Discrete Event Dynamic Systems manuscript No.
(will be inserted by the editor)

Dependability Analysis of DES based on MARTE and UML State Machines models

José Merseguer · Simona Bernardi

Received: date / Accepted: date

Abstract UML (Unified Modeling Language) is a standard design notation which offers the state machines diagram to specify reactive software systems. The "Modeling and Analysis of Real-Time and Embedded systems" profile (MARTE) enables UML with capabilities for performance analysis. MARTE has been specialized in a "Dependability Analysis and Modeling" profile (DAM), then providing UML with dependability assets. In this work, we propose an approach for the automatic transformation of UML-DAM models into Deterministic and Stochastic Petri nets and the subsequent dependability analysis.

Keywords Dependability modeling and analysis · MARTE · UML State Machines · Deterministic and Stochastic Petri Nets

1 Introduction

The dependability of a software system is the ability to deliver trusted service and to avoid failures that are more frequent and severe than acceptable [Avizienis et al (2004)]. Dependability encompasses availability, reliability, safety, integrity and maintainability; when the system misses one or more of them, it

This work has been supported by the European Community's Seventh Framework Programme under project DISC (Grant Agreement n. INFSO-ICT-224498), by CICYT DPI2010-20413 and by Fundación Aragón I+D.

J. Merseguer (✉)
Group of Discrete Events Systems Engineering
Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain
E-mail: jmerse@unizar.es phone: (+34) 976762336 fax:(+34)976761914

S. Bernardi
Group of Discrete Events Systems Engineering
Centro Universitario de la Defensa, Zaragoza, Spain
E-mail: simonab@unizar.es

can cause and suffer consequences that may discredit its own dependability. Hence, dependability modeling and analysis is a must for software systems. The kind of dependability analysis we address in this work is known as *fault forecasting* [Avizienis et al (2004)], which means to estimate the present number, the future incidence, and the likely consequences of faults.

The Unified Modeling Language [UML2 (2010)] is a widely recognized standard for the design of software systems. UML defines state machines (SMs) as a set of concepts that can be used for modeling discrete behavior and reactive systems through finite state-transition models. However, UML lacks support for dependability modeling. Hence we rely on the "Dependability Analysis and Modeling" (DAM) profile developed by Bernardi et al (2009). DAM was constructed on top of the standard "Modeling and Analysis of real-time Embedded systems" profile MARTE (2008), which extends UML to support schedulability and performance analysis. Nevertheless, a UML-DAM dependability specification lacks semantics to be formally analyzed.

The goal of the paper is to enable software engineers to formally analyze their UML-DAM SMs designs for dependability. To this end, we propose an approach for the automatic transformation of the UML-DAM SMs models into Deterministic and Stochastic Petri nets (DSPNs) [Ajmone Marsan and Chiola (1987)], which indeed can be used for dependability analysis. DSPNs are well suited for the modeling of systems in which events occur either after constant or stochastic durations, such as real-time systems with deterministic timeouts subjects to hw/sw failures. Due to page length restrictions, only basic features of SM are herein considered; more advanced ones (e.g., composite, join, fork and history states, deferred events, transition guards) were addressed in [Merseguer (2003)].

The standard UML describes the structure and behavior of its diagrams informally. The structure is given in terms of meta-models and constraints, while the behavior is described by English text. So, firstly we formalize the structure of the UML-SMs inspired by work of Lilius and Paltor (1999). Next, we translate this structure into DSPNs by means of a set of functions, which provide our formal interpretation of the UML-SMs. At this regard, we want to recall, from Bondavalli et al (2001), that the input of this transformation does not have formal semantics and, also, the UML-SM specification might be incomplete or ambiguous, so formal correctness of this transformation cannot be provided.

The balance of the paper is as follows. Section 2 states the formal definitions for UML-SMs and DSPNs. Section 3 adapts the translation of UML-SM basic features into Generalized Stochastic Petri Nets [Merseguer (2003)] to get a DSPN. Section 4 presents a formal translation of MARTE-DAM profiled SMs into DSPN. Section 5 describes the mapping of the dependability metrics, specified with DAM, onto DSPN output parameters. Related work is discussed in Section 6 and Section 7 concludes the paper and offers research directions. Appendix A includes the definition of LDSPN composition operators used in Sections 3 and 4.

2 Basic definitions

Definition 1 A Deterministic and Stochastic Petri net (DSPN) [Ajmone Marsan and Chiola (1987)], is a tuple $\mathcal{N} = (P, T, I, O, H, M^0, \Phi, \Lambda)$ where:

- P is the set of places,
- $T = T_I \cup T_D \cup T_E$ is the set of transitions, divided into *immediate* (T_I), *deterministic* (T_D) and *exponential* (T_E) transitions,
- $I, O, H : P \times T \rightarrow \mathbb{N}$ are, respectively, the input, output and inhibitor arc multiplicity functions,
- $M^0 : P \rightarrow \mathbb{N}$ assigns the initial number of tokens in each place,
- $\Phi : T \rightarrow \mathbb{N}$ assigns a priority to each transitions: timed transitions (deterministic and exponential) have zero priority, while immediate transitions have priority greater than zero,
- $\Lambda : T \rightarrow \mathbb{R}$ assigns to each immediate transition a weight, and to each timed transition a firing time delay. The firing time delay is a constant for a deterministic transition, while for an exponential one represents the mean value of the negative exponential distribution.

We will consider labeled DSPNs [Donatelli and Franceschinis (1996)], that is DSPNs provided with transition and place labeling functions:

Definition 2 A labeled DSPN (LDSPN) is a triplet $\mathcal{LN} = (\mathcal{N}, \lambda, \psi)$, where $\mathcal{N}$ is a DSPN, as in def.(1); $\lambda : T \rightarrow 2^{L^T}$ and $\psi : P \rightarrow 2^{L^P}$ are the transition and place labeling functions, respectively, that assign to a transition/place a set of labels (or the empty set).

2.1 Definition of basic state machines

UML state machines (SM) can be used to specify behavior of software components (or objects) [UML2 (2010)]. A SM basically consists of *states* and *transitions*. States model situations during which some invariant condition holds, such as the component performing some computational *activity* or waiting for some external *event* to occur. Transitions between states (*external* transitions), labelled as *event/action*, represent how a component in a source state reacts upon receiving an *event*, so performing an *action*, and then entering the target state. The action can also be the sending of a new event to other component. Transitions that do not specify an event are named *completion* transitions, they are implicitly triggered by a *completion event* and fire as soon as the *activity* in the state completes. States can contain *entry/exit* actions and *internal* transitions; entry (exit) actions are executed when the state is entered (exited), internal transitions do not cause a state change and, when triggered, they fire without exiting nor entering the source state.

Let us assume the system made of n state machines that cooperate by exchanging events, $Sys =< \{\mathcal{SM}_i\}_{i=1}^{n}, \mathcal{E}, \mathcal{L}_a >$, where $\mathcal{E}$ is the set of events, that includes the completion event λ, and $\mathcal{L}_a$ the set of actions.

Definition 3 A state machine is a tuple:
$$\mathcal{SM} = (\Sigma, A_{entry}, A_{exit}, A_{do}, \Theta, E_{trigger}, A_{effect}, source, target)$$

- $\Sigma = \Sigma_{ini} \cup \Sigma_{final} \cup \Sigma_{simple}$ is the set of initial ($|\Sigma_{ini}| = 1$), final (Σ_{final}) and simple (Σ_{simple}) states;
- $A_{entry} : \Sigma \hookrightarrow \mathcal{L}_a$ assigns to a state an optional entry action;
- $A_{exit} : \Sigma \hookrightarrow \mathcal{L}_a$ assigns to a state an optional exit action;
- $A_{do} : \Sigma \hookrightarrow \mathcal{L}_a$ assigns to a state an optional activity;
- $\Theta \subseteq \Sigma \times \Sigma$ is the set of transitions including external, completion and internal transitions, i.e., $\Theta = \Theta_{out} \cup \Theta_{\lambda} \cup \Theta_{int}$;
- $E_{trigger} : \Theta \rightarrow \mathcal{E}$ assigns to each transition a trigger event;
- $A_{effect} : \Theta \hookrightarrow \mathcal{L}_a \bigcup \{\mathcal{SM}_i.ev\}$ assigns to a transition an optional action or the dispatch of an event to other SM $\mathcal{SM}_i$, $ev \in \mathcal{E} \setminus \{\lambda\}$;
- $source : \Theta \rightarrow \Sigma$, where $\forall (s,t,s) \in \Theta : source((s,t,s')) = s$; and
- $target : \Theta \rightarrow \Sigma$, where $\forall (s,t,s) \in \Theta : target((s,t,s')) = s'$.

Following assumptions come from the fact that we are not dealing with guards, however they can be easily overcome following indications in [Merseguer (2003)]:
1. $\forall t, t' \in \Theta_{out} \cup \Theta_{int} : source(t) = source(t') \ \wedge \ target(t) = target(t') \Rightarrow E_{trigger}(t) \neq E_{trigger}(t')$.
2. $\forall t, t' \in \Theta_{\lambda} : source(t) \neq source(t')$.

3 Formal translation of UML state machines

The operational semantics of a SM, informally described in UML2 (2010), is herein interpreted by the formal DSPN operational semantics.

3.1 Translation of simple states

We propose four different translations for a SM state, shown in Figure 1(a..d), depending on whether the state includes activity and/or completion transition. Note that apart of the actions, the Figure also depicts interface transitions to later compose this model with the models of Θ_{int} (interfaces $t_{end\_int}, t_{int1}, t_{int2}$), Θ_{out} (interfaces t_{out1}, t_{out2}) and Θ_{λ} (interface t_{ce}). Figure 1(e,f) are variations of the previous ones to show the case when there is no entry action. Although more cases exist, they are just variations of these first four and all them are formalized below.

These LDSPNs give a formal interpretation of the informal execution semantics UML describes for a state. So, a token in ini_S represents the entrance in the state that is always followed by the execution of the *entry* action. When the do-activity $A_{do}(S)$ exists, then the timed transition t_{do} will be the next to be executed. Note that the exit action $A_{exit}(S)$ is not represented in Figure 1 since it comes after the execution of an external transition.

The LDSPN in Figure 1(c) models a variation of (a) to cope with completion transitions. The only difference is that interface transitions t_{int2} and

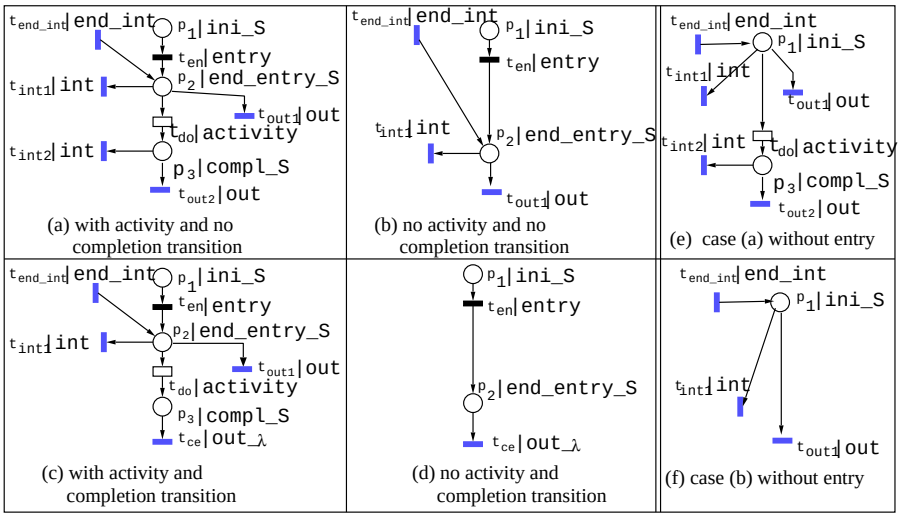

Fig. 1 Translation of a simple state

t_{out2} are omitted, so internal and external transitions cannot take place after the activity. Indeed, transition t_{ce}, whose firing represents the triggering of the completion event, forces the exit of the state. Figure 1(d) has not interfaces for external and internal transitions since they are not allowed for states without activity but with completion transition.

Let $\chi, \overline{\chi}$ be the non empty-set and empty-set indicator functions, respectively[1]. Then, the sets of places and transitions of the basic LDSPN $\mathcal{BN}_S$, which interprets the state $S \in \Sigma$, are defined as follows:

$$
\begin{aligned}
P_{\mathcal{BN}_S} &= \{p_1\} \cup \chi(A_{entry}(S)) \cdot \{p_2\} \cup \chi(A_{do}(S)) \cdot \{p_3\};\\
T_{\mathcal{BN}_S} &= \chi(A_{entry}(S)) \cdot \{t_{en}\} \cup \chi(A_{do}(S)) \cdot \{t_{do}\} \cup \chi(\Theta_{\lambda_S}) \cdot \{t_{ce}\} \cup\\
&\quad \chi(\Theta_{int_S}) \Big[\widehat{\chi} \cdot \{t_{int1}, t_{end\_int}\} \cup \Big(\chi(A_{do}(S)) \cdot \overline{\chi}(\Theta_{\lambda_S})\Big) \cdot \{t_{int2}\}\Big] \cup\\
&\quad \chi(\Theta_{out_S}) \Big[\widehat{\chi} \cdot \{t_{out1}\} \cup \Big(\chi(A_{do}(S)) \cdot \overline{\chi}(\Theta_{\lambda_S})\Big) \cdot \{t_{out2}\}\Big],
\end{aligned}
$$

where $\Theta_{out_S}, \Theta_{int_S}$ and Θ_{λ_S} are the sets of external, internal and completion transitions of S, respectively, and $\widehat{\chi} = \chi(A_{do}(S)) + \overline{\chi}(A_{do}(S)) \cdot \overline{\chi}(\Theta_{\lambda_S})$. The LDSPN has no inhibitor arcs, i.e., $H_{\mathcal{BN}_S}(p,t) \equiv 0$, while the input and output functions are defined as:

$$
I_{\mathcal{BN}_S}(p_1,t) = \begin{cases} 1 & \text{if } t \in \chi(A_{entry}(S)) \cdot \{t_{en}\} \cup \overline{\chi}(A_{entry}(S)) \cdot \Big(\chi(A_{do}(S)) \cdot \{t_{do}\} \cup \\ & \widehat{\chi} \cdot \chi(\Theta_{int_S}) \cdot \{t_{int1}\} \cup \widehat{\chi} \cdot \chi(\Theta_{out_S}) \cdot \{t_{out1}\} \cup \overline{\chi}(A_{do}(S)) \cdot \chi(\Theta_{\lambda_S}) \cdot \{t_{ce}\}\Big) \\ 0 & \text{otherwise} \end{cases}
$$

[1] $\chi(Q) = \begin{cases} 0 & \text{if } Q = \emptyset \\ 1 & \text{if } Q \neq \emptyset \end{cases}$ and $\overline{\chi}(Q) = 1 - \chi(Q)$.

$$I_{\mathcal{BN}_S}(p_2,t) = \begin{cases} 1 & \text{if } t \in \chi(A_{entry}(S)) \cdot \Big(\chi(A_{do}(S)) \cdot \{t_{do}\} \cup \widehat{\chi} \cdot \chi(\Theta_{int_S}) \cdot \{t_{int1}\} \cup \\ & \qquad \widehat{\chi} \cdot \chi(\Theta_{out_S}) \cdot \{t_{out1}\} \cup \overline{\chi}(A_{do}(S)) \cdot \chi(\Theta_{\lambda_S}) \cdot \{t_{ce}\}\Big) \\ 0 & \text{otherwise} \end{cases}$$

$$I_{\mathcal{BN}_S}(p_3,t) = \begin{cases} 1 & \text{if } t \in \chi(A_{do}(S)) \cdot \Big(\chi(\Theta_{int_S}) \cdot \overline{\chi}(\Theta_{\lambda_S}) \cdot \{t_{int2}\} \cup \\ & \chi(\Theta_{out_S}) \cdot \overline{\chi}(\Theta_{\lambda_S}) \cdot \{t_{out2}\} \cup \ \chi(\Theta_{\lambda_S}) \cdot \{t_{ce}\}\Big) \\ 0 & \text{otherwise} \end{cases}$$

$$O_{\mathcal{BN}_S}(p_1,t) = \begin{cases} 1 & \text{if } t \in \overline{\chi}(A_{entry}(S)) \cdot \chi(\Theta_{int_S}) \cdot \widehat{\chi} \cdot \{t_{end\_int}\} \\ 0 & \text{otherwise} \end{cases}$$

$$O_{\mathcal{BN}_S}(p_2,t) = \begin{cases} 1 & \text{if } t \in \chi(A_{entry}(S)) \cdot \Big(\{t_{en}\} \ \cup \chi(\Theta_{int_S}) \cdot \widehat{\chi} \cdot \{t_{end\_int}\}\Big) \\ 0 & \text{otherwise} \end{cases}$$

$$O_{\mathcal{BN}_S}(p_3,t) = \begin{cases} 1 & \text{if } t \in \chi(A_{do}(S)) \cdot \{t_{do}\} \\ 0 & \text{otherwise} \end{cases}$$

The initial marking $M^0_{\mathcal{BN}_S}$ and the transition delay $\Lambda_{\mathcal{BN}_S}$ functions will be defined in Section 4, considering the MARTE-DAM annotations. The labeling functions for places and transitions are defined as:

$$\psi_{\mathcal{BN}_S}(p) = \begin{cases} ini\_\mathcal{S} & \text{if } p = p_1 \\ end\_entry\_\mathcal{S} & \text{if } p = p_2 \\ compl\_\mathcal{S} & \text{if } p = p_3 \end{cases} \quad \lambda_{\mathcal{BN}_S}(t) = \begin{cases} A_{entry}(S) & \text{if } t = t_{en} \\ A_{do}(S) & \text{if } t = t_{do} \\ end\_int & \text{if } t = t_{end\_int} \\ int & \text{if } t = \{t_{int1}, t_{int2}\} \\ out & \text{if } t = \{t_{out1}, t_{out2}\} \\ \lambda_S & \text{if } t = t_{ce} \end{cases}$$

Finally, we remark that initial and final states match the translation herein presented. Consider that in UML final states have not actions neither external nor internal transitions, while initial ones only own an external transition labeled by the event "create".

3.2 Translation of transitions

Figure 2 proposes a translation for transitions, that formally interprets the informal execution semantics UML assigns them. In Fig. 2(a), $t \in \Theta_{out_{s_1}}$ with $source(t) = s_1$, $target(t) = s_2$, $A_{effect}(t) = action$, $E_{trigger}(t) = evx$ and $exit = A_{exit}(s_1)$. The execution semantics of the LDSPN states that when the SM is executing in s_1 and receives event evx, it is accepted and if *action* and/or *exit* exist, they are executed, finally s_2 is entered. Note that if evx reaches the SM when it is not executing s_1, the event is lost. So, t_1 has priority over t_3; in the final composed model (eq.1), t_1 will have another input place to indeed represent execution of s_1. It is worth noting that the effect of a transition can be a send event to other SM, e.g. $A_{effect}(t) = \mathcal{SM}_i.evy$. The dotted square, in Fig. 2, offers the corresponding translation, in this case the label of t_2 will be $\mathcal{SM}_i.evy$. Figure 2(a) is also valid for an external transition t where $source(t) = target(t)$. Then, Figure 2(b) shows that the only difference between an internal transition and an external with same target and source is that the *exit* action of the state is not executed by the former. A formalization

of these LDSPNs can be obtained similarly as the one developed for the basic state (sub-sect. 3.1).

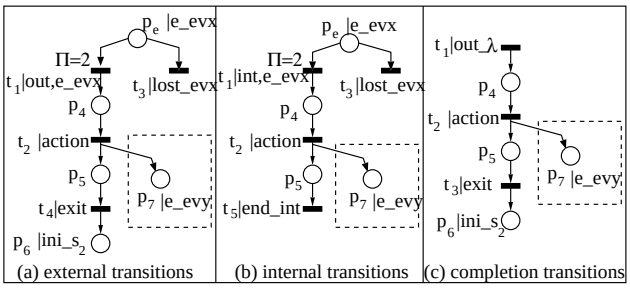

Fig. 2 Translation of transitions.

3.3 Composition of simple states and transitions

Given a state S with h internal transitions and l external/completion transitions, we get $h + l$ LDSPN models plus the $\mathcal{BN}_S$, that need to be combined to get the LDSPN $\mathcal{LN}_S$. We apply the LDSPN composition operator defined in Appendix A.

Let $Lev^P = \{e\_evx, \forall evx \in \mathcal{E} \setminus \{\lambda\}\}$ and $Lstate^P = \{ini\_target, \forall target \in \Sigma\}$. We first compose sub-models of internal and external transitions:

$$INT = \overset{i=1,..,h}{||}_{Lev^P,\emptyset} INT_i, \qquad OUT = \overset{j=1,..,l}{||}_{Lstate^P \cup Lev^P,\emptyset} OUT_j.$$

Then, the LDSPN $\mathcal{LN}_S$ is obtained by:

$$\mathcal{LN}_S = (INT \;||_{Lev^P,\emptyset}\; OUT) \;||_{Lev^P,Ltr^T}\; \mathcal{BN}_S. \tag{1}$$

where $Ltr^T = \{int, end\_int, out, out\_\lambda\}$.

3.4 Composition of state machines and the final system

Definition 4 The LDSPN that interprets the state machine $\mathcal{SM}$ is given by:

$$\mathcal{LN}_{\mathcal{SM}} = \overset{s\in\Sigma}{||}_{Lev^P \cup Lstate^P,\emptyset} \mathcal{LN}_s.$$

Definition 5 Given a system Sys made of n SMs that cooperate by exchanging events and being $Lev^P_{Sys} = \{e\_evx, \forall evx \in \mathcal{E} \setminus \{\lambda\}\}$. The LDSPN that interprets Sys is given by: $\mathcal{LN}_{Sys} = ||^{i=1..n}_{Lev^P_{Sys},\emptyset} \mathcal{LN}_{\mathcal{SM}_i}$.

4 Formal translation of dependability annotations

One of the main issues in dependability modeling and analysis is the definition of the system fault assumption in terms of: 1) which components can be affected by faults and in which states, 2) the maximum number of faults that can concurrently affect the system components and 3) the fault characterization, such as the fault occurrence rate.

Unfortunately, UML does not provide sufficient capabilities neither for timing specification, which is a fundamental feature in systems quantitative evaluation, nor for dependability modeling. The DAM profile [Bernardi et al (2009)] overcomes this drawback extending UML in a *lightweight* fashion (i.e., through the use of UML stereotypes and tagged-values). DAM has been defined as a specialization of the OMG standard profile MARTE (2008); in particular, it inherits from MARTE the capability of supporting the timing specification and, in addition, it enables the specification of dependability input parameters and metrics (Figure 3 - top side).

In the following, we provide a formal definition of a SM enriched with those MARTE-DAM annotations which enable the derivation of a dependability LDSPN model.

Definition 6 A MARTE-DAM profiled state machine is a tuple:

$$\widehat{\mathcal{SM}} = \langle \mathcal{SM}, pool, demand, ft\_rate, failure, \mathcal{L}_{pred} \rangle$$

where:

- $\mathcal{SM}$ is a state machine, as in def. 3 (sect. 2), where the trigger event function is refined to include the (local) fault events, i.e., $\widehat{E}_{trigger} : \Theta \rightarrow \mathcal{E} \cup \mathcal{E}_{ft}^{\mathcal{SM}}$;
- $pool : \{s_i\} \rightarrow \mathbb{N}$ is a function that assigns to a single state, i.e., $\{s_i\} \subseteq \Sigma$, the initial number of components/objects in such a state;
- $demand : A_{do} \hookrightarrow \mathbb{R} \times \{const, mean\}$ is a partial function that assigns to a do-activity the host demand required to execute it, together with the type of statistical qualifier (i.e., a constant value or a mean value);
- $ft\_rate : \mathcal{E}_{ft}^{\mathcal{SM}} \rightarrow \mathbb{R}$ is a function that assigns to each fault event its occurrence rate;
- $failure : \Sigma \rightarrow \{true, false\}$ is a function that assigns a boolean value to each state, the *true* value identifies the failure state;
- $\mathcal{L}_{pred} = \{(type, var)\}$ is a set of pairs, where the first element $type \in \{occurrenceDist, MTTF, occurrenceRate\}$ is the type of dependability metric and the second one $var \in \mathbb{R}$ is the corresponding output parameter variable.

4.1 Mapping of the MARTE specifications

The *pool* and *demand* functions of $\widehat{\mathcal{SM}}$ are the formal counterpart of the MARTE annotations attached, respectively, to SM state and do-activities

shown in Figure 3 (top). In particular the *pool* function corresponds to the *poolSize* tag of the *PARunTInstance* stereotyped state, while the *demand* function corresponds to the *hostDemand* tag of the *GaStep* stereotyped do-activity. Such information is used to define the place marking $M^0_{\mathcal{BN}_S}$ and transition weight/delay $\Lambda_{\mathcal{BN}_S}$ functions of the LDSPN $\mathcal{BN}_S$ associated to a given state $S \in \Sigma$:

$$p \in P_{\mathcal{BN}_S} \mapsto M^0_{\mathcal{BN}_S}(p) = \begin{cases} pool(S) & \text{if } \psi_{\mathcal{BN}_S}(p) = ini\_S \\ 0 & \text{otherwise} \end{cases}$$

$$t \in T_{\mathcal{BN}_S} \mapsto \Lambda_{\mathcal{BN}_S}(t) = \begin{cases} \pi_1(demand(A_{do}(S))) & \text{if } \lambda_{\mathcal{BN}_S}(t) = A_{do}(S) \\ 1 & \text{otherwise} \end{cases}$$

where π_1 is the left projection function. Observe that the timed transition of the LDSPN modelling the do-activity $A_{do}(S)$ will be either determistic or exponential according whether $\pi_2(demand(A_{do}(S)))$ is equal to either *const* or *mean*, respectively, where π_2 is the right projection function.

4.2 Mapping of the DAM fault specifications

The function ft_rate of $\widehat{\mathcal{SM}}$ is used to create new LDSPNs, i.e., the fault generators, that create fault event occurrences. We assume that fault events are local to $\widehat{\mathcal{SM}}$ and a fault event is the trigger event of a SM transition stereotyped as *DaFaultGenerator* (Figure 3, top - SM Process). Then, for a given fault event $f \in \mathcal{E}^{\mathcal{SM}}_{ft}$, the fault generator LDSPN is defined as follows:

$$\mathcal{LN}_f = (\ P = \{p, p'\}, \{t\} \in T_E, I(x,t) = \begin{cases} 1 & \text{if } x = p \\ 0 & \text{if } x = p' \end{cases},$$

$$O(x,t) = \begin{cases} 0 & \text{if } x = p \\ 1 & \text{if } x = p' \end{cases}, H(P,t) = 0, M^0(x) = \begin{cases} 1 & \text{if } x = p \\ 0 & \text{if } x = p' \end{cases},$$

$$\Phi(t) = 0, \Lambda(t) = ft\_rate(f), \lambda(t) = f\_occ, \psi(x) = \begin{cases} no\_f & \text{if } x = p \\ e\_f & \text{if } x = p' \end{cases}).$$

The fault generator LDSPNs are included in the composition to get the LDSPN of $\widehat{\mathcal{SM}}$:

$$\mathcal{LN}_{\widehat{\mathcal{SM}}} = \mathcal{LN}_{\mathcal{SM}} \underset{Lft^P, \emptyset}{||} FT \tag{2}$$

where Lft^P is the set of place labels of fault events, i.e., $Lft^P = \{e\_f, \forall f \in \mathcal{E}^{\mathcal{SM}}_{ft}\}$, $\mathcal{LN}_{\mathcal{SM}}$ is the LDSPN of $\mathcal{SM}$ (sub-sect. 3.4, def. 4) and FT is the LDSPN including all the fault generators associated to $\widehat{\mathcal{SM}}$, i.e.:

$$FT = \underset{\emptyset,\emptyset}{\overset{i=1,..,|\mathcal{E}^{\mathcal{SM}}_{ft}|}{||}} \mathcal{LN}^i_f.$$

Observe that the maximum number of concurrent active faults in a $\widehat{\mathcal{SM}}$ is equal to $|\mathcal{E}^{\mathcal{SM}}_{ft}|$, while the maximum number of concurrent active faults affecting the whole system is given by the sum of the fault events of all the n SM, i.e., $\sum_{i=1}^{n} |\mathcal{E}^{\mathcal{SM}_i}_{ft}|$.

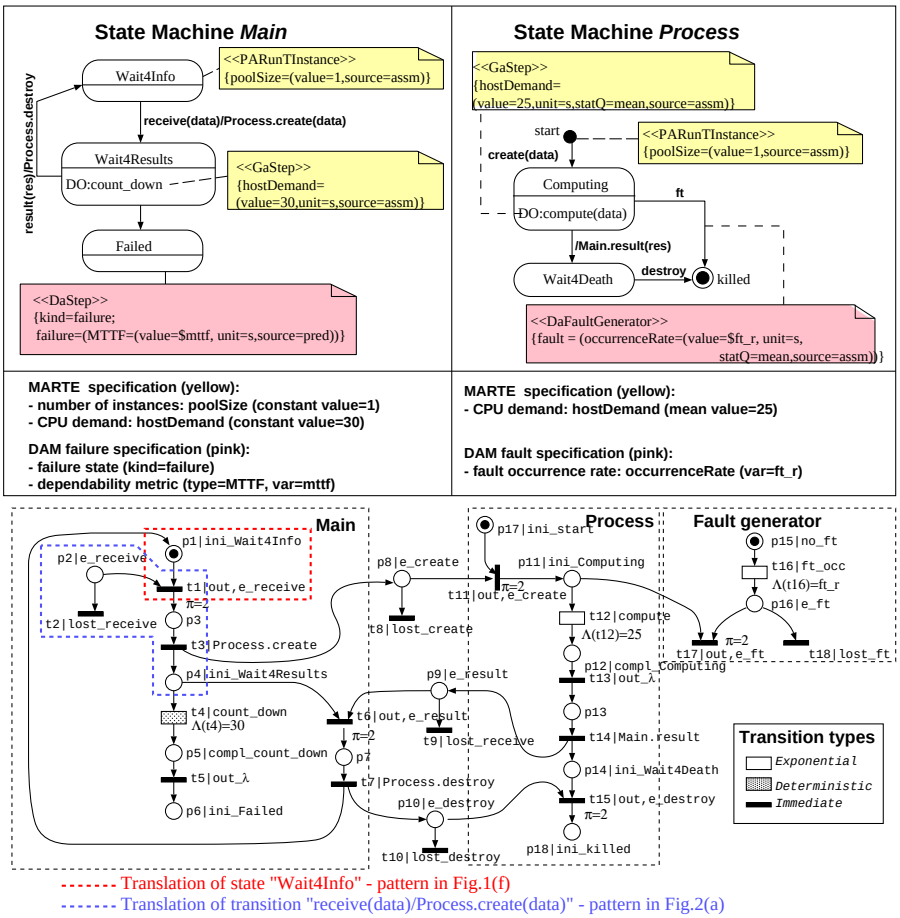

Fig. 3 MARTE-DAM annotated SMs (top) and the corresponding LDSPN $\mathcal{LN}_{Sys}$ (bottom).

5 System analysis

Besides the system fault assumptions, also failure assumptions and dependability metrics need to be specified as well, then to enable dependability analysis of the LDSPN system $\mathcal{LN}_{Sys}$ (sub-sect. 3.4, def. 5).

Concretely, failure assumptions specify the failure states of the system, those representing an interruption of a system service. Their definition depends on the system requirements and, when several services are specified, different (service) failure modes can be assumed. To simplify the treatment, we will assume that a SM can have at most one failure state[2]. This is a reason-

[2] Observe that, this assumption is not an actual restriction since the DAM profile supports the specification of combinations of single SM failure states through logical AND/OR expressions.

able assumption in DES where there is a monitor that is in charge of detecting anomalous behavior of the set of controlled processes, which implement a given system service. Then, the dependability of the service is evaluated by computing the metric associated to the failure state of the monitor, e.g., the Mean Time To Failure (MTTF) specified in Figure 3 (top - SM Main).

The DAM annotation, attached to the SM state stereotyped as *DaStep* (Figure 3, top - SM Main), is formalized through the $failure, \mathcal{L}_{pred}$ features of $\widehat{\mathcal{SM}}$ (sect. 4, def. 6). In particular, $failure$ is used to identify the SM failure state and $\mathcal{L}_{pred}$ to specify the dependability metrics to be computed.

On the other hand, dependability metrics associated to $\mathcal{LN}_{Sys}$ can be defined in terms of the following reward function [Goseva-Popstojanova and Trivedi (2000)]:

$$r(M) = \begin{cases} 1 & \text{if } M \in O \\ 0 & \text{if } M \in F. \end{cases}$$

which partitions the set of reachable markings $RS(M^0)$ of $\mathcal{LN}_{Sys}$ into two subsets of markings: O that represents the set of operational system states and F that represents the system failure states. Then, the problem of translating the failure specification of $\widehat{\mathcal{SM}}$ into LDSPN dependability metrics boils down in:

1. Defining a mapping of SM failure state onto the subset of markings $F \subseteq RS(M^0)$ and,
2. For each pair $(type, var) \in \mathcal{L}_{pred}$, assigning to the output parameter var the corresponding value, according to the *type* of dependability metric.

Considering the first point, let sm be a SM characterized by a failure state and $\mathcal{M}^{sm} : \Sigma_{sm} \rightarrow 2^{P_{sm}}$ be the function that assigns to each state of sm the set of corresponding places of the LDSPN $\mathcal{LN}_{sm}$, i.e.:

$$s \in \Sigma_{sm} \mapsto \Big\{ p \in P_{sm} : \psi_{sm}(p) \in \{ini\_s, end\_entry\_s, compl\_s\} \Big\}.$$

Observe that $\mathcal{M}^{sm}(s) \subseteq P_{sm}$ is also a subset of the set of places P_{Sys} of $\mathcal{LN}_{Sys}$, by definition of the place composition operator (Appendix A) and of def. 5 (sub-sect. 3.4). The set of markings of $\mathcal{LN}_{Sys}$, representing the system failure states (associated to the failure state of sm), is then defined as:

$$F = \{M \in RG(M^0) : \exists p \in P_{sm}^{fail} : M(p) \geq 1\}$$

where $M(p)$ is the marking of place p and $P_{sm}^{fail} = \{p \in \mathcal{M}^{sm}(s) : failure(s) = true\} \subseteq P_{Sys}$. The set of markings representing the system operational states is then $O = RG(M^0) \setminus F$.

Finally, considering a pair $(type, var) \in \mathcal{L}_{pred}$, the *type* indicates the dependability metric to be computed. Each type represents a failure tag of the DAM annotation and the output parameter var is the corresponding tagged-value. Table 1 summarizes the reliability metrics that can be evaluated. A formula is assigned to each var that corresponds to the LDSPN dependability metric, where $\sigma_M(t) = Pr\{X(t) = M\}$ is the (transient) probability of the

Table 1 Reliability metrics

type	var	description
$MTTF$	$\sum_{M \in O} \int_0^\infty \sigma_M(\tau) d\tau$	mean time to failure
occurrenceRate	$1/MTTF$	failure occurrence rate
occurrenceDist	$\sum_{M \in F} \sigma_M(t)$	unreliability function $(1 - R(t))$

marking process associated to the LDSPN being in marking M at time instant $t \geq 0$. Availability metrics, which are meaningful only for repairable systems, can be defined in a similar manner [Goseva-Popstojanova and Trivedi (2000)].

6 Related work

One of the open issues raised by Dingel et al (2009) is the existing gap between discrete event systems (DES) theory and software engineering practice. In this paper, we provide a contribution to address such issue, by showing that UML-SM models have their counterpart in a largely used formalism in DES, i.e., Petri Nets. At this regard, Basile et al (2008) develop a complete case study to show the combined use of UML and Petri nets for modeling automation systems and Risco-Martin et al (2007) propose a framework to integrate DES and UML, in this case to provide an execution environment.

Several works have addressed the dependability analysis based on UML, herein we mention those that consider UML SMs in their proposal. Bondavalli et al (2001) propose an integrated UML environment considering structural and behavioral diagrams. In particular, SMs are converted into PROMELA to feed the SPIN model checker. Unlike our proposal, the transformation relies upon an intermediate model and no standard UML profiles were available at time of the publication. Huszerl et al (2002) define a transformation of SMs into Stochastic Reward Nets (SRN) to enable performance analysis, under erroneous state and faulty behavior assumptions. Mustafiz et al (2008) use probabilistic statecharts to derive a Markov chain for quantitative assessment of safety and reliability. The aforementioned works do not focus on DES.

Trowitzsch et al (2007) add a new functionality to TimeNet tool, the modeling of stochastic SMs and their automatic transformation to extended Stochastic Petri Nets for performance and dependability analysis. Actually, they only use performance extensions and not dependability specific ones. Moreover, only one state machine is used at a time, so they do not use communicating state machines as we do in our work.

7 Conclusion and future work

The paper develops an approach, built on [Merseguer (2003)], for the analysis of reactive software systems equipped with a dependability specification.

Concretely, the approach formalizes the translation of UML-SMs and DAM into DSPNs and also formalizes the subsequent analyses. Therefore, assuming the interest of software engineers in the use of UML-SMs for the modeling of dependable DES, the paper proves that: 1) DAM allows the definition of fault and failure assumptions in UML-SMs models; 2) DAM is compatible with standard MARTE annotations to characterize the quantitative aspects of the system (processor demands); 3) the UML-DAM-MARTE non-formal models can be interpreted in terms of a formal model, DSPNs; and 4) the DSPN model is useful to compute dependability metrics, such as MTTF, for the software engineers to demonstrate how dependable their models are. Indeed, our formalizations aim at easing tool support for software engineers. Papyrus (2010) tool implements MARTE, while DAM has been implemented by GISED group (2011), besides, SMs translation into Petri nets was also accomplished by GISED group (2006). However, tool support for the formal analysis is still on-going work.

Future work envisions different research directions since there is a lot of work to bridge the gap between DES and software engineering. For example, how to apply non-formal software engineering methods (e.g., hierarchical methodologies) to DES to avoid state explosion problems. More pragmatically, the combined use of MARTE and DAM needs more blueprints and case studies to bring closer the profiles to practitioners.

Acknowledgements

The authors thank the anonymous reviewers for their valuable help to improve this work.

References

Ajmone Marsan M, Chiola G (1987) On Petri nets with deterministic and exponentially distributed firing times. In: Advances in Petri Nets 1987, covers the 7th European Workshop on Applications and Theory of Petri Nets, Springer-Verlag, London, UK, pp 132–145

Avizienis A, Laprie JC, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 01(1):11–33, DOI http://doi.ieeecomputersociety.org/10.1109/TDSC.2004.2

Basile F, Chiacchio P, Grosso DD (2008) Modelling automation systems by UML and Petri nets. In: Proceedings of of the 9th International Workshop on Discrete Event Systems (WODES 2008), IEEE Explore, pp 308–313

Bernardi S, Merseguer J, Petriu D (2009) A Dependability Profile within MARTE. Journal of Software and Systems Modeling DOI: 10.1007/s10270-009-0128-1.

Bondavalli A, Dal Cin M, Latella D, Majzik I, Pataricza A, Savoia G (2001) Dependability analysis in the early phases of UML-based system design. International Journal of Computer Systems Science & Engineering 16(5):265–275

Dingel J, Rudie K, Dragert C (2009) Bridging the gap: Discrete-event systems for software engineering. In: Canadian Conference on Computer Science & Software Engineering (C3S2E 2009), ACM, Montreal, Quebec, Canada, pp 67–71

Donatelli S, Franceschinis G (1996) The PSR methodology: Integrating hardware and software models. In: Billington J, Reisig W (eds) Application and Theory of Petri Nets, Springer, LNCS, vol 1091, pp 133–152

GISED group (2006) http://argospe.tigris.org, Universidad de Zaragoza
GISED group (2011) http://webdiis.unizar.es/GISED/?q=tools, Universidad de Zaragoza
Goseva-Popstojanova K, Trivedi KS (2000) Stochastic modeling formalisms for dependability, performance and performability. In: Haring G, Lindemann C, Reiser M (eds) Performance Evaluation: Origins and Directions, Springer, Lecture Notes in Computer Science, vol 1769, pp 403–422
Huszerl G, Majzik I, Pataricza A, Kosmidis K, Dal Cin M (2002) Quantitative Analysis of UML Statechart Models of Dependable Systems. The Computer Journal 45(3):260–277
Lilius J, Paltor IP (1999) The semantics of UML state machines. Tech. rep., Turku Centre for Computer Science, Åbo Akademi University, Turku (Finland)
MARTE (2008) A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Object Management Group. Document Number: ptc/2008-06-09
Merseguer J (2003) Software performance engineering based on UML and Petri nets. PhD thesis, University of Zaragoza, Spain
Mustafiz S, Sun X, Kienzle J, Vangheluwe H (2008) Model-driven assessment of system dependability. Software and System Modeling 7(4):487–502
Papyrus (2010) www.papyrusuml.org/, CEA LIST
Risco-Martin J, Mittal S, Zeigler B, de la Cruz J (2007) From UML state charts to DEVS state machines using XML. In: Workshop on Multi-Paradigm Modeling within MoDELS, Nashville, Tennessee (USA), pp 35–48
Trowitzsch J, Jerzynek D, Zimmermann A (2007) A toolkit for performability evaluation based on stochastic UML state machines. In: Proceedings of the 2nd International Conference on Performance Evaluation Methodolgies and Tools (VALUETOOLS 2007), ACM, Nantes, France, p 30
UML2 (2010) UML Unified Modeling Language: Superstructure. Object Management Group, http:/www.omg.org, version 2.3

A LDSPN composition operator

In general, more than one label can be associated to a transition (place). However, in the composition of two nets, we actually consider at most one label per transition (place): with this restriction we can use a simplified version of the composition operator [Donatelli and Franceschinis (1996)]. Given two LDSPN $\mathcal{LN}_1 = (\mathcal{N}_1, \lambda_1, \psi_1)$ and $\mathcal{LN}_2 = (\mathcal{N}_2, \lambda_2, \psi_2)$, the LDSPN $\mathcal{LN} = (\mathcal{N}, \lambda, \psi)$:

$$\mathcal{LN} = \mathcal{LN}_1 \underset{L_P, L_T}{||} \mathcal{LN}_2$$

resulting from the composition over the sets of labels L_P and L_T is defined as follows.

Let $E_P = L_P \cap \psi_1(P_1) \cap \psi_2(P_2)$ and $E_T = L_T \cap \lambda_1(T_1) \cap \lambda_2(T_2)$ be the subsets of L_P and of L_T, respectively, comprising place and transition labels that are common to the two LDSPNs, P_1^l (T_1^l) be the set of places (transitions) of $\mathcal{LN}_1$ that are labeled l and $P_1^{E_P}$ ($T_1^{E_T}$) be the set of all places (transitions) in $\mathcal{LN}_1$ that are labeled with a label in E_P (E_T). Same definitions apply to $\mathcal{LN}_2$.

Then: $P = P_1 \backslash P_1^{E_P} \cup P_2 \backslash P_2^{E_P} \cup \bigcup_{l \in E_P} \{P_1^l \times P_2^l\}$, $T = T_1 \backslash T_1^{E_T} \cup T_2 \backslash T_2^{E_T} \cup \bigcup_{l \in E_T} \{T_1^l \times T_2^l\}$, the functions $F \in \{I(), O(), H()\}$ are equal to:

$$F(p,t) = \begin{cases} F_1(p,t) & \text{if } p \in P_1 \backslash P_1^{E_P}, t \in T_1 \backslash T_1^{E_T} \\ F_1(p,t_1) & \text{if } p \in P_1 \backslash P_1^{E_P}, t \equiv (t_1,t_2) \in \bigcup_{l \in E_T} \{T_1^l \times T_2^l\} \\ F_1(p_1,t) & \text{if } p \equiv (p_1,p_2) \in \bigcup_{l \in E_P} \{P_1^l \times P_2^l\}, t \in T_1 \backslash T_1^{E_T} \\ F_2(p,t) & \text{if } p \in P_2 \backslash P_2^{E_P}, t \in T_2 \backslash T_2^{E_T} \\ F_2(p,t_2) & \text{if } p \in P_2 \backslash P_2^{E_P}, t \equiv (t_1,t_2) \in \bigcup_{l \in E_T} \{T_1^l \times T_2^l\} \\ F_2(p_2,t) & \text{if } p \equiv (p_1,p_2) \in \bigcup_{l \in E_P} \{P_1^l \times P_2^l\}, t \in T_2 \backslash T_2^{E_T} \\ min\{F_1(p_1,t_1), F_2(p_2,t_2)\} & \text{if } p \equiv (p_1,p_2) \in \bigcup_{l \in E_P} \{P_1^l \times P_2^l\}, \\ & \quad t \equiv (t_1,t_2) \in \bigcup_{l \in E_T} \{T_1^l \times T_2^l\} \end{cases}$$

Functions $F \in \{\Phi(), \Lambda()\}$ are equal to:

$$F(t) = \begin{cases} F_1(t) & \text{if } t \in T_1 \backslash T_1^{E_T} \\ F_2(t) & \text{if } t \in T_2 \backslash T_2^{E_T} \\ F_2(t_2) & \text{if } t \equiv (t_1,t_2) \in \bigcup_{l \in E_T} \{T_1^l \times T_2^l\} \end{cases}$$

The initial marking function is equal to:

$$M^0(p) = \begin{cases} M_1^0(p) & \text{if } \ p \in P_1 \backslash P_1^{E_P} \\ M_2^0(p) & \text{if } \ p \in P_2 \backslash P_2^{E_P} \\ M_1^0(p_1) + M_2^0(p_2) & \text{if } \ p \equiv (p_1,p_2) \in \bigcup_{l \in E_P} \{P_1^l \times P_2^l\} \end{cases}$$

Finally, the labeling functions for places and transitions are respectively equal to:

$$\psi(x) = \begin{cases} \psi_1(x) & \text{if } \ x \in P_1 \backslash P_1^{E_P} \\ \psi_2(x) & \text{if } \ x \in P_2 \backslash P_2^{E_P} \\ \psi_1(p_1) \cup \psi_2(p_2) & \text{if } \ x \equiv (p_1,p_2) \in \bigcup_{l \in E_P} \{P_1^l \times P_2^l\} \end{cases}$$

$$\lambda(x) = \begin{cases} \lambda_1(x) & \text{if } \ x \in T_1 \backslash T_1^{E_T} \\ \lambda_2(x) & \text{if } \ x \in T_2 \backslash T_2^{E_T} \\ \lambda_1(t_1) \cup \lambda_2(t_2) & \text{if } \ x \equiv (t_1,t_2) \in \bigcup_{l \in E_T} \{T_1^l \times T_2^l\} \end{cases}$$

The relation being associative with respect to place superposition, we use also as an n-operand by writing $\mathcal{LN} = |\,|_{\emptyset, L_P}^{k=1,..,K} \mathcal{LN}_k$.