Building Petri net scenarios for dependable automation systems

Simona Bernardi and Susanna Donatelli
Dipartimento di Informatica
Università di Torino, 10149 Torino, Italy
\{bernardi,susi\}@di.unito.it

Abstract

This paper considers the problem of building Petri net based evaluation scenarios for dependable automation systems and proposes a modelling process in which as much information as possible is extracted from a high level description of system entities and their relationships using the Class Diagram notation. Having fixed the domain to that of dependable automation systems also allows to reuse a number of predefined models. The class of nets of reference is that of Generalized Stochastic Petri Net, and their colored extension Stochastic Well Formed Nets, so that the system under study can be both validated and evaluated.

1. Introduction

Building models for dependability is not an easy task, since there is the additional complexity of modelling the presence of faults, errors and failure, and their relationships, and the influence they have on the model of the normal system behaviour. The problem grows even bigger when the system includes software mechanisms that aim at increasing the system dependability by breaking the fault-error-failure chain.

Modelling of complex system is a problem that can be faced by the modeller with two extreme approaches: the “totally artistic” one, and the “totally automatic” one. By totally artistic we mean that the experience of the modeller is the primary element of the model construction process, upon which we rely for defining a model with the adequate level of abstraction, albeit considering all the system aspects that are relevant with respect to the properties that have to be evaluated. It is only the experience that ensures that the model is consistent with respect to system specification, and that only the “right” abstractions and simplifications have been made.

On the other side lays the field of research that studies the automatic production of performance evaluation and system validation models starting from a high level operational system specification of some sort, for example using UML Sequence Diagrams enriched with performance annotations according to the UML Performance Profile [20]. The weak points of the approach are the need for a formal and complete specification of the system, and the fact that the resulting net models are usually quite rich of (not always useful) details and they may not be solvable with state-of-the-art tools, although, of course, consistency with the system specification is ensured by construction.

The approach proposed here is placed in the “middle” since we try to extract as much information as possible from an abstract and semi-formal description of the system and from its fault tolerance strategies given in the Class Diagram (CD from now on) notation [19], to then complete this preliminary description with operational specifications of system behaviour. The advantage is that this information is available already at the early stages of the application development and that it is quite abstract, so that it allows a first comprehension of the problem and of the validation and evaluation activities requested.

Our work complies with the principles proposed in [23], where a general model-based approach for an early performance validation of software systems (i.e., oriented to validate the responsiveness and the throughput) is defined as consisting of the following steps: 1) capture performance requirements (understand the system functions and rates of operations), 2) understand the structure of the system and develop a model representing its “performance” abstraction, 3) capture the resource requirements and insert them as model parameters, 4) solve the model and compare the results to the requirements, and 5) interpret the predictions to suggest changes if needed.

The approach of this paper is specifically targeted on dependability, it is placed inside a specific application domain, and it addresses only the first three steps
of the five above.

**DepAuDE methodology** This work was developed as part of the methodological effort of the EEC-IST project DepAuDE [11] to support the analyst from the early phases of the project (collection of dependability requirements) down to the definition, validation and dependability evaluation of fault tolerance strategies [5].

The application domain for the methodology is that of cyclic automation control systems, possibly locally or geographically distributed.

Three different formalisms collaborate in a synergistic manner in the methodology: UML Class Diagrams [19], a static paradigm, TRIO [13] logic, and Stochastic Petri nets [17] (PN), an operational paradigm. The methodology follows the approach of integrating different notations during the dependability process as suggested by emerging standard like IEC 60300 [10].

Class Diagrams are the “entry level” in the DepAuDE methodology that provides a set of predefined CD for the automation system domain (called *generic* CD scheme) and guidelines on how to produce from it the *customized* one that refers to the target application (left part of Figure 1). Diagrams are meant as a support for the requirements collection and/or for structuring and/or reviewing for completeness already available requirements.

In the methodology TRIO is used for specifying and analyzing dependability requirements and fault tolerance strategies in a formal logic context.

Stochastic Petri nets [17] (and in particular Generalized Stochastic Petri Net - GSPN [1]) role in the methodology is to support dependability design validation and evaluation through modelling. Their role is the topic of this paper.

The methodology supports the construction of PN evaluation scenarios. A PN scenario consists of a set of PN model components and of their interactions, plus the set of model parameters and the set of performance and validation properties of interest.

Figure 1 summarizes the DepAuDE approach to PN modelling. The methodology helps in the construction of a PN scenario by providing a set of predefined reusable PN models for some of the UML classes, a suggested structure of interaction of the model components, guidelines on how to extract information from Class Diagrams, and in particular on the Class Diagram instantiated on the specific application (lower horizontal arrow on Figure 1), automatic translation from UML StateCharts and Sequence Diagrams into PN, and a suggested approach to the dependability analysis. We would like again to emphasize that the presented approach is intended as a support to a human modeller, whose experience is necessary to complete the PN scenario (right corner of Figure 1).

In this paper we focus on the derivation of information from CDs (Section 2), assuming the reader is familiar with the Class Diagram notation; on the structuring of the PN model and on the definition and reuse of fault PN models (Section 3); and on the completion of the PN structure so as to get an executable PN model (Section 4). A running example will be used to illustrate the concepts. The organization of the analysis is the topic of [4], and the automatic translation of Sequence Diagrams and StateCharts into PN is proposed in [7, 16].

Our approach to the construction of Petri net models shares similarities with other approaches. In [22] there is an example of organization of dependability analysis GSPN model into layers to separate the architecture model, the service model and the failure modes model, although without explicitly modeling the Fault, Error and Failure (FEF) chain [15], and without any explicit use of high level requirement models as we propose.

The European Esprit project HIDE [8] has instead devised an integrated environment supporting dependability analysis of UML-based system design from the early stages, based on the automatic generation of PN from a number of UML diagrams that encode specific dependability aspects in a rather abstract form. The goal being the evaluation from the early stages of the design the resulting model is obviously rather abstract (since a limited amount of information is available), which is an advantage from a computational point of view. This approach has the advantages and disadvantages of the automatic methods discussed above, and can be seen as a complementary approach to the one proposed here.

In this paper we use the term PN component to
indicate a GSPN system [1] (or an SWN one [9]) labeled over transitions, parametric with respect to transition rates (weights) and/or initial marking and with an associated list RESULTS of performance results to be computed and/or verified and a list CONSTR of constraints to be verified.

Running example The automation system considered as a running example is a cyclic application that activates two concurrent processes: each process reads a sample input from a plant, elaborates the future state, saves the new state in memory and produces the new output for the plant. The memory units can be affected by physical faults that may cause errors in the automation functions. Communication units are instead assumed not affected by faults. To increase the dependability of the automation system a fault-tolerance strategy has been devised consisting of error detection, error diagnosis and error recovery. The error detection step uses a standard watchdog mechanism while error diagnosis and recovery steps are implemented by a recovery mechanism. If the watchdog expires, it sends a notification message to a software recovery mechanism, that provides to terminate the watchdog and check the status of the automation system: if no error is present then it is a false alarm, and the watchdog is simply reinitialized. If instead an error is present then a recovery action is carried out. A full description of the example can be found in [3].

2. UML Class Diagrams

This section discusses what type of information can be extracted from the Class Diagrams of the generic CD scheme, and from the CD scheme instantiated on our running example. We shall introduce only a small subset of the CDs of the DepAuDE methodology, hopefully enough to justify our choices; a detailed description of the CDs can be found in [5].

The CDs of the generic CD scheme are grouped into the hierarchical structure of packages represented in Figure 2, where each non-leaf package of the structure encapsulates a set of inner packages together with their definition dependency relationships. The scheme is therefore constituted by a set of CDs that describe the system in terms of automation components, automation functions, dependability attributes, and timing requirements (left branch in Figure 2), a set of CDs that describe the dependability model in terms of the FEF chain (central branch), and a set of CDs devoted to the strategy model (right branch) that is seen as a set of dependability actions/steps, that can be achieved through a number of software “mechanisms”.

The CDs are organized into a set of views (one CD may belong to more than one view), to emphasize different grouping of information: 1) hierarchy views put emphasis on specialization relationship; 2) aggregation views put emphasis on aggregation associations; 3) class definition views put emphasis on class attributes.

Class attributes are used to represent either parameters, whose values have to be provided as input to the specifications, or measures to be computed or upper/lower bounds whose values have to be provided as input to the specification and to be validated at later stages of the development. We have chosen to discriminate these different types of usage of class attributes by prefixing the name of the attributes with a specific symbol (“$” , “/” or “$/”, respectively).

Elements of a generic CD can be customized on a specific application with the help of a set of guidelines [5]. In the customized CD the value of the class attributes and of the association multiplicities have been set and new classes and associations are added.

The customized CD still refers to classes, and not to objects, but certain classes and associations have been made more specific using information from the application. We now illustrate a few Class Diagrams of the generic CD scheme, trimmed so as to simplify explanation, while still, we hope, containing enough information for what will be discussed in later sections.

Figure 3. The CD Structure of Composition.
Figure 4. The generic $FEF$ chain (a) and its instantiation to the running example (b).

Figure 3 describes a portion of the aggregation view of the system: an automation site is a plant (optional) together with one or more automation systems. An automation system is composed by a set of automation functions, that can communicate among them, and by a set of automation component that can be used to perform one of more automation functions. An automation component controls zero or more plant components. An instantiated version of the CD of Figure 3 will contain application specific information, in our running example there are two types of automation components dealing with communication ($AC_{COM}$) and memory ($AC_{MEM}$), three automation functions for asynchronous communication ($ACF_A$), synchronous communication ($ACF_S$), and memory ($AF_{MEM}$), and a single type of automation system $AS$.

Figure 4(a) is a trimming of the CD of the FEF chain. Once customized on a specific application it shows which faults cause which errors, how errors propagate, and which (set of) errors cause a failure, that is to say a deviation from the service delivered by the system. The diagram also connects each type of fault, error and failure with the corresponding system components affected by it. The instantiated version is shown in Figure 4(b): it contains only one type of faults (Physical Faults), a single type of error (Memory Error) and a failure (Halting Failure). The instantiation of the affect relationship relates the fault to the $AC_{MEM}$ component only, the error to function $AF_{MEM}$, and failure to $AS$. Values are set to the input attributes of Physical Fault and Memory Error classes (i.e., the attributes prefixed with the "$\$" symbol), while the attribute $PF$ (prefixed with the "$f$" symbol) emphasizes that the probability of failure of the automation system is a measure of interest to be computed.

The CD description of a system contains a lot of useful information for the construction of PN evaluation scenarios, in particular we have observed the following relationships: (1) the package structure provides indication on the organization of PN component models; (2) the aggregation system view provides information that allows to identify PN components and the composition formulae; (3) binary general associations (associations from now on) among classes indicate interactions and can therefore be used to identify labels for PN model composition; (4) class definition views are rich of attributes that are useful to set rate parameters (input and/or upper-lower bound attributes) and to define the performance indices (output measures and upper-lower bound attribute to be checked); (5) information on the $FEF$ chain is fundamental to set the relationship among the PN models of faults, errors, failures, and system components; (6) hierarchical views can indicate reuse of PN components through inheritance [24]; (7) the $Strategy$ Model package allows to identify which mechanisms are used for which fault, error, or failure (dependability strategy).

3. Composition scheme of PN models

We now exploit the information extracted from the CDs to provide a structure for the system PN model construction, and to identify a number of PN models that are expected to be common to all applications that share the same generic CD scheme. We deal first with the PN model structure, and then consider the predefined PN models.

**PN models structure.** To give structure and organization to the PN model we have followed the PSR layered approach proposed in [12] for modelling in an integrated manner hardware and software systems. In PSR the PN model is organized into three levels: re-
resources, services and processes. Resources are at the
bottom level, and they provide operations for the ser-
vice, where a service is basically a complex pattern
of use of the resources. Services are then requested
by the model placed at the highest level, that of pro-
cesses. Each level is defined through net composition
operators based on transition superposition ("horizon-
tal composition"), the resource level is composed with
the service level, and the resulting net is composed with
the process level also through transition superposition
("vertical composition").

PSR also provide a schema of how the resource and
service nets should look like, while the process level can
be made up of arbitrary nets.

Figure 5, bottom part, depicts a model of a resource
transition labels are written in italics, transition names,
if present, in bold. Let us consider only the non-shaded
portion for the moment. A resource can be idle, and it
can offer one or more operations through the se-
quence of actions start operation, operation, end op-
eration (S_op, op, E_op), either with or without
a lock request. Figure 5, upper and non-shaded part,
models a service. A service can be requested by a pro-
cess through the pair of labels of start and end service
(S_Serv, E_Serv). Once activated the service can re-
quest resource operation via the S_op, E_op labels.

The shaded portion of Figure 5 shows the modifica-
tion to the original PSR resource and service models
to include the interaction with faults and errors: from
each state in which a fault can be perceived a transi-
tion has been added (to be used for synchronization
with the fault model) that takes the component into a
faulty state. Similarly for the other levels.

To use the PSR in the automation system domain
we need to identify the main PN models involved and
their interactions. This identification is driven by the
CD scheme. The organization of PN models into lay-
ers is described by Figure 6. The package structure
with the three branches of Figure 2 is reflected in the
organization into three columns of Figure 6, while to
decide in which level to place the various PN mod-
els we have considered the FEF chain first (Figure 4).
Faults are at the lowest level of the chain, and they
have therefore been placed at the resource level. Con-
sequently also automation components (AC), that are
affected by faults, have been placed at the same level.
With a similar reasoning errors and automation func-
tions (AF) have been placed at the service level, and
failure models and automation system (AS) have been
placed at the process level.

This is depicted in Figure 6 by the set of boxes ACi
for automation components, FTj for the fault models,
AFk for the automation functions, ERh for the error
models, AS for the automation system, and FAILn for
the failure models.

The composition of automation components with
faults requires a proper assignment of labels to inter-
face transitions for horizontal composition, that can be
derived from the associations affect(Fault, Automation
Component) of the CD scheme in Figure 4.

The labels for the composition of automation func-
tions with errors and for error propagation are derived
from the association affect(Error, Automation Func-
tion) and Eeffect(Error, Error), from the CD of Fig-
ure 4.

The inter-level interaction between service and re-
source is given by the association perform(Automation
Component, Automation Function) of the CD of Fig-
ure 3 and by the cause-effect association between faults
and errors of Figure 4.

The labels for the composition of automation sys-
tem with failure are derived from the association af-
fect(Failure, Automation System), while the propaga-
tion from error to failure is based on the association
effect(Error, Failure).

The software mechanisms are instead placed either
at the service or at the process level, depending on
whether they address errors or failures.

Observe that the relationship between software mechanisms and automation functions is through the error models, although there are cases, like the Local Voter, that execute replicas and votes on the outcomes, in which it seems more natural to have a direct relationship between the mechanisms and the functions, but unfortunately, no information of the subject is contained in the CDs, so that no general guidelines can be derived.

**Predefined PN models and their reuse.** A number of models can be predefined and re-used for applications that share the generic CD scheme.

In [4] we have presented a few simple models that allow different type of communication to be all viewed as transition synchronization.

In the DepAnt project a number of GSPN and SWN models of the dependability mechanisms have been constructed and evaluated [2, 4, 6], that can be included in our layered structure with a limited effort.

Another source of reuse is the model of faults. In [3] a hierarchy of classes has been defined according to the classification in [15], and an associated hierarchy of PN models has been created accordingly. Figure 7 shows the first two of the four levels of the hierarchy.

Indeed PN models representing the behavior of super-classes can be used to derive PN models representing the behavior of the sub-classes through refinement and specialization. The refinement of a PN model can be traced back to the inheritance of the static structure of a class (i.e., attributes, operations, associations) and it consists of 1) inheriting parameters (rate/weight, initial marking), results to compute, constraints to verify, and possibly adding new ones; 2) inheriting, and in case modifying, the labels associated to either places or transitions. The specialization, instead, corresponds to the inheritance of the dynamic behavior of the super-class and it consists of either maintaining the same PN model for the sub-class or to modifying it by applying transformation rules that preserve the behavioral inheritance [24]. The basic PN component is an elaboration of the fault generator proposed in [18].

The behavior of Fault super-class of Figure 7 (a) corresponds to the GSPN component model $H_0$ of Figure 7 (b). Transition $ft$ is the occurrence of a fault, transition $ft_{prev}$ is the perception of the fault. The labels of $ft_{prev}$ are used to synchronize with the affected system entity ($AC_i$) and with an error model ($ER_h$). Transition $ft_{end}$ is the fault termination.

Classes Design Faults, Interaction Faults, Malicious Faults, present the same behavior of the more general class Fault, so that the GSPN model $H_0$ is reused to represent these classes also.

The class Physical Faults is associated with the GSPN model $H_1$ that inherits from $H_0$ and adds the fault rate and duration parameters and the
fault dormancy result (i.e., time to perception of a fault). Moreover, the label of transition ft prcv has been assigned according to the inherited association affect (Physical Faults, Automation Component). The net structure of H0 has been maintained but rates of transitions ft occ and ft end have been defined as functions of the added parameters, i.e., w(ft occ) = fault rate and w(ft end) = 1/duration.

Figure 7. Fault classification (a) and GSPN component models of fault classes (b).

4. Getting an executable PN model

Once the basic structure of the PN models has been identified it is necessary to complete them with a number of information related to the specific application. To reach this goal we have identified a number of steps that will be illustrated through an example.

Step 1. Select the concrete classes of the customized CD scheme amenable to a PN model

The set of PN component models can be identified by examining the customized CD scheme to select the classes that are relevant from a quantitative point of view. Good candidates are classes of the customized CD scheme that contain attributes specifying input parameters, metrics to be computed and/or to be verified.

If we assume a class level specification, then all GSPNs are initialized with a single token, while in case of object level specification the identities of the objects come into play. In the context of modeling of distributed object software, an in-depth treatment of the specification level is reported in [25], where a formalism derived from Colored Petri Nets (CPN) [14] has been defined. Having used GSPN for the class models, SWN [9] is a natural choice at the object level to keep track of the object identities.

Example. The customized CD scheme allows the identification of the PN components shown in Fig. 8. At the resource level, left to right, we have: the communication unit, the memory units, and the physical fault models. For AC COM and AC MEM we have reused the resource model of Figure 5 (without and with faults respectively) with transmission and copy as basic operations, while adding a reset transition. For FTP, the predefined physical fault model of Figure 7 has been used (with reset).

At service level, left to right, there are: the synchronous communication function used by the automation system to initialize the watchdog, the asynchronous communication function used by the automation system to send signals to the watchdog, the automation functions f, the memory errors affecting the automation functions f, and the recovery mechanism. Models ACF AND ACF S are simplified models of communication presented in [4]. AF MEM follows the basic skeleton of service model of Figure 5 with “reset” transitions. ER MEM is a predefined error memory model taken from [3]. Model REC has been produced from the high level design specification of the recovery mechanism.

At process level there are: the automation system, the halting failure mode of the automation system, and the watchdog mechanism. Model AS has been produced from the high level design specification of the automation system. Model FAIL is a predefined failure mode model taken from [3]. WD is a simplification of the watchdog model resulting from the automatic translation of the StateChart specification of the watchdog [7].

Colours have been used to keep track of the multiple copies of AC MEM, FTP, and ER MEM, as well as to model two parallel subprocesses of AS.

Step 2. Customize the composition rules using the associations of the customized CDs.

The names/roenames of the binary general associations defined in the CDs have been used to characterize the labels of the GSPN components in Figure 8.
Figure 8. Example of analyzable PN model: the set of PN components
If object level specification is assumed, and therefore SWN models are used, further “control” SWN component models may be necessary, to do the right association between colors, as, for example, in the case of AC and AF model, to associate to each Automation Component the corresponding Automation Function.

Example. The associations named affect allow to define three synchronization labels: ftmem, to synchronize the memory model and the faults, erra, to synchronize the automation functions and the memory, and finally, fail, to synchronize the system and the halting failure model. New labels are introduced for the interaction between the recovery mechanism and the memory error model (detect, noerr) and between the interaction mechanism and the automation functions model (reca). We can then define the sets of labels for the horizontal compositions of the resource level (L_{res} = \{ftmem\}), of the service level (L_{svc} = \{erra\}), and of the process level (L_{pr} = \{fail\}).

Another similar procedure allows to identify the labels for the vertical composition of layers based on the associations perform between automation components and automation functions, and on the associations effect between physical faults and memory errors, and between memory errors and halting failure. New labels are also added to represent: the interactions among the recovery mechanism model and the component models laying at resource level and at process level, the interactions among the automation system model and the automation (communication) functions models and, finally, the interactions among the watchdog model and the automation communication functions.

The final PN model PSR is then obtained using the parallel composition of the various levels upon the identified labels, according to the PSR methodology.

Step 3. Define the initial marking of the composed PN.

A complete definition of the initial marking is possible only when system design specification is available, by composing the initial marking of the components, that may require information on the object identities.

Example. The initial marking is based on the assumptions that there is one communication unit, and that physical faults affects only one the first memory. The marking parameter M_0 has one token per color of the class C = C_1 ∪ C_2 = \{c_1\} ∪ \{c_2\} and it is used for ACMEM and AFMEM. The marking parameter M_1, defined as the single color c_1, is used for FTPH, ERMEM and FAIL.

Step 4. Initialize the rate/weight parameters of the PN composed model and define the results.

The rate/weight parameters and performance/dependability indices should be defined according to the values set to the input and output attributes of the customized classes. The remaining ones are added and initialized by the modeller. Example. From the customized CD of Figure 4(b) we can identify two input parameters for the PN model: fault_rate and duration representing the rate of the fault occurrence and its duration, respectively. For all other rates and weights no indication is provided by the CDs.

The metrics to be evaluated are specified by three attributes: availability, defined in the Automation Function customized class; PF, defined in the Halting Failure customized class, specifying the probability of failure; and false_alarm, defined in the Watchdog customized class, that suggests the computation of the frequency of false alarms. These information extracted from the CDs are only indications, no formal definition is associated to them, and they have to be defined by the modeller.

Perform the analysis.

To perform the analysis it may not be a straightforward task, since it may require a modification of the PN model. This topic is out of the scope of the paper, we would only like to point out that for certain types of indices it may be necessary to perform a transient analysis in which the states representing a failure are made absorbing, while if instead a recovery strategy is being evaluated it is likely that the model should be made ergodic.

Example. We have used GreatSPN tool [21] to construct the PN component models depicted in Figure 8 and the program algebra [6] to carry out their composition. The reachability graph of the final SWN model contains 115 tangible markings, 778 vanishing markings and 4 dead markings (failure of the automation system). The modified ergodic model is characterized by 103 tangible markings and 1051 vanishing markings.

5. Conclusions

This paper advocates the use of the high level information already available in the requirement phase, to set up Petri net scenarios, that is to say the identification of the model components, of their interactions and of the performance and dependability indices of interest. In our approach we assume that this high level information is conveyed by a system description based on Class Diagrams.

The approach has been developed inside the EEC project DepAuDE, that has as reference domain that of dependable automation systems. Having a fixed domain allowed us to propose a specific organization of the net models, and to reuse predefined models, much along the lines of the re-usability concepts supported.
by Class Diagrams hierarchies.

A number of important issues are indeed still open, among them we would like to recall the exploitation of resource information based on UML Deployment Diagrams, and a more systematic approach to the use of information from the object diagrams and to the relationship between models and analysis settings.

The running example of this paper is still a toy example, and indeed an extensive application of the method proposed here to a number of case studies is planned for the future.

References


