Methods for the Dependability Evaluation of Critical Adaptive Systems

Research Report

István Majzik

Budapest University of Technology and Economics

2018.

Contents

1 Introduction and Overview 3
2 The General Approach 3
3 Elements of the Solution 5
3.1 Architecture Modelling 5
3.2 Graph Patterns 6
3.3 Stochastic Models 6
3.4 Mapping of Static Architectures to GSPN Models 7
3.5 Mission Automaton 8
3.6 Analysis of the PMS 9
4 Evaluation 10
5 Conclusions 10
6 References 11

1 Introduction and Overview

In this research report our approach is summarized that supports the evaluation of adaptive systems used in dependability-critical application areas.

Model-driven engineering methodologies are often used for the architectural design of critical systems. Architecture models (given in general purpose or domain specific modelling languages) allow the modular and systematic construction of formal analysis models for the evaluation of dependability and performability.

The adaptation (on-line reconfiguration and fault handling actions) in the system introduce changes in the architecture and parameters, and thus result in a multi-phased operation. Hence analysis models must be constructed from the accordingly modified architectural models for each phase of system operation.

In this work an approach is proposed for deriving a stochastic Phased Mission System (PMS) analysis model directly from an architectural model instance and the description of its changes. To capture the changes, a so-called Mission Automaton model is proposed, which is an abstract state machine formalism that supports the definition of reconfigurations, fault handling, and parameter changes depending on the status of operation.

In the derived PMS analysis model, system operation during a single phase is modelled with a Generalized Stochastic Petri Net, which is assembled from fragments according to the architectural model. Phase transitions specified by the mission automaton modify a run-time version of the architectural model according to the changes, which also causes an update of the related analysis model. Thus stochastic models compatible with PMS analysis tools are obtained. This way these models can be solved using existing external tools to compute system level dependability measures like reliability, availability and safety.

2 The General Approach

The architecture design of critical embedded systems is often supported by model-driven engineering techniques. The architecture that is captured in a formal or semi-formal architecture design language (like UML, SysML, AADL etc.) offers the possibility for an early analysis of the extra-functional properties of the design. Namely, the architecture model extended with the local parameters of the components can be mapped to an analysis model that is used to compute system level measures.

In case of dependability (reliability, availability, safety) analysis this generic approach is instantiated in the following steps:

1. Extending the model with the local dependability parameters of the components and links composing the architecture. The most important local parameters of components are the failure rate, repair rate, and error detection latency.

The local parameters of links are the error propagation probability and the error propagation latency. In case of redundant subsystems, the logic of the redundancy is also attached to the subsystem model (e.g., in a form that describes the conditions for leading to a subsystem failure from component failures).

2. The extended model is mapped to an analysis model, which is (in this case) the so-called dependability model that captures the fault occurrence processes and the error propagation, thus the conditions of the occurrence of a system failure. The elements of the dependability model are nodes that represent the basic components that may fail (abstracting from the functional role and related parameters of the architecture components) and edges that represent error propagations.
3. The structure of the dependability model is mapped to a stochastic analysis model, which is typically a Continuous Time Markov Chain (CTMC) or a Stochastic Petri Net (SPN) or a Generalized Stochastic Petri Net (GSPN). This stochastic model is attached a reward that represents the reliability or availability measure of the system.
4. The stochastic analysis model is solved to compute the reward. Typical tools to solve a stochastic model are, for example, the Functional Safety Suite (solving CTMC models), the PetriDotNet tool (solving SPN models), or the Möbius tool (solving Stochastic Reward Nets that extend SPN).

Such analyses can be supported by an automated derivation of the stochastic analysis model from the architecture model by model transformations [1], [2], [3].

In this work, an approach is proposed to support the dependability analysis of adaptive systems [4] [5]. The approach consists of the following contributions:

- A technique to capture dynamic adaptation, reconfigurations, fault handling and parameter changes on the level of the architecture model. To do this, we define a mission automaton formalism by extending Graph Transformation Abstract State Machines (GT+ASM) [6] with stochastic and timing properties. GT+ASM leverages graph pattern matching, which is the technique applied on the architecture model for the description of reconfigurations and fault handling.
- A technique to map the architecture model together with the mission automaton to an analysis model. Namely, the changes described by the mission automaton result in multiple, non-overlapping phases of operation, that can be captured by stochastic Phased Mission Systems (PMS) [7]. In a PMS model, each phase (as a non-changing “snapshot” of the architecture) is represented by an SPN model, and these are properly connected to represent the changes described by the steps of the mission automaton. This way a PMS analysis

model is derived from the architecture model and its evolution described by the mission automaton.

- Solution of the PMS model is provided by external tools.

Regarding the roles involved in the design flow, the following ones can be mentioned:

- Domain modelling engineers use the architecture modelling language to capture the architecture, and then define the graph patterns used for specifying changes, and construct the mission automaton.
- Reliability engineers extend the architecture model with local dependability parameters that characterize stochastic behaviours, such as failures and repairs. Moreover, they specify the analysis model transformation in terms of Stochastic Petri Nets, which are amenable for automated, modular construction. To do this, existing transformations like [1] can be leveraged. The reliability engineer may also refine the mission automaton if needed.
- Domain modelling engineers use automated tools to process the extended architecture model and the mission automaton to derive PMS analysis models and compute the system level measures. If these measures do not satisfy the requirements then the architecture and/or the reconfiguration and adaptation policy (captured by the mission automaton) shall be changed.

3 Elements of the Solution

In this section the main elements of the approach described in Section 2 are summarized. The detailed description can be found in our papers [4] and [5].

3.1 Architecture Modelling

In model-driven engineering, graph based languages, including UML, SysML and AADL [14], are used to capture architectures. Metamodels explicitly describe the abstract syntax of these modelling languages, including the classes, references and attributes that comprise the language. An architecture model is an instance model of the architecture modelling language.

The Eclipse Modeling Framework (EMF) [8] is a de-facto standard metamodelling technology, which we used in the implementation part of our work.

In Figure 1, an example architecture model is presented [5].

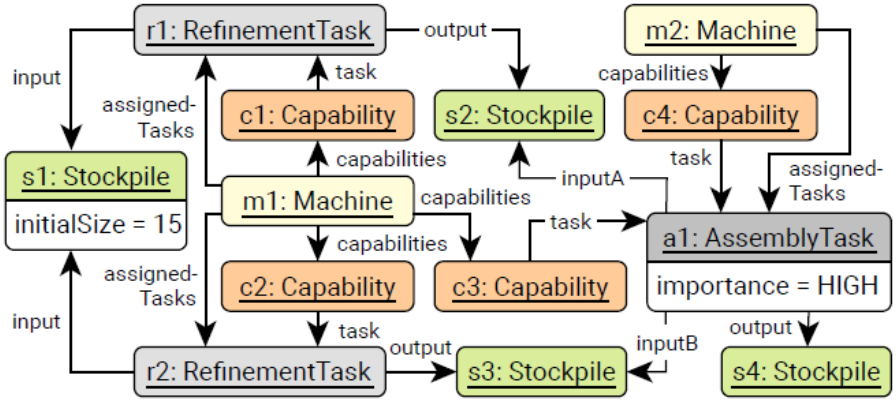

Figure 1: An example architecture model

3.2 Graph Patterns

State-of-the-art modelling tool-chains often rely on model queries to retrieve fragments of interest from a model, to specify model to model and model to text transformations, as well as to validate well-formedness constraints on models [9].

A graph pattern is a graph-like structure that represents a condition matched against an instance model. It can prescribe structural and attribute constraints, as well as negative application conditions on the pattern match. Parameter variables match distinguished objects inside a pattern. In Figure 4 a graph pattern is presented.

3.3 Stochastic Models

In our approach we refer to the following stochastic models:

- Generalized Stochastic Petri Net (GSPN) is a commonly used formalism for the dependability evaluation of asynchronous systems. Formally, a GSPN is a directed bipartite graph with a set of places and transitions [10]. A marking assigns token counts to the places. Starting from the initial marking, if enough tokens are available at its input arcs, and no transition with higher priority is fireable, a transition may be fired to remove tokens from its input places and put tokens to its output places.
- A continuous-time Markov chain (CTMC) represents the stochastic behaviours of the GSPN. Timed transitions are fired when an exponentially distributed delay with a given rate parameter has elapsed, while immediate transitions are fired immediately according to their priority and probability weight when they become enabled.
- Phased Mission Systems (PMS) are characterized by consecutive phases of operation caused by changes in system configuration or environment [11]. Modelling and analysis of PMSs are made more complex than single phased systems by the history of the system, such as degradation of the components, affecting subsequently occurring phases. In state-based stochastic PMS mod-

els, each phase is described by a lower level model like a GSPN. The upper level model determines the length of each phase and the possible phase transitions. In order to propagate the history of the system, phase transitions map states of the lower level model associated with the source phase to the target phase.

3.4 Mapping of Static Architectures to GSPN Models

First let us consider the construction of analysis models for failure processes of static (unchanging) architectures, which is the so-called static analysis model transformation. Creation of the dynamic analysis model that incorporates reconfigurations of the architecture model calls the static transformation as a subroutine.

In case of architecture models, the typical approach is a modular transformation where patterns from the architecture model are systematically mapped (based on the types of elements) to interconnected model fragments in the analysis model. This process is facilitated by modular and compositional extensions to Petri nets [12]. For example, the modular Petri nets formalism [13] allows the assembly of large models by instantiating and connecting net fragments.

Model transformations tools, such as [14], [15], [16], construct target (right-side) models according to matches of precondition patterns in the source (left-side) models. The left side of a single transformation rule is a precondition graph pattern. The right side is a template for target model objects, in our case, a Generalized Stochastic Petri Net fragment [13]. For each match argument tuple, its right side is instantiated by adding a copy of it to the target model. Traceability information relates the source and the target instance models. The horizontal trace hyper-edges connect the objects of the match argument tuples on the left to the target model objects on the right.

In Figure 2, Generalized Stochastic Petri Net analysis model for the architecture in Figure 1 is presented [5].

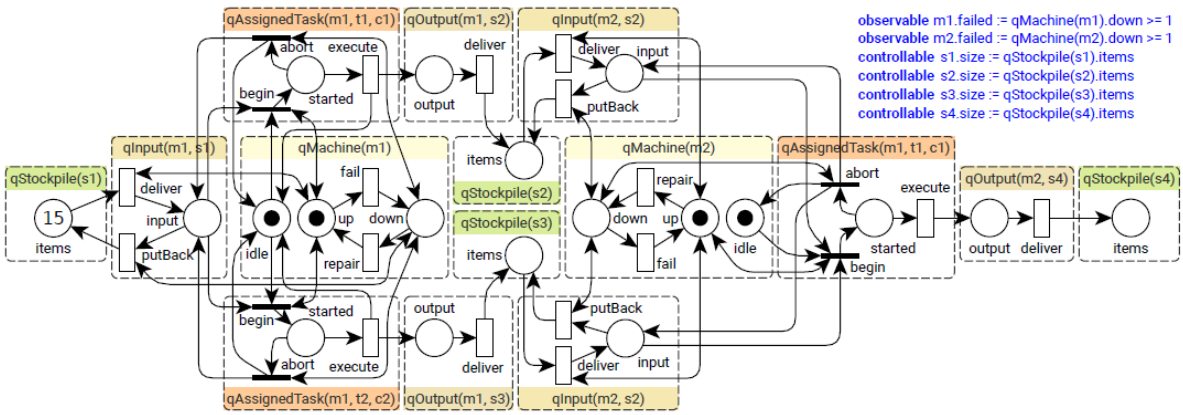

Figure 2: GSPN analysis model for the architecture in Figure 1

3.5 Mission Automaton

Informally, the mission automaton is an abstract state machine (ASM) that refers to graph transformation rules (GT) as a mathematically precise description of model changes. As a novel contribution, we defined a stochastic and timed variant of GT+ASM for reconfigurations of adaptive systems.

The mission automaton runs along the Generalized Stochastic Petri Net analysis model. Transitions in the automaton may be triggered by changes of runtime attributes in the analysis model or by the elapsing time. Actions attached to transitions may reconfigure the architecture model, as well as update a set of global variables.

In the mission automaton, interactions between the static elements and run-time attributes of the architecture model are avoided by forbidding access to the run-time attributes in actions that modify the static architecture elements. Instead, a specific run-time attribute update action is offered with limited control flow. Therefore, the parts of mission automaton that depend solely on the static architecture model are separated from those that also depend on run-time attributes, and hence the GSPN marking. Thus the state space of the mission automaton can be over-approximated without exploring the state spaces of the derived GSPN. State space and probability distribution handling is delegated to a PMS analysis tool.

Formally, a mission automaton is a 5-tuple *(L, l_0, F, G, T)*, where *L* is the set of locations, l_0 is the initial location, *F* is the set of final locations, *G* is the set of global variables, and *T* is the set of transitions. A transition is equipped with a trigger, a guard condition (a k-parameter precondition pattern), a list of parameters, and a list of actions. The parameters are global or local variables. Variables are bound to objects of the architecture model.

As an example, in Figure 3 a mission automaton model and in Figure 4 a related graph pattern is presented [5]. The mission automaton is responsible for reconfiguring the architecture given in Figure 1. Transitions are denoted as *trigger / [guard] / actions* where the guard is omitted if it is the trivial graph pattern (which has 0 parameters and holds always). The concrete syntax is detailed in [4].

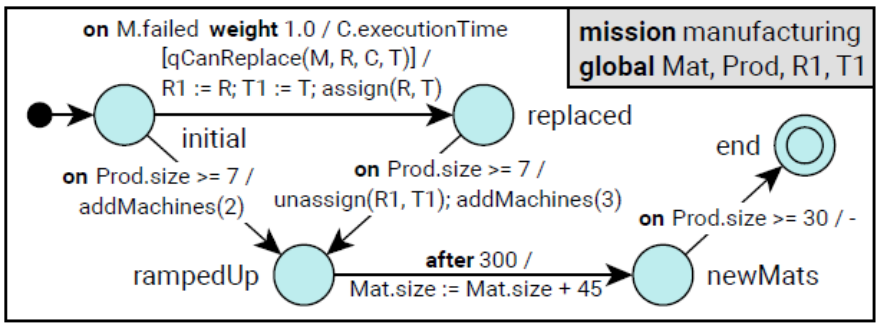

Figure 3: A mission automaton

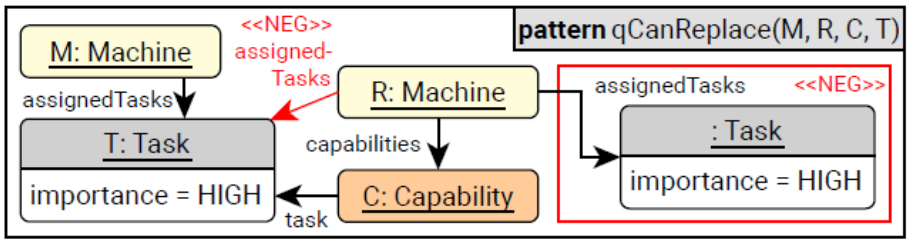

Figure 4: Graph pattern qCanReplace referenced in the mission automaton

3.6 Analysis of the PMS

Analysis of architecture models and mission automata is performed in two steps.

- Firstly, the mission automaton is unfolded by taking into account the potential instantiations of its transitions, as well as the modifications of the architecture model instance. The unfolded mission automaton and architecture model configurations form a tree.

  The (indirectly) marking dependent behaviour of the mission automaton is over-approximated by ignoring triggers in the automaton. Each mission automaton transition is considered fireable, regardless of the reachable markings of the Generalized Stochastic Petri Net analysis model.

  Dependence on the static and run-time parts of the architecture model is erased from the triggers and actions by substituting architectural concepts with their GSPN representations in the analysis model.

- Secondly, the tree of architecture configurations is turned into a PMS for analysis. The upper level model is the unfolded mission automaton, where transitions between phases of operation are governed by the trigger expressions. The lower level models are the GSPN sub-models that describe the behaviour of the system during a phase. As the mission completes successfully upon reaching a final location, the corresponding phases are marked in the upper level model.

In Figure 5, phases of the PMS model generated using the mission automaton in Figure 3 is presented. Each phase refers to a GSPN analysis model (PN_0 … PN_7).

The hierarchical methodology proposed in [7], [17] can be adapted for the PMS analysis. We direct the reader to our report [5] for more details.

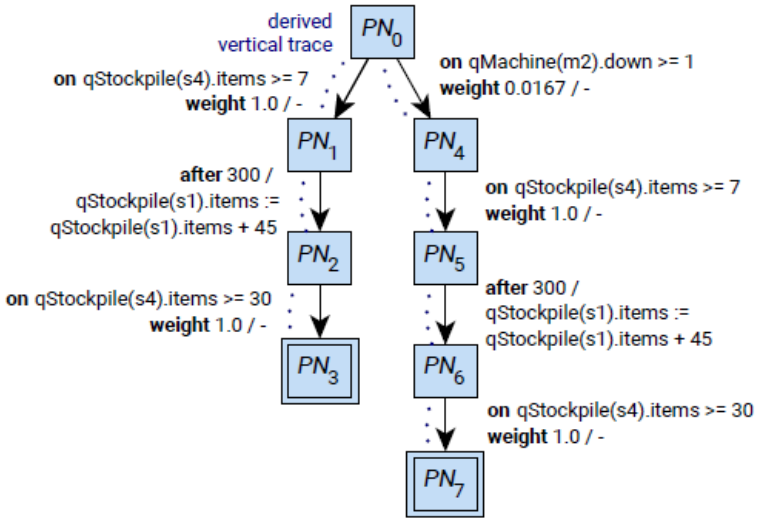

Figure 5: Phases of a PMS model

4 Evaluation

We evaluated the scalability of the analysis model construction approach in the context of incremental analysis model transformations [4]. The mission automaton was unfolded using scaled versions of the architecture model, such that the architecture contained multiple copies (1 to 16) of the machine components.

The number of elements in the GSPN models obtained, as well as the median running times (running time of the transformation of the initial phase, including the execution of the static transformation in batch mode, and also the total running time, and the average incremental execution time for the non-initial phases) were measured.

It turned out that the sizes of the analysis models for a single phase grew linearly as machines were added to the architecture, while the number of phases in the upper level model grew in a quadratic way, along with the execution time of the full PMS construction. The unfolding could take advantage of incremental execution of the static transformation. Thus no more than 20 ms per non-initial analysis model was taken. Total execution time remained below 20 seconds.

5 Conclusions

This report aimed at the description of an approach to the evaluation of critical adaptive systems.

We presented (1) mission automaton formalism for specifying reconfigurations and fault handling in system architectures and (2) mapping from these to stochastic PMS analysis models. According to our empirical evaluation, good scalability can be provided by our incremental analysis model construction.

Possible extensions include support for more advanced transformation chains that could increase applicability in complex multi-paradigm modelling scenarios.

6 References

[1] A. Bondavalli, I. Majzik, and I. Mura, “Automatic dependability analysis for supporting design decisions in UML,” in Proc. 4th IEEE Int. Symp. High-Assur. Syst. Eng. IEEE, 1999.

[2] H. Koziolek, “Performance evaluation of component-based software systems: A survey,” Perform. Eval., vol. 67, no. 8, pp. 634–658, 2010.

[3] S. Bernardi, J. Merseguer, and D. C. Petriu, “Dependability modelling and analysis of software systems specified with UML,” ACM Comput. Surv., vol. 45, no. 1, 2012.

[4] K. Marussy and I. Majzik, “Constructing Dependability Analysis Models of Reconfigurable Production Systems”, In Proc. 14th IEEE International Conference on Automation Science and Engineering (CASE 2018), Münich, Germany, August 21-25, 2018.

[5] K. Marussy and I. Majzik, “Constructing phased-mission systems for dependability analysis of reconfigurable production systems,” Tech. Rep., 2018. [Online]. Available: http://doi.org/10.5281/zenodo.1290661, 2018.

[6] D. Varró and A. Balogh, “The model transformation language of the VIATRA2 framework,” Sci. Comput. Program., vol. 68, no. 3, 2007.

[7] I. Mura and A. Bondavalli, “Hierarchical modeling and evaluation of phased-mission systems,” IEEE Trans. Rel., vol. 48, no. 4, pp. 360-368, 1999.

[8] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, EMF: Eclipse Modeling Framework, 2nd ed. Addison-Wesley Prof., 2009.

[9] Z. Ujhelyi, G. Bergmann, Á. Hegedüs, Á. Horváth, B. Izsó, I. Ráth, Z. Szatmári, and D. Varró, “EMF-INCQUERY: An integrated development environment for live model queries,” Sci. Comput. Program., vol. 98, no. 1, pp. 80–99, Feb. 2015.

[10] M. A. Marsan, G. Conte, and G. Balbo, “A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems,” ACM Trans. Comput. Syst., vol. 2, no. 2, 1984.

[11] A. K. Somani, J. A. Ritcey, and S. H. L. Au, “Computationally efficient phased-mission reliability analysis for systems with variable configurations,” IEEE Trans. Rel., vol. 41, no. 4, pp. 504–511, 1992.

[12] A. Marechal and D. Buchs, “Generalizing the compositions of Petri nets modules,” Fundam. Inform., vol. 137, no. 1, pp. 87–116, 2015.

[13] E. Kindler and L. Petrucci, “Towards a standard for modular Petri nets: A formalisation,” in PETRI NETS 2009, ser. LNCS, vol. 5606. Springer, 2009, pp. 43–62.

[14] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, “ATL: A model transformation tool,” Sci. Comput. Program., vol. 72, no. 1–2, pp. 31–39, 2008.

[15] G. Bergmann, I. Dávid, Á. Hegedüs, Á. Horváth, I. Ráth, Z. Ujhelyi, and D. Varró, “VIATRA 3: A reactive model transformation platform,” in ICMT 2015, ser. LNCS, vol. 9152. Springer, 2015, pp. 101–110.

[16] MOF Query/View/Transformation Specification, Object Management Group Std., Rev. 1.3.

[17] I. Mura and A. Bondavalli, “Markov regenerative stochastic Petri nets to model and evaluate phased mission systems dependability,” IEEE Trans. Comput., vol. 50, no. 12, pp. 1337–1351, 2001.