A Compositional Semantics for UML State Machines Aimed at Performance Evaluation*

José Merseguer, Javier Campos
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Spain
{jmerse,jcampos}@posta.unizar.es

Simona Bernardi, Susanna Donatelli
Dipartimento di Informatica
Università di Torino, Italy
{bernardi,susi}@di.unito.it

Abstract

Unified Modeling Language (UML) is gaining acceptance to describe the behaviour of systems. It has attracted the attention of researchers that are interested in deriving, automatically, performance evaluation models from system's descriptions. A required step to automatically produce a performance model (as any executable model) is that the semantics of the description language is formally defined. Among the UML diagrams, we concentrate on States Machines (SMs) and we build a semantics for a significant subset of them in terms of Generalized Stochastic Petri Nets (GSPNs). The paper shows how to derive an executable GSPN model from a description of a system, expressed as a set of SMs. The semantics is compositional since the executable GSPN model is obtained by composing, using standard Petri net operators, the GSPN models of the single SMs, and each GSPN model is obtained by composition of submodels for SM basic features.

1 Introduction

Unified Modeling Language (UML) [13] is a semi formal language that defines twelve types of diagrams, divided into three categories: static, behavioural and diagrams to organize application modules. Behavioural diagrams are of five kinds: Use Case diagram, Sequence diagram (SD), Activity diagram, Collaboration diagram and Statechart diagram (SC). The SC *describes possible sequences of states and actions through which the modeled element can proceed during its lifetime as a result of reacting to discrete events* [13]. The SD specifies a set of partially ordered messages.

We consider discrete event systems whose behaviour is described by means of SC and SD and we aim at validating and evaluating the system behaviour by checking properties and computing performance on the SC and SD description. Since SC and SD are not meant for performance, we translate them into Generalized Stochastic Petri Nets (GSPNs) [1]. In this paper we concentrate on SC, while the case of SD is tackled in a companion paper [3] where it is also shown how the SD and SC descriptions can be used synergically in the performance analysis step.

An SC maps into a UML State Machine (SM), a behavioural package of UML, which *specifies a set of concepts that can be used for modeling discrete behaviour through finite state-transition systems* [13]: indeed in the UML definition SC is seen as a graph representation of a SM, and therefore the semantics of an SC is given in terms of SM.

We assume that a system is described by a set of SMs, and we show in this paper how a GSPN can be generated by composing the GSPN models of the single SMs. The GSPN models are defined compositionally starting from the GSPN models of each state together with its transitions. Also these models are defined in terms of smaller GSPNs that model the basic elements of a SM: entry and exit actions, do activities, internal and outgoing transitions, events, deferred events, etc. The translation is performed taking into account that the operational semantics of the PN system must guarantee the "run to completion assumption" of UML, that means that an event can only be dequeued and dispatched if the processing of the previous current event is fully completed.

The main contribution of this paper is therefore to define the translation of a number of elementary SM concepts into GSPN submodels and to define the formulas that allow to obtain a translation of a set of UML SMs into a GSPN model. So, we accomplish the future work proposed in [12] to get a formal translation instead the set of translation rules given in that work, then gaining the benefits of a formalization.

Our translation is meant for performance evaluation, and this justifies the choice of GSPNs as the target language of

*This work has been developed within the projects: P084/2001 of the Gobierno de Aragón, UZ00-TEC-03 of the Universidad de Zaragoza and IST 25434 DepAuDE.

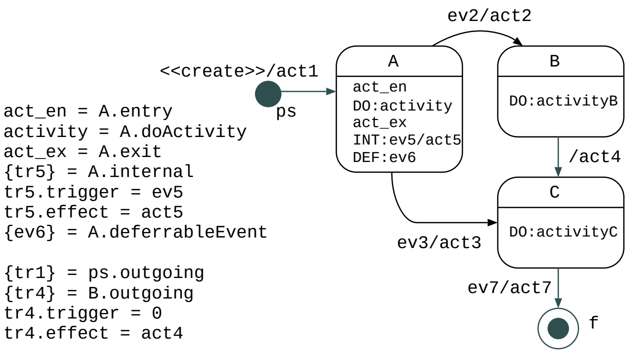


Figure 1. A UML State Machine.

the translation, since there is a large number of performance evaluation tools available for GSPNs. GSPNs being a language with a formal semantics, we also get the equally important contribution of providing a formal semantics to a subset of SMs.

From the richness of the SM package we have chosen a subset of elements, in particular those that relate to "flat" SMs, disregarding other important constructs such as SynchStates, StubStates, CompositeStates, SubmachineStates, Pseudostates (except initial states) that are mainly meant for a hierarchical definition of the SMs, and to introduce concurrency inside a single SM, instead of between different SMs.

UML SMs are defined in [13] as an object-based variant of Harel statecharts [5], with major differences identified in [13]. Several semantics for Harel statecharts have been proposed in the literature [14, 6, 18, 11], but none of them in the context of performance evaluation. Other works have been devoted to give formal semantics to UML SMs as [9, 8], but with model checking of properties as main goal.

A good survey of the different approaches for performance evaluation based on UML diagrams can be found in [2]: among them [7] is the one that shares most similarity with this paper. In [7], GSPN models are produced starting from UML diagrams: the main difference with our work is that the construction in [7] is described only at an intuitive level, through an example, and no systematic approach to the translation is given.

In this paper we adopt the notation defined in [1] for GSPN, but simplified to consider only ordinary systems. A labeled ordinary GSPN (LGSPN) is a triplet $\mathcal{LS} = (S, \psi, \lambda)$, where S is a GSPN ordinary system, $\lambda : T \longrightarrow L^T \cup \tau$ is the labeling function that assigns to a transition a label belonging to the set $L^T \cup \tau$ and $\psi : P \longrightarrow L^P \cup \tau$ is the labeling function that assigns to a place a label belonging to the set $L^P \cup \tau$. τ-labeled net objects are considered to be internal.

The rest of the article is organized as follows: Section 2 describes through an example, the UML informal semantics of the subset of elements that we consider and establishes the three steps of the translation. Section 3 discusses the first step, the translation of a state and its associated transitions. Section 4 illustrates the second step, how the GSPN models of the states are composed to produce a GSPN model of a SM. Section 5 accomplishes the third step, how a model of a system is obtained by composing the GSPNs of the component SMs. Section 6 summarizes the paper and discusses future extensions.

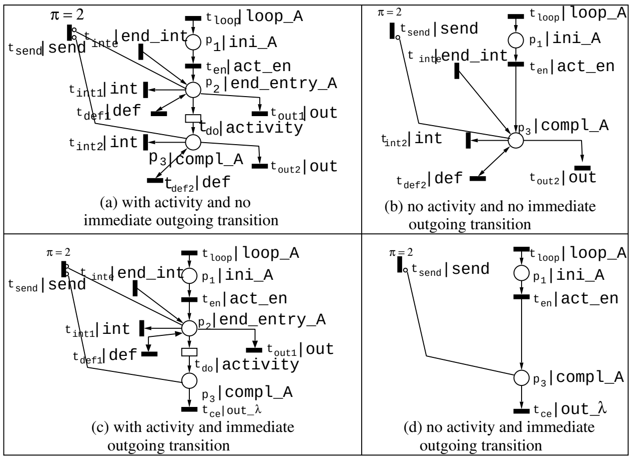
2 Translating UML State Machines into GSPNs

A SM $sm \in StateMachine$ is basically characterized by states and transitions. In the following we describe the informal semantics given by the UML SMs package, considering only those elements that are used for the definition of "flat" SM. Figure 1 shows a flat SM that is used as a running example in the paper.

The SM sm in Figure 1 is composed of three simple states $A, B, C \in SimpleState$, an initial pseudostate, represented by a black dot, $ps \in Pseudostate$, a final state, represented by a bull eye $f \in FinalState$, and five outgoing transitions. The SM sm starts its execution by firing transition $tr1 \in ps.outgoing$ which means the arrival of the stereotyped event $\ll create \gg$, $create = tr1.trigger \in Event$; as a consequence, action $act1 = tr1.effect \in Action$ is performed. When in state A, the entry action $act\_en = A.entry \in Action$ is executed first; after, the execution of the activity $activity = A.doActivity \in Action$ begins. Either outgoing transitions of state A or the internal transitions as $tr5 \in A.internal$ or deferrent events such as $ev6 \in A.deferrableEvents$ may occur during the activity execution. Outgoing transitions provoke an exit from the state and an entry into the target state (possibly the source state itself – self-loop outgoing transitions). Internal transitions instead do not provoke a change of state and no entry or exit action is executed. Deferred events are not triggered in the present state, they are retained by the SM. Completion of activity means the generation of the "completion event" for state A. When an outgoing transition as $tr4$ from B to C has no trigger it means that it fires when the "completion event" for its source state $tr4.source = B$ is generated (they are called immediate outgoing transitions). After visiting states B and C or just C depending on the triggered events, sm completes when arrives to its final state f.

Given a SM, the approach taken for its translation into a LGSPN model consists of the following steps:

step 1 Each $s \in SimpleState$ is modelled by a LGSPN representing the basic elements of states and transitions. Section 3 discusses this step.



(a) with activity and no immediate outgoing transition
(b) no activity and no immediate outgoing transition
(c) with activity and immediate outgoing transition
(d) no activity and immediate outgoing transition

Figure 2. The LGSPN basic systems.

step 2 The initial pseudostate and the final states are translated into LGSPN subsystems to be composed with those of the previous step to produce a LGSPN model of the entire SM. Section 4 discusses this step.

step 3 If the system is described through a set of SMs then compose the LGSPN subsystems of the SMs and define the initial marking. Section 5 discusses this step.

3 Translation of simple states

A state $s \in State$ models a situation during which some invariant condition holds. The invariant may represent a static situation such as an object waiting for some external event to occur. However, it can also model dynamic conditions such as the process of performing some activity. As we see in Figure 1 two special cases of states are considered in a "flat" SM: simple states, those that do not have substates (*A*, *B* and *C*), and final states. Some of the associations involving the class $State$ are heavily related with the class $Action$. Actions are specifications of executable statements and can be realized by: sending a message to an object (actions belonging to the subclass $CallAction$ or $SendAction$) or modifying a link or a value (the rest); they are the following:

- $entry : State \longrightarrow Action$, associates to s an optional $Action$ that is executed whenever s is entered regardless of the transition taken to reach s.
- $exit : State \longrightarrow Action$, associates to s an optional $Action$ that is executed whenever s is exited regardless of which transition was taken out of s.
- $doActivity$[1] $: State \longrightarrow Action$, associates to s an optional $Action$ that is executed while being in s.

[1] In the following $doAct$.

Moreover, in order to understand the semantics of a state, the following associations must be mentioned:

- $internal : State \longrightarrow Transition$, associates to s a set of transitions that, if triggered, occur without causing a state change.
- $outgoing : StateVertex \longrightarrow Transition$, given a state $s \in State$ playing a $StateVertex$ role, associates a set of transitions that, if triggered, occur causing a state change.
- $deferrableEvent : State \longrightarrow Event$, associates to s a set of events that can be retained in it.

3.1 Entry actions and activities

In the following, we give a representation into LGSPN formalism for the entry actions and the activities in a simple state, such as state A of sm depicted in Figure 1. The resulting LGSPN will include also "interface" transitions to compose itself with the rest of the nets that interpret A, and interface transitions that will be useful for the composition of A with the rest of the states in sm; for this reason we call this net "basic". The last kind of information contained in the net refers to immediate outgoing transitions, that is transitions without triggering events. An immediate outgoing transition tr, such as the one connecting state B to state C in Figure 1, is characterized by the following restriction: $tr \in A.outgoing$ and $tr.trigger = \emptyset$. Note that if a state has got several immediate outgoing transitions then the choice of which transition to execute is non deterministic, otherwise guards should be introduced to solve the conflict.

Depending on whether an activity exists or not and whether immediate outgoing transitions exist or not in the state, a different translation is required for the corresponding state. Figure 2 shows the four possible translations of a state into LGSPN: net objects are denoted as $name|label$ where $name$ is the name of the object and $label$ is the label.

Let us consider the more complex case with activity and without immediate outgoing transitions of Figure 2(a): transition t_{loop} puts a token in place p_1 meaning the entry in state A due to outgoing self-loop transitions; t_{loop} may exist or not depending whether there exist self-loop transitions. Firing of transition t_{en} represents the execution of entry action $A.entry$ that, being an action, can belong to any of the subclasses of class $Action$. In particular, if an action belongs to either the subclass $SendAction$ or to the subclass $CallAction$ it will generate the corresponding events. The interpretation of these particular cases will be given when transitions are translated; now, for simplicity, we assume that actions do not belong to any of these subclasses. A token in place p_2 represents action termination as well as the beginning of the activity $A.doAct$ modeled by the timed transition t_{do}. The computation of its rate depends on stereotyped annotation in the SM largely described in [12] and surveyed in [2]. The "completion event" for the state is generated when the activity completes and it is represented by a token in place p_3.

The rest of the elements in the LGSPN are interface transitions which represent deferred events, outgoing and internal transitions. Deferred events are accepted either when the activity is in execution or after the "completion event" has been generated; the reception of those events is modeled by transitions t_{def1} and t_{def2}, respectively. Deferred events are sent back to the SM event queue just before the exit action is executed: the sending is therefore represented by transition t_{send} characterized by an higher priority ($\pi = 2$) and by inhibitor places p_2, p_3. Internal and outgoing transitions, as well as deferred events, are accepted either when the activity is in execution (modeled by transitions t_{int1} and t_{out1}, respectively) or after the "completion event" has been generated (modeled by transitions t_{int2} and t_{out2}, respectively). The termination of internal transitions does not cause neither an entry to the current state nor an exit to the current state and it is modeled by t_{inte}.

The LGSPN of Figure 2(c) models the case of a state with both the activity and immediate outgoing transition(s); it differs from the LGSPN of Figure 2(a) since transitions t_{def2}, t_{out2} and t_{int2} do not appear, while transition t_{ce} is added to represent the triggering of the "completion event".

In order to give the definition of the LGSPN basic system for a state $A \in SimpleState$ of the SM that encompasses the four cases of Figure 2 we need the following emptiness indicator functions:

$$\chi(X) = \begin{cases} 0 & \text{if } X = \emptyset \\ 1 & \text{if } X \neq \emptyset \end{cases} \text{ and } \overline{\chi}(X) = 1 - \chi(X),$$

where X is a subset of class instances. Moreover, by abuse of notation, we assume that $0 \cdot X = \emptyset$ and $1 \cdot X = X$.
A basic system for state A of the SM sm is a LGSPN $\mathcal{LS}^b_A = (S^b_A, \psi^b_A, \lambda^b_A)$ characterized by the following set of transitions:

$$\begin{aligned} T^b_A \;\; = \;\; & \{t_{en}\} \cup \chi(A.doAct)\{t_{do}\} \cup \\ & \chi(A.internal)\big(\chi(A.doAct)\{t_{int1}, t_{inte}\} \cup \\ & \overline{\chi}(OUT_\lambda)\{t_{int2}, t_{inte}\}\big) \cup \chi(A.outgoing) \\ & \big(\chi(A.doAct)\{t_{out1}\} \cup \overline{\chi}(OUT_\lambda)\{t_{out2}\}\big) \cup \\ & \chi(A.deferrableEvent)\big(\{t_{send}\} \cup \\ & \chi(A.doAct)\{t_{def1}\} \cup \overline{\chi}(OUT_\lambda)\{t_{def2}\}\big) \cup \\ & \chi(OUT_{loop})\{t_{loop}\} \cup \chi(OUT_\lambda)\{t_{ce}\}, \end{aligned}$$

where $OUT_{loop}=\{tr \in A.outgoing, tr.source = tr.target\}$ and $OUT_\lambda = \{tr \in A.outgoing, tr.trigger = \emptyset\}$.

The set of places is $P^b_A = \{p_1, p_3\} \cup \chi(A.doAct)\{p_2\}$.
The input, output and inhibitor functions are respectively:

$$I^b_A(t) = \begin{cases} \{p_1\} & \text{if } t = t_{en} \\ \{p_2\} & \text{if } t \in \{t_{do}, t_{def1}, t_{int1}, t_{out1}\} \\ \{p_3\} & \text{if } t \in \{t_{def2}, t_{int2}, t_{out2}, t_{ce}\} \\ \emptyset & \text{otherwise} \end{cases}$$

$$O^b_A(t) = \begin{cases} \{p_1\} & \text{if } t = t_{loop} \\ \{p_2\} & \text{if } t \in \{t_{def1}\} \\ & \cup\ \chi(A.doAct)\{t_{en}, t_{inte}\} \\ \{p_3\} & \text{if } t \in \{t_{do}, t_{def2}\} \\ & \cup\ \overline{\chi}(A.doAct)\{t_{en}, t_{inte}\} \\ \emptyset & \text{otherwise} \end{cases}$$

$$H^b_A(t) = \begin{cases} \chi(A.doAct)\{p_2\} \cup \{p_3\} & \text{if } t = t_{send} \\ \emptyset & \text{otherwise} \end{cases}$$

The priority and the weight functions are respectively:

$$\Pi^b_A(t) = \begin{cases} 0 & \text{if } t = t_{do} \\ 2 & \text{if } t = t_{send} \\ 1 & \text{otherwise} \end{cases}, \quad W^b_A(t) = \begin{cases} r_{do} & \text{if } t = t_{do} \\ 1 & \text{otherwise} \end{cases}$$

where r_{do} is the rate parameter of the timed transition t_{do}.
The labeling functions for places and transitions are:

$$\psi^b_A(p) = \begin{cases} ini\_A & \text{if } p = p_1 \\ end\_entry\_A & \text{if } p = p_2 \\ compl\_A & \text{if } p = p_3 \end{cases}$$

$$\lambda^b_A(t) = \begin{cases} loop\_A & \text{if } t = t_{loop} \\ \lambda & \text{if } t \in \overline{\chi}(A.entry)\{t_{en}\} \\ act\_en & \text{if } t \in \chi(A.entry)\{t_{en}\} \wedge \\ & \quad act\_en = A.entry.name \\ activity & \text{if } t = t_{do} \wedge \\ & \quad activity = A.doAct.name \\ send & \text{if } t = t_{send} \\ def & \text{if } t \in \{t_{def1}, t_{def2}\} \\ end\_int & \text{if } t = t_{inte} \\ int & \text{if } t = \{t_{int1}, t_{int2}\} \\ out & \text{if } t = \{t_{out1}, t_{out2}\} \\ out\_\lambda & \text{if } t = t_{ce} \end{cases}$$

where, by abuse of notation, $A = A.name$.
Finally, the initial marking function is defined as $\forall p \in P^b_A : M^{b0}_A(p) = 0$.

3.2 Deferred events

An event $e \in Event$ is a specification of a type of observable occurrence. The occurrence that generates an event instance is assumed to take place at an instant in time with no duration.

A state may specify a set of event types that are candidates to be retained by the SM if they trigger no transition in that state. Let us give a representation into the

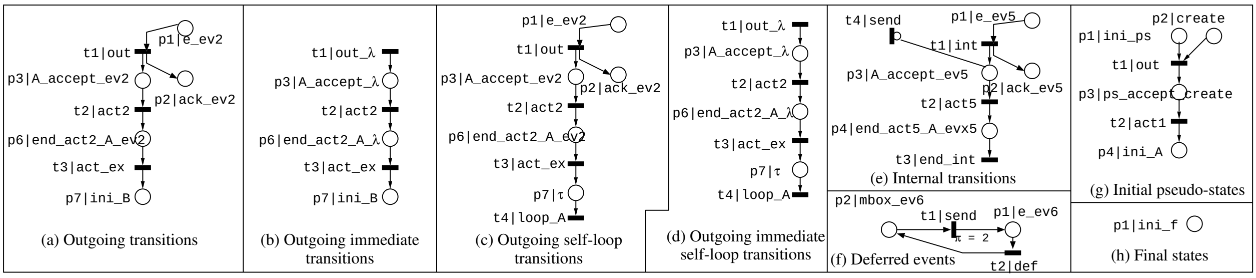


Figure 3. Translation of the elements in a simple state, the initial state and the final state into LGSPNs.

LGSPN formalism of a deferred event, such as $ev6 \in A.deferrableEvents$ of the SM depicted in Figure 1. Figure 3(f) shows the translation of the deferred event into a LGSPN. Note that the size of the set $A.deferrableEvents$ in the example is one; if it were larger then there would appear a net as the one in Figure 3(f) for each event in the set.

A token in place p_1 labelled e_ev6 represents an instance of an event of type $ev6$ in sm, it will not trigger any transitions in state A, therefore it will be queued by transition t_2 into place p_2 labelled $mbox\_ev6$ until a state is reached where either the event is no longer deferred or where the event triggers a transition. The LGSPN in isolation does not show the decisions taken to guarantee the behaviour described for the deferred events, this behaviour will be understood when the LGSPN is composed with the rest of the systems for A when inhibitor arcs are added.

A formal definition of the LGSPN can be derived similarly to what was previously done for basic states.

3.3 Transitions

A transition $tr \in sm.transition$ represents a complete response of sm to a particular event instance $ev = tr.trigger \in Event$. Firing of a transition can provoke an action $act = tr.effect \in Action$ to be performed.

Given a simple state A there may exist two kinds of transitions: internal transitions, belonging to the set $A.internal$, and outgoing transitions, belonging to the set $A.outgoing$.

Depending on whether a transition is internal, such as transition $tr5$ of Figure 1, or outgoing, such as transition $tr2$ of the same Figure, a different translation into LGSPN is required. Figure 3(e) depicts the translation into LGSPNs of the internal transition $tr5$, Figure 3(a) depicts the translation into LGSPNs of the outgoing transition $tr2$. The additional pictures (b), (c), and (d) are modifications of transition $tr2$ to show the case of outgoing immediate transitions, outgoing self-loop and outgoing immediate self-loop, respectively.

Let us consider first the case of the internal transitions. A token in place p_1 with label e_ev5 represents an instance of an event of type $ev5$. Places labeled with event names represent event queues (observe that no policy is associated to the place, apart for the choice of the term "queue"). Place p_1 triggers the transition t_1 meaning that the event has been accepted and therefore tokens in places p_2 and p_3 are added. A token in place p_2 with label ack_ev5 means the acknowledge of the arrival of the event $ev5$ to the action which generates the event, but actually there is no way to determine whether the event has been generated by a synchronous or asynchronous action, the only possibility is that each transition that consumes an event evx puts a token into the place of label ack_evx. Place p_3 inhibits transition t_4, that when composed with the system for the deferred events means that they cannot be dequeued until the internal transition has been accepted. Transition t_2 is labelled with the effect $act5$ of the transition $tr5$.

Let us consider the case of the outgoing transition $tr2$ modelled in Figure 3(a). The translation given for the previous case is valid with slight changes: the exit action of A is represented by transition t_3 labeled act_ex (if there is no exit action then transition t_3 is labeled λ), transition labeled end_int is replaced by other transition labeled act_ex (that represents $A.exit$), finally the interface place ini_B has been added, it represents the entrance in the state B. The other cases of outgoing transitions are just minor variations of this one.

A LGSPN system $\mathcal{LS}^o_A = (S^o_A, \psi^o_A, \lambda^o_A)$ for outgoing transition - illustrated in Figure 4(a,b,c,d) - is characterized by the following sets of transitions and places, respectively:

$$T^o_A = \{t_1, t_2, t_3\} \cup \chi(OUT_{loop})\{t_4\},$$

$$P^o_A = \chi(tr.trigger)\{p_1, p_2\} \cup \{p_3, p_6, p_7\}$$

where, $\chi()$ is the emptiness indicator function and OUT_{loop} is the set of the outgoing self-loop transitions of state A as defined in the previous subsection 3.1. For brevity, we omit the other net elements that can be easily derived as in subsection 3.1.

Depending on whether an action act belongs to $SendAction \cup CallAction$ or not, one or more events may

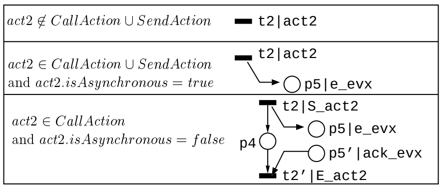


Figure 4. Different types of actions.

or may not be generated. For simplicity, we assume that at most an event may be generated: it is trivial to extend the translation in case of more events. Moreover, act is characterized by the attribute *isAsynchronous* that allows to specify if the dispatched stimulus is asynchronous or not, where as *synchronous* means that the action will not be completed until the event eventually generated by the action is consumed by the receiver. Figure 4 shows the different ways of translating an action, observe in particular the case of synchronous call actions that require an acknowledge, and therefore for each event place an acknowledge place has been added.

3.4 The model of a simple state

The LGSPNs obtained in the previous sections can be composed using the associative operator of superposition of places and transitions based on non injective labeling functions defined in [3],

$$\underset{L_T, L_P}{||} : LGSPN \times LGSPN \rightarrow LGSPN$$

where L_T, L_P are the sets of transition and place labels, respectively. The resulting LGSPN $\mathcal{LS}_s = (S_s, \psi_s, \lambda_s)$ interprets a simple state s together with its outgoing transitions. Obviously the interest of this net is not to perform any kind of analysis but to establish the fine grain unit to compose SMs (together with the LGSPNs for the initial pseudostate and the final states).

Let Ev be the set of events produced/consumed by s and Lev^P the set of labels of event and event acknowledge places, $Lev^P = \{e\_evx, \forall evx \in Ev\} \bigcup \{ack\_evx, \forall evx \in Ev\}$. Let $States$ be the set of the states of the SM, and $Lstate^P$ the set of labels of places representing the entrance into states, $Lstate^P = \{ini\_target, \forall target \in States\}$.

According to the translations defined up to now, given a state s with h internal transitions, k deferred events, l outgoing self-loop transitions and l' outgoing transitions we get $h + k + l + l' + 1$ LGSPN models that need to be combined to get a model of the state s. Figure 5 shows an interface view of the five kinds of models that are composed: (a) is the basic net model BN, (b) is the internal transition model INT_i, (c) is the deferred event model DEF_j, (d) is the outgoing transition model OUT_p, (e) is the outgoing self-loop transition model OUT_S_r.

The composition proceeds in two steps: first we compose submodels of the same "type" and then the resulting models are composed together,

$$INT = \underset{Lev^P}{\overset{i=1,..,h}{||}} INT_i, \;\; DEF = \underset{Lev^P}{\overset{j=1,..,k}{||}} DEF_j,$$

$$OUT = \underset{Lev^P \cup Lstate^P}{\overset{p=1,..,l'}{||}} OUT_p, \; OUT\_S = \underset{Lev^P}{\overset{r=1,..,l}{||}} OUT\_S_r$$

where $||^{i=\cdots}$ is the n-ary extension of $||$.
Finally, $\mathcal{LS}_s = (S_s, \psi_s, \lambda_s)$ is the LGSPN model representing state s and it is defined by:

$$\mathcal{LS}_s = (((INT \underset{Lev^P}{||} DEF) \underset{Lev^P}{||} OUT\_S) \underset{Lev^P}{||} OUT) \underset{Lev^P, Ltr^T}{||} BN$$

where BN is the LGSPN basic system for state s and $Ltr^T = \{int, end\_int, def, send, out, loop\_s\}$.

Figure 5(f) shows an interface view of $\mathcal{LS}_s$, while the net $\mathcal{LS}_s$ for state s equal to A is shown in Figure 6.

4 Translation of a State Machine

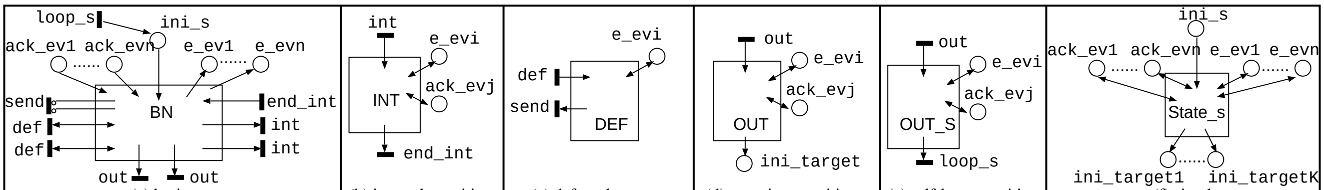
In this section, we address the second step of our proposal by defining LGSPNs for the initial and final states and composing them with the LGSPNs for the simple states. The resulting net is the interpretation of the whole SM.

4.1 Initial and final states

An initial pseudostate depicted by a black dot means the start point of the SM. At most one initial pseudostate can appear in a "flat" SM , let us name it ps, such as $ps \in Pseudostate$ of the SM depicted in Figure 1. Moreover, the pseudostate can have at most one outgoing transition, $tr \in ps.outgoing$, without trigger or with a trigger stereotyped $\ll create \gg$. It is possible to associate an effect $act \in tr.effect$ to the outgoing transition. No incoming transitions are allowed for initial pseudostates.

Figure 3(g) shows the translation into a LGSPN of ps and its outgoing transition. It results into a LGSPN $\mathcal{LS}_{ps} = (S_{ps}, \psi_{ps}, \lambda_{ps})$.In case the effect of the transition is left out then transition t_2 and place p_3 are removed, and in case of outgoing transition is not stereotyped place p_2 is removed.

A token in place p_1 represents a resource waiting for an instance event of type "create" (a token in p_2) to fire transition t_1 that starts the SM. After completion of the action (firing of transition t_2), a token in place p_4 means the completion of the initial pseudostate therefore the entry into the state A.



(a) basic net (b) internal transitions (c) deferred events (d) outgoing transitions (e) self-loop transitions (f) simple state

Figure 5. (a..e) LGSPN components for a state. (f) LGSPN representing a state.

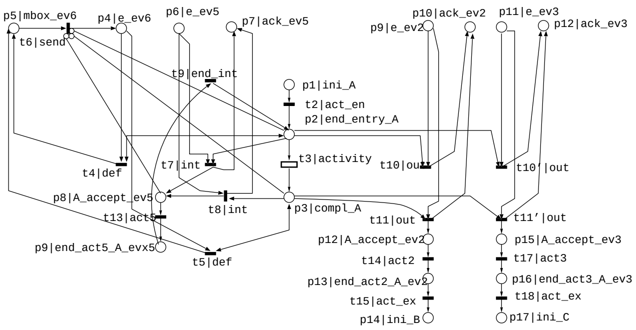


Figure 6. LGSPN of the state A in Figure 1.

A final state is a special kind of state that can appear in a flat SM meaning that the entire SM has completed; a final state, like $f \in FinalState$ for the SM of Figure 1, is depicted by a bull eye. Figure 3(h) shows the translation into a LGSPN of the final state f, i.e., a single place p_1.

4.2 The model of a State Machine

The LGSPN that interprets the whole SM is obtained as follows. Let $States$ be the set of states of the SM sm (including initial and final ones), $Lstate^P$ the set of labels of the initial places of the LGSPN models of the states. Let Ev be the set of events produced/consumed by sm, Lev^P the set of labels of event and event acknowledge places. The LGSPN model of sm is obtained by composing the models $\mathcal{LS}_s$ as follows:

$$\mathcal{LS}_{sm} = \underset{Lev^P \cup Lstate^P}{\overset{s \in States}{||}} \mathcal{LS}_s$$

Figure 7 shows the LGSPN $\mathcal{LS}_{sm}$ that represents the interpretation of the SM in Figure 1.

5 System translation

This section explains how to create an analysable model for the system assuming it is described as a set of SMs. By analysable model we mean a GSPN model that includes the behaviour of the SMs that describe it on which we can compute logical properties and performance results: we have therefore to define how the LGSPN components are composed, what is the initial marking and the performance indices.

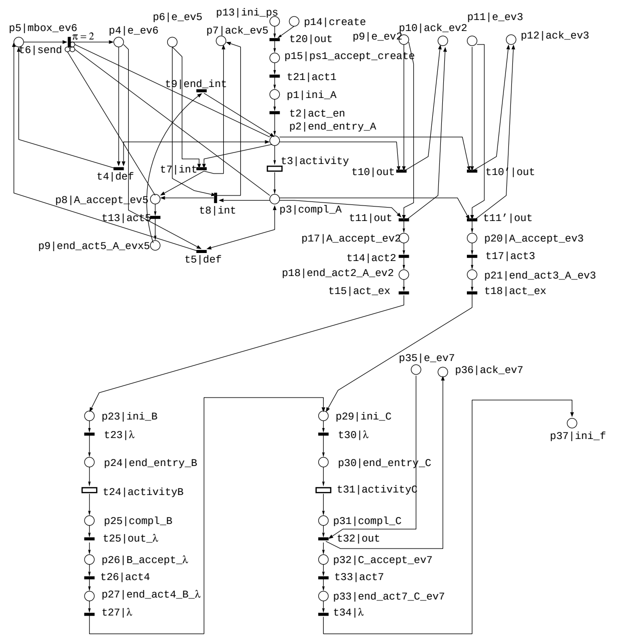


Figure 7. LGSPN of the SM in Figure 1.

We assume that the system is described by K SMs $\{sm_1, \ldots, sm_k\}$ which interact by exchanging synchronous and asynchronous messages through actions of the type $CallAction$ and $SendAction$. Let $\{\mathcal{LS}_{sm1}, ..., \mathcal{LS}_{smK}\}$ be the LGSPN models of the K SMs produced according to the translation given in step two.

A complete SM model for the system is obtained by superposition over event and event acknowledge places of the K SMs. Let Ev_j be the set of events produced/consumed by sm_j and Lev_j^P the set of labels of event and event acknowledge places, $Lev_j^P = \{evx, \forall evx \in Ev_j\} \bigcup \{ack\_evx, \forall evx \in Ev_j\}$, and $Lev^P = \bigcup_{j \in \{1,\ldots,K\}} Lev_j^P$, then the complete model of the K SMs is given by the LGSPN

$$\mathcal{LS}' = \underset{Lev^P}{\overset{j=1,..,k}{||}} \mathcal{LS}_{smj}$$

$\mathcal{LS}'$ can contain acknowledge places that are sinks (indeed all transitions that represent the consumption of an event send an acknowledge back since it is not defined if the event is synchronous or asynchronous), but if the event is generated by an asynchronous action no acknowledge is ever consumed and therefore the corresponding places should be removed. Let P_{ack} be the set of sink places with label of type ack_evx, then the model is, $\mathcal{LS} = \mathcal{LS}' \setminus P_{ack}$, where $A \setminus B$ removes from net A all places in B and their incidence arcs.

The initial marking for the LGSPN will be defined in two steps:

- The places that represent the initial pseudostates of the K SMs will be marked. For those SMs without initial pseudostate, one should be chosen for this purposes. These tokens represent the population (resources) of the system.

- The event or the set of events that launch the system is selected: the place/s that represent queues for these events are marked so that each token represents an event instance.

From the GSPN, system performance indices should be defined. The usual way to do that is to define proper reward functions [1] to compute throughput of transitions (representing speed of the system), average marking of places (representing utilization of a resource or average occupation of a buffer), or any other measure of interest for the particular system.

6 Conclusions

In this paper we have presented a translation of UML SMs into GSPNs. This allows the qualitative and quantitative analysis of systems that are described using UML SMs (or their graph-based representation called SC) by means of GSPN tools. The translation, being automatic, may contain unnecessary elements that contribute to increase the "state space explosion poblem", then additional work to define equivalent models [15] of reduced size is necessary.

The translation, being compositional, should be easy to implement, since only the basic elements need to be translated, while tools already exist that implement the composition operators used in the paper, for example the program algebra associated with the GreatSPN tool [4]. We have implemented a prototype [10] for the UML Activity diagrams, that also are based on the semantics of the SMs.

There are a number of features of SM that we have not considered: among them compound states and concurrent regions. The extension may not be trivial: as pointed out in [16] some of the hierarchical constructs of SM may lead to ambiguity and destroy compositionality. The work in [16] also indicates the restrictions that should be posed to avoid the problems and we intend to adhere to them. Another feature that we have not considered are guards: in SM guards may be associated to transitions to condition their execution. A guard is a boolean expression, usually over the values of attributes of the object modelled by the SM, if indeed the SM models the behaviour of an object. Since we have not considered objects and attributes we have omitted guards as well. This implies that our analysis is of the so called "static" type, as discussed in [17].

References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley, 1995.

[2] S. Balsamo and M. Simeoni. On transforming UML models into performance models. In *Transfs. in UML, ETAPS'01*.

[3] S. Bernardi, S. Donatelli, and J. Merseguer. From UML sequence diagrams and statecharts to analysable Petri net models. In *WOSP'02*. ACM. To appear.

[4] The GreatSPN tool. http://www.di.unito.it/~greatspn.

[5] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[6] D. Harel and A. Naamad. The STATEMATE semantics of the statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[7] P. King and R. Pooley. Using UML to derive stochastic Petri nets models. In *UKPEW'99*, pages 45–56.

[8] D. Latella, I. Majzik, and M. Massink. Towards a formal operational semantics of UML statechart diagrams. In *FMOODS'99*. Kluwer.

[9] J. Lilius and I. Paltor. The semantics of UML state machines. *Tech. rep. no.273 - Turku Centre for CS, Finland*, 1999.

[10] J. López-Grao, J. Merseguer, and J. Campos. Performance engineering based on UML and SPN: A software performance tool. In *ISCIS XVII*. To appear.

[11] A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of statecharts. LNCS 1119, pages 687–702. Springer.

[12] J. Merseguer, J. Campos, and E. Mena. Performance evaluation for the design of agent-based systems: A Petri net approach. In *SEPN, within ICATPN'00*, pages 1–20.

[13] OMG. Unified Modeling Language v1.4, Sept. 2001. http:/www.omg.org.

[14] A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. LNCS 526, pages 44–264. Springer.

[15] L. Pomello, G. Rozenberg, and C. Simone. A survey of equivalence notions for net based systems. LNCS 609, pages 410–472. Springer.

[16] A. Simons. On the compositional properties of UML statechart diagrams. In *ROOM'00*.

[17] R. Taylor. A general purpose algorithm for analyzing concurrent programs. *Communication of ACM*, (26), May 1983.

[18] A. Uselton and S. Smolka. A compositional semantics for statecharts using labeled transition systems. LNCS 836, pages 2–17. Springer.