems, asynchronous composition

1 Introduction

Specification in component-based software products. In component based software, *specification* is a mandatory phase of components' life cycle. It aims to produce a formal description, a *specification*, of the component's desired properties and behavior. A specification can be presented either in terms of transition systems or in terms of logic, which cannot be processed by a machine. Thus an *implementation phase* is required to produce concrete executable programs.

Modal specifications. Modal specification have been introduced in [6] as a common formal model for specifications and implementations. A modal specification explicitly distinguish between required actions and allowed ones. Required actions, denoted with the modality *must*, are compulsory in all possible implementations while allowed actions, denoted with the modality *may*, are not necessary. An implementation is seen as a specific specification in which all actions are required. Modal specification is adequate for loose specification as decisions could be delayed to a later step of the component lifecycle. Two different modal formalisms have been adopted in the literature, the first is a system-based model introduced in [3] and defines *modal transition system* while the second is a language-based model introduced in [10] and defines *modal language specification*. For consistency reason, both formalisms assume that every required action is also an allowed one. In order to be able to express conflicting behaviors of a single specification the authors consider specifications that relax the consistency condition by defining *mixed specification* (or *pseudo-modal specification*) for which the existence of an implementation is not guaranteed and if it does not exist, the model is said to be *inconsistent*.

Modal specification refinement and composition. A consistent modal specification admits an implementation that could be obtained by transforming some allowed actions into required ones or omitting other allowed actions. A transformation step is called a *refinement* and produces a specification that is more precise i.e. with less possible implementations. It follows that the set of implementations of a refinement is included in the set of possible implementations of the original specification. Usually specifications are not exploited separately but are combined in order to describe complex systems. Thus composition operators have been proposed to combine abstract system definitions and reflect composition of independently developed implementations. Raclet et al. have proposed a *synchronous product* for pseudo-modal language specification composition in [10] and Larsen et al. have proposed a composition operator for modal transition systems in [5]. The operator proposed by Larsen et al. was supposed to garantee the property of independent implementability of the composed systems but Raclet et al. have found a counterexample and proposed a correction in [11]. Along with the composition, Raclet et al. proposed a *quotient* operator which is the dual of the composition operator. Given a global specification of a two-components system, the aim of quotienting operation is to construct the specification of one of the component assuming that the other specification is realized by a given component.

Computational issues. While dealing with modal specifications, three main decision problems have been raised:

- (C) Consistency problem: deciding whether a specification is consistent i.e deciding whether it admits an implementation. Consistency is guaranteed for modal transition systems by definition [6] as every required transition is also an allowed one. Whereas in case of mixed transition systems, consistency decision problem is EXPTIME-complete in the size of the specification [2].

A better result is obtained with modal languages since a polynomial time algorithm has been proposed in [10].

- (CI) Common implementation: deciding whether $k > 1$ modal specifications have a common implementation. Deciding common implementation of k modal transition systems is PSPACE-hard in the sum of the sizes of the k specifications [1] while it is EXPTIME-complete when dealing with mixed modal transition systems[2].
- (TR) Thorough refinement: deciding whether one modal specification S thoroughly refines another modal specification S' i.e deciding whether the set of possible implementations of S is included in the set of possible implementations of S'. The problem is also PSPACE-hard if S and S' are modeled with modal transition systems and is EXPTIME-complete when they are modeled with mixed modal transition systems.

Limits of the existing formalisms. Both formalisms consider finite-state systems. This restriction limits the existing approaches to synchronous composition. In fact asynchronous composition introduces a delay between the actions of sending and receiving a message between the communication partners. Therefore, assuming for instance that the sender is always active while the receiver is always blocked, naturally leads to infinite state systems.

Our contribution. We aim in this paper to deal with asynchronous composition of modal specifications while keeping most of the problems decidable. Petri nets seem to be an appropriate formalism to our needs. Automata with queues might also be an alternative for modeling infinite state systems but all significant problems (e.g. the reachability problem) are known to be undecidable. We also followed a language approach for the advantages mentioned above and use *deterministic* Petri nets as a device to generate deterministic languages in the same way as Raclet et al used deterministic finite transition systems as a device to generate regular languages. We are mainly interested in the following issues:

1. decide whether a given implementation is a refinement of a given specification i.e decide whether the implementation simulates the specification,
2. decide whether the composition of two specifications is *consistent*, i.e. can be turned in a modal specification.

In order to do this, we introduce *modal Petri nets* by adding modalities to Petri net transitions and extend the modal language refinement to languages generated by Petri nets. We then define an asynchronous composition operator.

Outline of the paper. We proceed by reviewing in section 2 modal language specification formalism and its associated notions of refinement and composition. In section 3 we first review basic definitions related to Petri net theory and then introduce our formalism of modal Petri net, refinement relation and asynchronous composition operator. Finally in section 4, we describe our decision algorithms for closure under composition and refinement relation.

2 Modal Language Specifications

Modal specification was first introduced by Larsen and Thomsen in [6] with finite state modal transition systems by defining restrictions on specifications transitions by the mean of *may* (allowed) and *must* (required) modalities. This notion has then been adapted by Raclet in his PhD thesis who applied it to regular languages. Finite transition systems are low level models based on states, which limits the level of abstraction of the specification. They also lead to state number explosion while composing systems with an important number of states. Moreover, the complexity of the decision problems discussed above are better with modal languages than with modal transition systems. Besides, modal refinement is sound and complete with the language-based formalism while it is non-complete with transition system based formalism [4]. So a language approach is more convenient to deal with modal specification issues. Next we review the definition of modal language specification and its related notions of refinement and synchronous composition as introduced in [10].

Definition 1 (Modal language specification). *A* modal language specification S *over an alphabet* Σ *is a triple* $\langle L, may, must\rangle$ *where* $L \subseteq \Sigma^*$ *is a prefix-closed language over* Σ *and* $may, must : L \to P(\Sigma)$ *are partial functions. For every trace* $u \in L$*,*

- $a \in may(u)$ *means that the action* a *is allowed after* u*,*
- $a \in must(u)$ *means that the action* a *is required after* u*,*[1]
- $a \notin may(u)$ *means that* a *is forbidden after* u*.*

The modal language specification S *is* consistent *if the following two conditions hold:*
(C1) $\forall\ u \in L\ must(u) \subseteq may(u)$
(C2) $\forall\ u \in L\ may(u) = \{a \in \Sigma \mid u.a \in L\}$

Example 1. Let us consider the example of a message producer and a message consumer represented in figure 2.

In the producer system, transition $s_0 \overset{in}{\to} s_1$ is allowed but not required (dashed line) while transition $s_1 \overset{m}{\to} s_0$ is required (continuous line). In consumer model, all transitions are required. The languages associated with the the producer is $L \equiv (in.m)^* + in.(m.in)^*$. The associated modal language specification is then $\langle L, may, must\rangle$ with:

- $\forall\ u\ \in\ (in.m)^*\ must(u) = \emptyset \wedge may(u) = \{in\}$
- $\forall\ u \in in.(m.in)^*\ must(u) = \{m\} \wedge may(u) = \{m\}$

Similarly, the modal language specification associated with the consumer is $\langle (m.out)^* + m.(out.m)^*, may, must\rangle$ with:

[1] If $must(u)$ contains more than one element, this means that any correct implementation must have after the trace u (at least) the choice between all actions in $must(u)$.

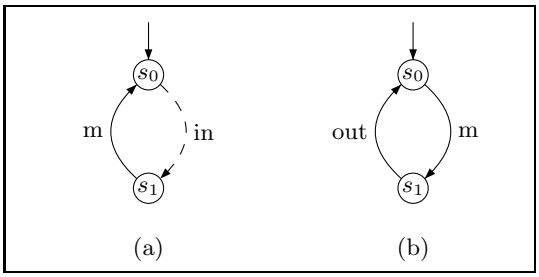

Fig. 1. Modal transition systems for a producer (a) and a consumer (b)

- $\forall\ u\ \in\ (m.out)^*\ must(u) = \{m\} \wedge may(u) = m$
- $\forall\ u \in m.(out.m)^*\ must(u) = out \wedge may(u) = out$

A particular class of modal language specifications can be represented by modal finite state transition systems. These languages are called *regular modal specification.*

Definition 2 (Regular modal specification). *A modal language specification* $\mathcal{S} = \langle L, must, may\rangle$ *over an alphabet* Σ *is regular if its language* L *is regular and for all event* $a \in \Sigma$*, the language*

$$L_a = \{u \in L \mid a \in must(u)\} \text{ is regular}$$

Modal language specifications are related by a refinement relation that translates the degree of specialization. One can obtain a possible refinement by either removing some allowed events or changing them to required events. So we review the formal definition of modal language specification refinement.

Definition 3 (Modal language specification refinement).
Let $\mathcal{S} = \langle L, may, must\rangle$ *and* $\mathcal{S}' = \langle L', may', must'\rangle$ *be two consistent modal language specifications.* $\mathcal{S}$ *is* a modal language specification refinement *of* $\mathcal{S}'$*, denoted by* $\mathcal{S} \sqsubseteq_{\mathbf{LS}} \mathcal{S}'$*, if:*

- $L \subseteq L'$,
- *for every* $u \in L$, $must'(u) \subseteq must(u)$, *i.e every required action after the trace* u *in* L' *is a required action after* u *in* L.

The *synchronous composition* (synchronous product) of modal language specifications is a generalization of the synchronous product of languages over the same alphabet.

Definition 4 (Synchronous composition of modal language specifications). *The composition of two modal language specifications* S_1 *and* S_2 *is the modal language specification* $S = S_1 \otimes S_2$ *whose associated language* $L = L(S_1) \cap L(S_2)$ *and:*

$$\forall\ u\ \in\ L(S_1) \cap L(S_2), \begin{cases} must_{S_1 \otimes S_2}(u)\ =\ must_{S_1}(u) \cap\ must_{S_2}(u) \\ may_{S_1 \otimes S_2}(u)\ =\ may_{S_1}(u) \cap\ may_{S_2}(u) \end{cases}$$

3 Modal Petri Nets and their Generated Language Specifications

The previous approach deals with finite state systems and finite modal transition systems have been used as a device to generate modal languages. In the same perspective we introduce modal Petri nets to model infinite state systems and use it as a device to generate languages which are our basic tool for the refinement and composition issues. Before introducing modal Petri nets and their generated languages we review basic definition of Petri net theory.

3.1 Basics of Petri Net Theory

Definition 5 (Marked labeled Petri net). *A marked labeled Petri net over an alphabet Σ is a tuple $\mathcal{N} = (P, T, W^-, W^+, \lambda, m_0)$ where:*

- *P is a finite set of places,*
- *T is a finite set of transitions with $P \cap T = \emptyset$,*
- *W^- (W^+ resp.) is a matrix indexed by $P \times T$ with values in $\mathbb{N}$;*
  W^- (W^+ resp.) is called the backward (forward) incidence matrix,
- *$\lambda : T \to \Sigma$ is a transition labeling function, and*
- *$m_0 : P \mapsto \mathbb{N}$ is an initial marking.*

A marking *is a mapping $m : P \mapsto \mathbb{N}$. The labeling function is extended to sequences of transitions $\sigma = t_1 t_2 ... t_n \in T^*$ where $\lambda(\sigma) = \lambda(t_1)\lambda(t_2)...\lambda(t_n)$. For each $t \in T$, ${}^\bullet t$ ($t^\bullet$ resp.) denotes the set of* input (output) places *of t. i.e. ${}^\bullet t = \{p \in P \mid W^-(p,t) > 0\}$ ($t^\bullet = \{p \in P \mid W^+(p,t) > 0\}$ resp.). Likewise for each $p \in P$, ${}^\bullet p$ ($p^\bullet$) denotes the set of* input (output) transitions *of p i.e. ${}^\bullet p = \{t \in T \mid W^+(p,t) > 0\}$ ($p^\bullet = \{t \in T \mid W^-(p,t) > 0\}$ resp.).*

In the sequel marked labeled Petri nets are simply called Petri nets.

The previous definition is limited to the system structure, but transitions are active elements of a Petri net: they consume tokens from theirs input places and produce others in output places. This dynamicity is captured by specifying a marking that characterizes the system state by describing the token distribution over the net places and the action of firing a transition that moves the system from state to another.

Definition 6 (Firing rule). *Let $\mathcal{N}$ be a Petri net. A transition $t \in T$ is firable in a marking m, denoted by $m[t\rangle$, iff $\forall p \in {}^\bullet t,\ m(p) \geq W^-(p,t)$. The set of firable transitions in a marking m is defined by $firable(m) = \{t \in T \mid \forall p \in {}^\bullet t.\ m(p) > 0\}$. For a marking m and $t \in firable(m)$, the firing of t from m leads to the marking m', denoted by $m[t\rangle\ m'$ and defined by $\forall p \in P, m'(p) = m(p) - W^-(p,t) + W^+(t,p)$.*

Definition 7 (Firing sequence). *Let N be a Petri net with the initial marking m_0. A finite sequence $\sigma \in T^*$ is firable in a marking m and leads to a marking*

m', also denoted by $m[\sigma\rangle m'$, iff either $\sigma = \epsilon$ or $\sigma = \sigma_1.t$ with $t \in T$ and $\exists m_1$ such that $m[\sigma_1\rangle m_1$ and $m_1[t\rangle m'$. We write $Reach(N, m_0) = \{m \mid \exists \sigma \in T^*$ such that $m_0[\sigma\rangle m\}$ for the set of reachable markings.

The reachable markings of a Petri net correspond to the reachable states of a modeled system. Since the capacity of places are not restricted, the set of the reachable markings of the Petri nets considered here may be infinite. Thus, Petri nets can model infinite state systems. Now, let us define the language generated by a Petri net.

Definition 8 (Petri net language). *Let $\mathcal{N}$ be a Petri net over the alphabet Σ. The language generated by $\mathcal{N}$ is:*

$$\mathcal{L}(\mathcal{N}) = \{u \in \Sigma^* \mid \exists \sigma \in T^* \text{ and } m \text{ s.t } \lambda(\sigma) = u \text{ and } m_0[\sigma\rangle m\} \ .$$

Since no accepting states are defined the language $\mathcal{L}(\mathcal{N})$ is prefix-closed.

Some issues with Petri nets, such as language inclusion problem, are not decidable in the general case while they are with deterministic Petri nets. So we limit ourselves to *deterministic* Petri nets.

Definition 9 (Deterministic Petri net and its generated language). *A labeled Petri net with labeling function $\lambda : T \to \Sigma$ is* deterministic*, if for each reachable marking m and for each label $a \in \Sigma$ there is at most one transition $t \in T$ with $\lambda(t) = a$ such that $t \in firable(m)$. The language generated by a deterministic Petri net is also called* deterministic Petri net language *and fulfills the following property. For each $u \in \mathcal{L}(\mathcal{N})$ there is a single sequence $\sigma \in T^*$ such that $\lambda(\sigma) = u$*

Next we define the synchronous composition of two Petri nets that we will use further.

Definition 10 (Synchronous composition of two Petri nets). *The synchronous composition of two Petri nets $\mathcal{N} = (P, T, W^-, W^+, \lambda, m_0)$ and $\mathcal{N}' = (P', T', W'^-, W'^+, \lambda', m'_0)$ is a Petri net $\mathcal{N}'' = ((P'', T'', W''^-, W''^+, \lambda'', m''_0)) = \mathcal{N} \parallel \mathcal{N}'$ where,*

- *$P'' = P \uplus P'$ (we assume that P and P' are disjoint),*
- *$T'' = \{(t, t') \in T \times T' \mid \lambda(t) = \lambda'(t')\}$,*
- *W''^- is defined for each $p \in P''$ and $(t, t') \in T''$ by:*

$$W''^-(p, (t, t')) = \begin{cases} W^-(p, t) & \text{if } p \in P \\ W'^-(p, t') & \text{otherwise} \end{cases}$$

- *W''^+ is defined for each $p \in P''$ and $(t, t') \in T''$ by:*

$$W''^+((t, t'), p) = \begin{cases} W^+(t, p) & \text{if } p \in P \\ W'^+(t', p) & \text{otherwise} \end{cases}$$

- *$\lambda''((t, t')) = \lambda(t) = \lambda'(t')$,*
- *m''_0 is defined, for each place $p \in P''$, by*

$$m''_0(p) = \begin{cases} m_0(p) \text{ if } p \in P \\ m'_0(p) \text{ otherwise} \end{cases}$$

3.2 Modal Deterministic Petri Nets

Here we introduce modal deterministic Petri nets which extend, similarly to modal language specifications, deterministic Petri nets with modalities *may* and *must* on its transitions.

Definition 11 (Modal deterministic Petri net). *A* modal deterministic Petri net $\mathcal{M}$ *over an alphabet* Σ *is a pair* $\mathcal{M} = (\mathcal{N}, T_\Box)$ *where* $\mathcal{N} = (P, T, W^-, W^+, \lambda, m_0)$ *is a deterministic Petri net over* Σ *and* $T_\Box \subseteq T$ *is a set of* must (required) *transitions. The set of* may (allowed) *transitions is the set of transitions* T*.*

Example 2. Let us consider the same example of a message producer and a message consumer.

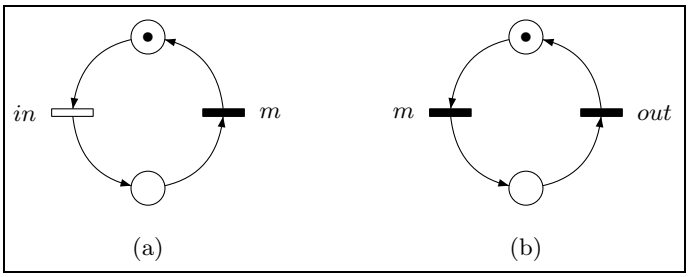

Fig. 2. Modal interface Petri nets for a producer (a) and a consumer (b)

The consumer may receive an input *in* (white transition) but must produce a message m (black transition). The consumer must receive an input message m and produce an output *out*.

Any modal deterministic Petri net gives rise to the construction of a modal language specification. The definition of this language is based on the definitions of modal language specifications (definition 1) and the generated language of a deterministic Petri net (definition9).

Definition 12 (Modal deterministic Petri net language specification). *Let* $\mathcal{M} = (\mathcal{N}, T_\Box)$ *be a modal deterministic Petri net over an alphabet* Σ*.* $\mathcal{M}$ *generates the* modal language specification $\mathcal{ML}(\mathcal{M}) = \langle \mathcal{L}(\mathcal{N}), may, must \rangle$ *where:*

- $\mathcal{L}(\mathcal{N})$ *is the language generated by the deterministic net* $\mathcal{N}$*,*
- $\forall u \in \mathcal{L}(\mathcal{N})$*, let* m *be the marking and* $\sigma \in T^*$ *be the sequence of transitions such that* $\lambda(\sigma) = u$ *and* $m_0[\sigma\rangle m$*,*
  - $may(u) = \{a \in \Sigma \mid \exists t \in firable(m)$ *such that* $\lambda(t) = a\}$
  - $must(u) = \{a \in \Sigma \mid \exists t \in firable(m)$ *such that* $t \in T_\Box$ *and* $\lambda(t) = a\}$*.*

Lemma 1. *Any modal deterministic Petri net language specification is a consistent modal language specification.*

The proof of this lemma is straightforward.

3.3 Refinement and asynchronous composition

Likewise modal language specifications introduced in section 2, modal deterministic Petri net language specifications are related by a refinement relation.

Definition 13 (Modal Petri net language specification refinement). *Let $\mathcal{S} = \langle \mathcal{L}, may, must \rangle$ and $\mathcal{S}' = \langle \mathcal{L}', may', must' \rangle$ be two modal language specifications generated respectively by a modal deterministic Petri net $\mathcal{M}$ and a modal Petri net $\mathcal{M}'$. $\mathcal{S}'$ is a* modal language specification refinement *of $\mathcal{S}$ denoted by $\mathcal{S}' \sqsubseteq_{\mathbf{LS}} \mathcal{S}$, if,*

- *(1) $\mathcal{L}' \subseteq \mathcal{L}$*
- *(2) for every $u \in \mathcal{L}'$, $must(u) \subseteq must'(u)$, i.e every required action after the trace u in $\mathcal{L}$ is a required action after u in $\mathcal{L}'$.*

Given two modal language specifications $\mathcal{S} = \langle \mathcal{L}, may, must \rangle$ and $\mathcal{S}' = \langle \mathcal{L}', may', must' \rangle$ generated respectively by the modal deterministic Petri net $\mathcal{M} = (P, T, W^-, W^+, \lambda, m_0)$ and the Petri net $\mathcal{M}' = (P', T', W'^-, W'^+, \lambda', m'_0)$, we aim to decide whether $\mathcal{S}'$ is a modal language specification refinement of $\mathcal{S}$. The decision problem will be discussed in the next section.

Before defining asynchronous composition of modal Petri nets we should specify a composability condition to ensure that the composed components are able to collaborate correctly.

Definition 14 (Composability condition). *Let $\Sigma = in_\Sigma \cup out_\Sigma$ and $\Sigma' = in_{\Sigma'} \cup out_{\Sigma'}$ be two alphabets partitioned into inputs and outputs. Σ and Σ' are composable if $\Sigma \cap \Sigma' \subseteq (in_\Sigma \cap \ out_{\Sigma'}) \cup \ (in_{\Sigma'} \cap \ out_\Sigma)$.*
Two Petri nets are composable if their alphabets are composable.

Observe that for composable alphabets $in_\Sigma \cap in_{\Sigma'} = \emptyset$, $out_\Sigma \cap out_{\Sigma'} = \emptyset$.

Definition 15 (Composition of compatible alphabets). *Let $\Sigma = in_\Sigma \cup out_\Sigma$ and $\Sigma' = in_{\Sigma'} \cup out_{\Sigma'}$ be two composable alphabets. The composition of Σ and Σ' is the alphabet $\Sigma'' = in_{\Sigma''} \cup out_{\Sigma''} \cup int_{\Sigma''}$ where:*

- $int_{\Sigma''} = \{*a \mid * \in \{?, !\} \text{ and } a \in \Sigma \cap \Sigma'\}$
- $out_{\Sigma''} = (out_\Sigma \setminus in_{\Sigma'}) \uplus (out_{\Sigma'} \setminus in_\Sigma)$
- $in_{\Sigma''} = (in_\Sigma \setminus out_{\Sigma'}) \uplus (in_{\Sigma'} \setminus out_\Sigma)$

Definition 16 (Concrete asynchronous composition). *Let $M_1 = (N_1, T_{1_\Box})$, $N_1 = (P_1, T_1, W_1^-, W_1^+, \lambda_1, m_{1_0})$ be a modal Petri net over Σ_1 and $M_2 = (N_2, T_{2_\Box})$, $N_2 = (P_2, T_2, W_2^-, W_2^+, \lambda_2, m_{2_0})$ be a modal Petri net over Σ_2. M_1 and M_2 are* composable *if $P_1 \cap P_2 = \emptyset$, $T_1 \cap T_2 = \emptyset$ and if Σ_1 and Σ_2 are composable. In this case, their* concrete asynchronous composition *M_c, also denoted by $M_1 \otimes M_2$, is the modal Petri net over Σ_c defined as follows:*

- $P_c = P_1 \uplus P_2 \uplus \{p_a \mid a \in \Sigma_1 \cap \Sigma_2\}$ *(the p_a are new place identifiers)*
- $T_c = T_1 \uplus T_2$ *and* $T_{c,\Box} = T_{1_\Box} \uplus T_{2_\Box}$

– W_c^- *(resp.* W_c^+*) is the* $P_c \times T_c$ *backward (resp. forward) incidence matrix defined by:*
  - *for each* $p \in P_1 \cup P_2,\ t \in T_c$,

$$W_c^-(p,t) = \begin{cases} W_1^-(p,t) \text{ if } p \in P_1 \text{ and } t \in T_1 \\ W_2^-(p,t) \text{ if } p \in P_2 \text{ and } t \in T_2 \\ 0 \qquad \text{otherwise} \end{cases}$$

$$W_c^+(p,t) = \begin{cases} W_1^+(p,t) \text{ if } p \in P_1 \text{ and } t \in T_1 \\ W_2^+(p,t) \text{ if } p \in P_2 \text{ and } t \in T_2 \\ 0 \qquad \text{otherwise} \end{cases}$$

  - *for each* $p_a \in P_c \setminus \{P_1 \cup P_2\}$ *with* $a \in \Sigma_1 \cap \Sigma_2$ *and for each* $t \in T_i$ *with* $i \in \{1,2\}$,

$$W_c^-(p_a,t) = \begin{cases} 1 \text{ if } a = \lambda_i(t) \in in_{\Sigma_i} \cap out_{\Sigma_j} \text{ with } i \neq j \\ 0 \text{ otherwise} \end{cases}$$

$$W_c^+(p_a,t) = \begin{cases} 1 \text{ if } a = \lambda_i(t) \in in_{\Sigma_j} \cap out_{\Sigma_i} \text{ with } i \neq j \\ 0 \text{ otherwise} \end{cases}$$

– $\lambda_c : T_c \to \Sigma_c$ *is defined, for all* $t \in T_c$ *and for* $i \in \{1,2\}$, *by*

$$\lambda_c(t) = \begin{cases} \lambda_i(t) \quad \text{if } t \in T_i,\ \lambda_i(t) \notin \Sigma_1 \cap \Sigma_2 \\ ?\lambda_i(t) \text{ if } t \in T_i,\ \lambda_i(t) \in in_{\Sigma_i} \cap out_{\Sigma_j} \text{ with } i \neq j \\ !\lambda_i(t) \text{ if } t \in T_i,\ \lambda_i(t) \in in_{\Sigma_j} \cap out_{\Sigma_i} \text{ with } i \neq j \end{cases}$$

– m_{c_0} *is defined, for each place* $p \in P_c$, *by*

$$m_{c_0}(p) = \begin{cases} m_{1_0}(p) \text{ if } p \in P_1 \\ m_{2_0}(p) \text{ if } p \in P_2 \\ 0 \qquad \text{otherwise} \end{cases}$$

Example 3. The composition of the producer and consumer Petri nets of figure 2 is shown in figure 3. Its alphabet has the input set $\{in\}$, the output set $\{out\}$ and the set of internal actions is $\{?m, !m\}$. The Petri net composition describes an infinite state system whose generated language is no more regular.

The next proposition shows that the concrete asynchronous composition is well defined.

Proposition 1. *The concrete asynchronous composition of two modal specifications is a*

The following proposition We are now looking for an abstraction of the composed system which hides the internal asynchronous communication.

Definition 17. *Let* $M_1 = (N_1, T_{1_\Box})$, $N_1 = (P_1, T_1, W_1^-, W_1^+, \lambda_1, m_{1_0})$ *be a modal Petri net over* Σ_1, $M_2 = (N_2, T_{2_\Box})$, $N_2 = (P_2, T_2, W_2^-, W_2^+, \lambda_2, m_{2_0})$ *be a modal Petri net over* Σ_2. *Assume that* M_1 *and* M_2 *are* composable *and denote* M_c *by their asynchronous composition. Then their abstract asynchronous composition of* M_1 *and* M_2, *denoted by* M_a, *is defined by:*

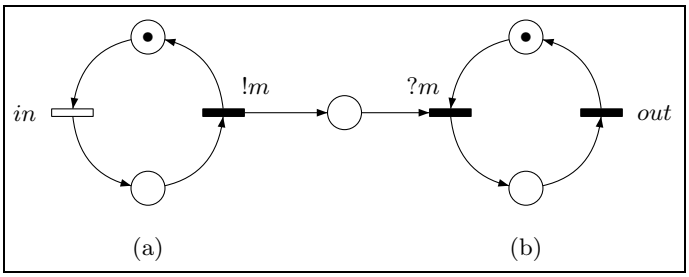

Fig. 3. Composition of the producer and consumer Petri nets

- *The language of M_a is the projection of the language on the inputs and the outputs of M_c.*
- *Given a word w of this language and a letter a, a belongs to $must(w)$ iff for every sequence in M_c, σ such that $m_0[\sigma\rangle m$, $\lambda(\sigma) = w$, there is a "must" transition $t \in T_{1_\Box} \uplus T_{2_\Box}$ with $\lambda(t) = a$ and $m[t\rangle$.*

The previous definition is more involved since asynchronous composition requires to hide the transitions related to communications and thus yield a non deterministic behaviour. The interpretation of the *must* mappings is that an event a is required after a word w of the language if **for every** behaviour whose observation is w a "must" transition labelled by a may be fired. This is consistent with the intuitive meaning of a must event.

Decision problems. Since asynchronous composition is a way to obtain a more refined specification and is not *a priori* a deterministic Petri net, we want solve the following problems (to be handled in the next section).

- Is a Petri net deterministic? Since we have now silent transitions, we need to adapt definition 9.
- Whatever the answer to the first question, the composition of two (or more) component specifications is a more precise specification than a global one. So given a Petri net $\mathcal{N}$ and a deterministic Petri net $\mathcal{N}'$ does $\mathcal{N}$ refine $\mathcal{N}'$?

For example, figure 4 presents an initial specification of producer/consumer system and the composition depicted in figure 3 refines this specification while not being deterministic.

4 Decision Algorithms

4.1 Deterministic Petri net membership

First we introduce a new definition of deterministic Petri net that takes into account the silent transition. On says that a transition is visible if its labels belongs to Σ. One denotes ε the empty word which now may label transitions.

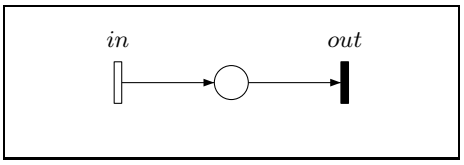

Fig. 4. A more abstract specification of a producer/consumer system

Definition 18 (Deterministic Petri net with silent transitions). *A labeled Petri net with labeling function $\lambda : T \to \Sigma \uplus \{\varepsilon\}$ is* deterministic*, if:*

- *It is* non divergent*. There does not exist a finite sequence σ and an infinite sequence σ' with $\lambda(\sigma') = \varepsilon$ such that $m_0[\sigma\sigma'\rangle$.*
- *It is* marking deterministic*. Let σ and σ' be two firing sequences $m_0[\sigma\rangle m$ and $m_0[\sigma\rangle m'$ such that $\lambda(\sigma) = \lambda(\sigma')$ and the last transitions of σ and σ' are visible. Then: $m = m'$.*

Discussion. The first condition means that the net must after firing a finite number of silent transitions either stop or fire a visible transition. This condition is relevant for a specification since divergence is related to an undesirable behaviour. The second condition is a relaxation of the previous definition: we do not require uniqueness of the firing sequence that generates a given word. Rather we require that the *visible* markings (the ones reached after the firing of a visible transition) reached by two sequences related to the same word must be the same.

Proposition 2. *Let $\mathcal{N}$ be Petri net, then it is decidable whether $\mathcal{N}$ is deterministic. Furthermore if $\mathcal{N}$ is deterministic, the maximal number of silent transitions consecutively fireable is computable.*

Proof. In order to decide whether $\mathcal{N}$ is divergent, we build a net $\mathcal{N}'$ as follows.

- First we add three places *run*, *check* and *count*. *run* has initially one token while *check* and *count* are empty.
- Then we duplicate every silent transition t becoming t_r and t_c. t_r includes an additional loop with *run* (i.e. $W^-[run, t_r] = W^+[run, t_r] = 1$) , t_c includes an additional loop with *check* and produces a token in *count*.
- A visible transition t includes an additional loop with *run*.
- Finally one adds a transition *switch* that transfers the token from *run* to *check*.

The behaviour of $\mathcal{N}'$ is the following one: it mimics $\mathcal{N}$ but can switch at any time to a behaviour when only silent transitions of $\mathcal{N}$ are possible and their firing are counted in place *count*.

Then one builds $\mathcal{G}_c(\mathcal{N}')$ the covering graph of $\mathcal{N}'$. If in an ω-marking m of $\mathcal{G}_c(\mathcal{N}')$, the component of place *count* is ω then $\mathcal{N}$ is divergent. Otherwise, the

maximal value occurring in this component corresponds to the maximal number of silent transitions consecutively fireable.

In order to check whether the net is marking deterministic, one builds a net $\mathcal{N}''$ as follows.

- We duplicate every place p becoming p_l (P_l denotes the set of these places) and p_r (P_r denotes the set of these places) with the same initial marking. Furthermore we add two places *stable* and *unstable*. *stable* has initially one token while *unstable* is empty.
- Then we duplicate every silent transition t into four transitions $t_{l,s}$, $t_{l,u}$ $t_{r,s}$, $t_{r,u}$. Transitions $t_{l,u}$ and $t_{l,s}$ have for inputs and outputs the ones of t with places p_l substituted for p. $t_{l,u}$ includes an additional loop with *unstable* while $t_{l,s}$ transfers the token from *stable* to *unstable*. Transitions $t_{r,u}$ and $t_{r,s}$ have for inputs and outputs the ones of t with places p_r substituted for p. $t_{r,u}$ includes an additional loop with *unstable* while $t_{r,s}$ transfers the token from *stable* to *unstable*.
- For every pair of visible transitions t, t' such that $\lambda(t) = \lambda(t')$, one defines two transitions $(t,t')_s$ and $(t,t')_u$. Transitions $(t,t')_s$ and $(t,t')_u$ have for inputs and outputs the ones of t with places p_l substituted for p and the ones of t' with places p_r substituted for p. $(t,t')_s$ includes an additional loop with *stable* while $(t,t')_u$ transfers the token from *unstable* to *stable*.

The behaviour of $\mathcal{N}''$ is the following one: it mimics simultaneously twice the behaviour of $\mathcal{N}$ keeping track whether both current markings (in the left and the right simulation) are stable markings with the help of places *stable* and *unstable*.

Given a marking m, let us denote by m_l (resp. m_r) the submarking on P_l (resp. P_r). In order to check whether $\mathcal{N}$ is not marking deterministic, one checks in $\mathcal{N}''$ the reachability of a marking m with $m(\mathit{stable}) = 1$ and $m_l \neq m_r$. The set of such markings is a semi-linear set, so this problem is decidable (combining the results of citeReachaPbMayer and [**?**]).

When a net is deterministic one can transform it into another one with the same language but without silent transitions. Furthermore as the transformation depicted in the proof of the next proposition relates a transition of the new net to a visible transition of the original one the may/must information can be preserved.

Proposition 3. *Let $\mathcal{N}$ be a deterministic Petri net, then one can build another deterministic net $\mathcal{N}'$ without silent transitions such that $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{N}')$.*

Proof. We first compute the number of maximal silent transitions consecutively fireable in $\mathcal{N}$, say b. Then $\mathcal{N}''$ is built as follows.

- Its set of places is the one of $\mathcal{N}$.
- For every sequence of transitions $t_1, \ldots, t_i, t_{i+1}$ such that (1) $0 \leq i \leq b$, (2) $t_1, \ldots, t_i$ are silent transitions and t_{i+1} is a visible transition one defines $(t_1, \ldots, t_i, t_{i+1})$ a transition of $\mathcal{N}'$. Backward and forward matrices

$W^-[p,(t_1,\ldots,t_i,t_{i+1})]$ and $W^+[p,(t_1,\ldots,t_i,t_{i+1})]$ are inductively defined as usual (W is the incidence matrix):
$W^-[p,(t_1,\ldots,t_j,t_{j+1})] = \max(W^-[p,(t_1,\ldots,t_j)], W^-[p,t_{j+1}] - W[p,(t_1,\ldots,t_j)])$
$W[p,(t_1,\ldots,t_j,t_{j+1})] = W[p,(t_1,\ldots,t_j)] + W[p,t_{j+1}]$
$W^+[p,(t_1,\ldots,t_j,t_{j+1})] = W^-[p,(t_1,\ldots,t_j,t_{j+1})] + W[p,(t_1,\ldots,t_j,t_{j+1})]$

The set of reachable markings of $\mathcal{N}'$ is the set of stable reachable markings of $\mathcal{N}$ and a firing sequence in $\mathcal{N}$ leading to a stable marking can be decomposed into subsequences of silent transitions ended by a visible transition and so may be simulated by a single transition in $\mathcal{N}'$. The other direction is similar.

4.2 Specification refinement

We now address the problem of specification refinement when the initial specification is given by a deterministic Petri net while the possibly refining one is given by an arbitrary Petri net.

Proposition 4. *Let $\mathcal{M}$ be a modal Petri net specification with $\mathcal{N}$ its associated* deterministic *Petri net and $\mathcal{M}'$ be a modal Petri net specification with $\mathcal{N}'$ its associated Petri net. The problem of determining whether $\mathcal{M}'$ is a refinement of $\mathcal{M}$ is decidable.*

Proof. First we check whether $\mathcal{L}(\mathcal{N}') \subseteq \mathcal{L}(\mathcal{N})$. This is not the case if there is a word w generated by $\mathcal{N}'$ but not by $\mathcal{N}$. Due to proposition 3, w.l.og. we assume that $\mathcal{N}$ has no silent transition. We look for a minimal word w, i.e. $w = w'a$ with $a \in \Sigma$ and $w' \in \mathcal{L}(\mathcal{N}') \cap \mathcal{L}(\mathcal{N})$.

We build a net $\mathcal{N}''$ as follows.

- The set of places of $\mathcal{N}''$ is the disjoint union of the ones of $\mathcal{N}$ and $\mathcal{N}'$.
- Every silent transition of $\mathcal{N}'$ is inserted in $\mathcal{N}''$ with the same backward and forward incidences.
- For every pair of transition $t \in T$, $t' \in T'$ with $\lambda(t) = \lambda'(t')$ one defines a transition (t,t') of $\mathcal{N}''$ whose incidences are the "union" of incidences of t and t'.

Observe that $\mathcal{N}''$ generates exactly the words of $\mathcal{L}(\mathcal{N}') \cap \mathcal{L}(\mathcal{N})$. In this net we are looking for a marking (say m such that $m_0[\sigma\rangle m$) whose projection on P' enables a transition t' labelled by some $a \in \Sigma$ but whose projection on P disables every transition t labelled by a. The set of such markings is a semi-linear set so this problem is decidable. Moreover this procedure is sound since if we note w' the labelling of σ, every sequence in $\mathcal{N}$ that is labelled w reaches the same marking (determinism hypothesis).

Once this first cheking is performed, with $\mathcal{N}''$ we check the second condition for refinement. More precisely we are looking for a marking (say m such that $m_0[\sigma\rangle m$) whose projection on P enables a must transition t labelled by some $a \in \Sigma$ but whose projection on P' cannot enable any must transition t' labelled by a. The set of such markings is again a semi-linear set so this problem is decidable.

The part of the proof related to inclusion of languages is close to the results of [8]. The difference lies in our definition of determinism which allows different firing sequences for the same word (but leading to the same stable marking).

References

1. A. Antonik, M. Huth, K. G. Larsen, U. Nyman and A. Wasowski. Complexity of decision problems for mixed and modal specifications. In *Proc. of the 10th Int. Conf. on Found. of Software Science and Comp. Struct. (FoSSaCS'08)*, vol. 4962 of *LNCS*, Springer, 2008.
2. A. Antonik, M. Huth, K.G. Larsen, U. Nyman and A. Wasowski. EXPTIME-complete decision problems for mixed and modal specifications. In: *Proc. of EXPRESS*, July 2008.
3. K.G. Larsen. Modal specifications. In: *Joseph Sifakis, editor, Automatic Verification Methods for Finite State Systems*, volume 407 of LNCS, pages 232–246, 1989.
4. K.G. Larsen, U. Nyman and A. Wasowski. On modal refinement and consistency. In *Proc. of the 18th International Conference on Concurrency Theory, (CONCUR07)*, LNCS pages 105–119, Springer Verlag, 2007.
5. K.G. Larsen, U. Nyman and A. Wasowski. Modal I/O automata for interface and product line theories. In *Programming Languages and Systems, 16th European Symposium on Programming (ESOP07)*, vol. 4421 of *LNCS*, pages 64–79. Springer, 2007.
6. K.G. Larsen and b. Thomsen. A modal process logic. In:*Third Annual IEEE Symposium on Logic in Computer Science LICS*, pages 203–210, 1988.
7. E. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. of the 13th Annual ACM Symposium on Theory of Computing (STOC'81)* pages 238–246, 1981.
8. E. Pelz. Closure properties of deterministic Petri net languages. In *STACS'87*, vol. 247 of *LNCS*, Springer 1987.
9. J.L. Peterson, Petri net theory and the modeling of systems, Prentice-Hall, Englewood Cliffs, NJ, 1981
10. J.-B. Raclet. Residual for Component Specifications. In: *Proc. of the 4th International Workshop on Formal Aspects of Component Software (FACS07), Sophia-Antipolis, France*, September 2007.
11. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay and R. Passerone. Modal Interfaces: Unifying Interface Automata and Modal Specifications. In *Proc. of 9th International Conference on Embedded Software (EMSOFT'09), Grenoble, France, ACM*, October 2009.