Software & Systems Modeling
https://doi.org/10.1007/s10270-018-0663-8

REGULAR PAPER

Towards a model-driven engineering approach for the assessment of non-functional properties using multi-formalism

Simona Bernardi[1] · Stefano Marrone[2] · José Merseguer[1] · Roberto Nardone[3] · Valeria Vittorini[3]

Received: 5 February 2016 / Revised: 27 November 2017 / Accepted: 21 December 2017

Abstract
Model-driven techniques can be used to automatically produce formal models from different views of a system realised by using several modelling languages and notations. Specifications are transformed into formal models so facilitating the analysis of complex system for design, validation or verification purposes. However, no single formalism suits for representing all system's views. In particular, the assessment of non-functional properties often requires integrated modelling approaches. The ultimate goal of the research work described in this paper is to develop a comprehensive, theoretical and practical framework able to support the development and the integration of new or existing model-driven approaches for the automatic generation of multi-formalism models. This paper defines the core theoretical ideas on which the framework is based and demonstrates their concrete applicability to the development of a multi-formalism approach for performability assessment.

Keywords Multi-formalism · UML profile · Performability · Model-driven engineering · Generalised Stochastic Petri Nets · Repairable fault trees

1 Introduction

In the last two decades, many researchers and practitioners have been involved in defining and developing model-driven approaches oriented to the quantitative analysis of software and systems. Also, several model transformation chains have been developed, from UML [41], or profiled UML, and from domain-specific modelling languages (DSMLs), in order to create analysable models, i.e. models for which solution techniques and tools are available.

On the other hand, multi-formalism refers to the usage of heterogeneous submodels. It enables different modelling languages to describe different system views and/or subsystems. Then, multi-formalism tends to reduce the complexity and effort of the analysis needed in case the system would be approached as a whole. This notwithstanding, multi-formalism has not received the same attention as single formal modelling, especially in the industrial community, maybe due to its greater modelling complexity and error-proneness. While a big effort has been spent on the "model–transform–analyse" chain involving single formalisms, few approaches have been defined to deal with multi-formalism models. For example, in [8], MARTE [40] and DAM [11] are used to allow evaluations of performance and dependability, but in a separate manner.

The aim of the research work herein described is to address methods and techniques for defining model-driven processes that can be applied to the generation and analysis of multi-formalism models. In particular, the focus is on the modelling and evaluation of quantitative system

Communicated by Dr Gabor Karsai.

✉ Stefano Marrone
stefano.marrone@unicampania.it

Simona Bernardi
simonab@unizar.es

José Merseguer
jmerse@unizar.es

Roberto Nardone
roberto.nardone@unina.it

Valeria Vittorini
valeria.vittorini@unina.it

[1] Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Zaragoza, Spain

[2] Dipartimento di Matematica e Fisica, Universitá della Campania "Luigi Vanvitelli", Naples, Italy

[3] Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Universitá di Napoli Federico II, Naples, Italy

Published online: 16 February 2018

properties, both primitive—performance, maintainability or reliability—and derived—performability or survivability.

Due to the many issues already open in the multi-formalism modelling and analysis research, the aim of solving these problems in a single research paper is not realistic. Once we have as a long-term aim the construction of a theoretical and practical framework for generating multi-formalism models by means of model-driven techniques. The choice of the proper languages is made according to the overall non-functional property (NFP) to evaluate; here, we address the performability NFP. The main contribution of this paper is twofold: first, the overall approach in its general terms is defined and then its application to the performability case is presented, seeing it as a composition of the traditional performance and reliability methods.

As for the application of the proposed approach to performability, we propose a methodology for obtaining multi-formalism models of complex systems starting from high-level performance and reliability models, expressed as UML profiled models. Indeed, we are investigating the issue of defining an approach to bridge DSMLs (including UML profiles) with formal languages and their technical spaces: this is the ultimate goal of our ongoing work. An objective of the present paper is then to provide a proof of concept by realising this goal in a restricted application field (performability) and under proper assumptions.

A practical aim of our approach is to reuse existing transformation chains, combining them for enabling the generation of multi-formalism models. In this regard, our performability methodology leverages two existing chains: MARTE to Generalised Stochastic Petri Nets (GSPNs) described in [25] and DAM to repairable fault trees (RFTs) described in [9,17].

Finally, we raise that little attention has been paid to the problem of automatically derive performability models expressed in different formal languages without relying on an intermediate language. The paper also deals with this specific aspect, which is a very challenging issue due to the inherent complexity of the problem. So, this is a further original contribution of our work with respect to the current state of the art.

The paper is organised as follows. Section 2 contains the background of our research and a discussion about related work. Section 3 introduces the main concepts and the definition of the general approach for model-driven multi-formalism modelling and evaluation of NFPs. Section 4 focuses on models integration. We explain how the general approach is instantiated to performability in Sect. 5, and this section also contains a description of the flexible manufacturing system robotic cell used as a running example in the rest of the paper. Section 6 provides the guidelines of the methodology for performance and reliability modelling and presents the supporting model transformations. Section 7 describes how the performance and reliability models are integrated to obtain the final analysis model and the solution process. Finally, Sect. 8 contains some closing remarks and hints about future work.

2 Background and related work

2.1 The multi-paradigm modelling domain

Different terms are used in the literature to describe approaches which deal with the heterogeneity of models, such as multi-paradigm modelling, multi-formalism modelling, multi-view modelling, multi-modelling or multi-language modelling, to name a few.[1] A unified vision of this research field has been proposed by Hardebolle and Boulanger [26] under an extended definition of the Multi-Paradigm Modelling domain, formerly introduced by Mosterman and Vangheluwe [36].

In the past years, a wide research effort has been devoted by the scientific community to investigate and propose solutions to many difficulties which arise at different levels when trying to combine heterogeneous models. A non-exhaustive list of research issues in this field includes the semantic differences among the modelling languages, the abstraction and refinement relationship existing between models progressively refined during a development process, the consistency and the correct integration of models used to analyse different aspects or views of a same system, the development of supporting methodologies and tools able to guarantee the consistency of the models used in different stages of the development cycle for different purposes.

Several multi-paradigm approaches and techniques have been proposed to cope with these issues, a classification of research areas and approaches is provided in [26,45].

Some of the available approaches are specific for a restricted set of modelling languages, such as the approaches driven by the necessity of modelling hybrid systems which have lead to well-known industrial verification tools (e.g. Simulink/Stateflow[2]) as well as non-proprietary frameworks (e.g. Modelica[3]).

Others solutions support an open set of languages by basing on metamodelling as a key technique to express the abstract syntax of DSMLs in the context of model-driven engineering (MDE) and allow to easily build DSMLs and tools (e.g. AToM3 [18], MetaEdit+ [49], EMF [47], GME [29]) or exploit the concept of heterogeneous models

[1] The models considered in this paper are oriented to the representation of discrete event systems.

[2] http://it.mathworks.com/.

[3] http://www.modelica.org/.

of computation (e.g. Ptolemy II project [15,21], Metropolis [44], ModHel'X [13]).

The techniques used to address the heterogeneity of models span from the definition of a unifying semantics (e.g. oriented towards the automation of hardware/software co-design, as in the Metropolis, or to the modelling and analysis of discrete event systems, as in the Möbius approach [19]), to the composition of models and the joint usage of several modelling and analysis tools (e.g. oriented towards simulation as in Ptolemy II and ModHel'X, or oriented towards the analysis of non-functional properties as in the SIMTHESys multi-formalism modelling framework [5] and OsMoSys [35,52]), to the composition of modelling languages, including techniques for merging metamodels [22,46] and translating models (e.g. in AToM3).

2.2 Automatic generation of performability models

Although several theoretical and practical results have been obtained and a number of tools supporting multi-paradigm modelling are available, a further issue to be investigated is the *automation in the generation of heterogeneous models*, given that the theoretical problems arising from their heterogeneity have been addressed and solved (at least for the set of the modelling languages involved). This is especially important when dealing with *formal models*, which are expressed in languages that have a formal syntax and a formal semantics. Formal models are widely advocated for analysing functional and non-functional properties of systems and they are necessary in the validation and verification process of critical systems, such as transportation, avionics, automotive, health care, etc.

A recent trend in critical system modelling for dependability and performability analysis sees the development of model-driven approaches that may automatically deriving quantitative models relying on DSMLs or on UML [41]. Model-driven processes are very appealing as they enable the automatic translation of models and analysis of different solutions during the overall system development life cycle and they can be easily integrated in industrial settings. The vast majority of these works are focused on the software application domain and use UML together with its profiling extension mechanisms, as source modelling language. Although there are also contributions in the systems engineering and in the software/hardware real-time embedded system domain, considering mainly SysML [42] and AADL [43] as DSML. The surveys [4] and [10] reveal that the *DSML-to-formal model transformation* is nowadays a well-settled approach to verification and validation of performance and dependability properties.

Focusing on performability assessment, several model-driven approaches have been proposed [12,14,27,51,54,55]. In [14], Stochastic Activity Networks are obtained from UML-based specification to evaluate the performability of mobile software systems. Trowitzsch et al. [51] propose an automatic derivation of Stochastic Petri Nets from UML state machines and a software tool as a support. In [12], a Deterministic and Stochastic Petri Net model is obtained from UML models (use cases, deployment, state machine and sequence diagrams). Khan et al. [27] consider instead Stochastic Reward Net as target modelling formalism. In order to include the quantitative system parameters necessary for performability evaluation, all the aforementioned works exploit the profiling mechanisms of UML. The works [14, 51,54] use the UML profile for Schedulability, Performance, and Time (SPT) [38], while the works introduced in [12, 27] rely on the complementary use of two UML profiles: MARTE [40]—for the performance annotations—and QoS-FT [39] (in [27]) or the DAM UML profile in [11]—for the dependability-related annotations. The Palladio approach [7] introduces its own DSML to specify component-based software architectures and provides a model-driven tool chain to perform performance predictions. The KlaperSuite [16] exploits the pivot language Klaper as an intermediate modelling language in order to separate (source) design models from (target) quality-related models and facilitate the transformation among them.

In conclusion, multi-paradigm and multi-formalism approaches do not explicitly address the problem to automate the construction of the resulting analysis models, while approaches leveraging MDE techniques propose a unique modelling language as target formalism of the transformation. An attempt to generate different formal models—i.e. fault trees, GSPNs and queueing networks—from the same input UML-based specification to evaluate, respectively, the reliability, availability and performance of an e-health system is proposed in [8]. However, in [8] the formal models are still analysed separately and no interaction between the models is considered. Frameworks exist supporting the automatic generation of multi-formalism modelling tools (e.g. AToM3) or the automatic solution of multi-formalism analysis models (e.g. Möbius) but not to automatically build heterogeneous models. We propose to exploit MDE techniques to define a framework able to aid in the automated generation of multi-formalism models. In doing that our work has some similarities with approaches to construct bridges between different notations, as introduced [56] or between DSMLs and UML profiles [1].

3 Overview of the approach in the MDE context

The proposed approach is oriented to the modelling and the analysis of quantitative NFPs of critical systems. Figure 1 gives an overview of the approach.

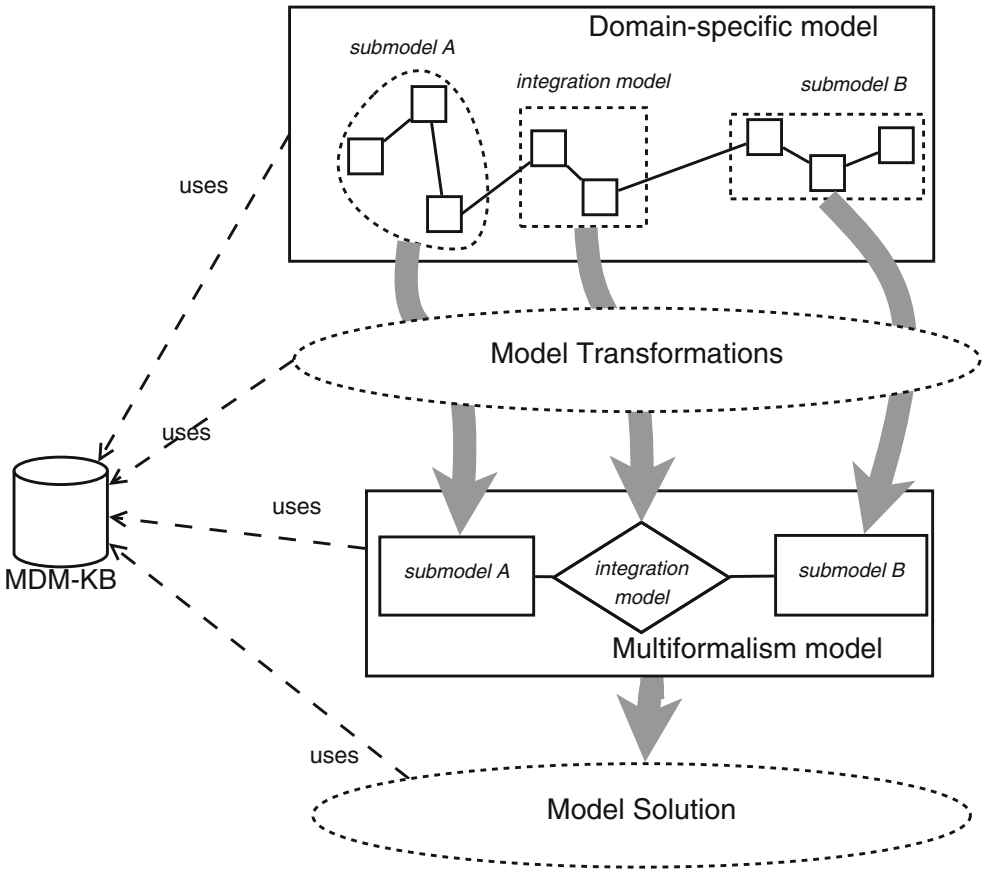

Fig. 1 Overview of the general approach

The modeller is in charge of creating a model of the system: such model should be able to describe meaningful subsystems in a compositional manner. The model can be built on the base of defined/existing DSMLs and according to the specific NFP to analyse. These submodels can be different diagrams of the same model, specific portions of the same diagram or different models linked together or, more generically, different models. According to the NFP to evaluate, the submodels are translated by model transformation chains into formal models: different submodels can be managed by different chains, generating formal models expressed in different formalisms. A key role is played by the integration model that is a high-level model that connects the different high-level submodels: one of the pillars of the entire approach is the application of transformation chains on integration model able to generate formal models still connecting the different formal submodels. Once multi-formalism models have been generated, they can be analysed by instantiating a solution process able to orchestrate different solvers, to compose formal models and to process model parameters/results.

To tame the huge methodological, linguistic and technological space of possibilities, the proposed approach needs the existence of a decision support system including: (1) a repository of techniques (model-driven multi-formalism knowledge base, MDM-KB—Fig. 1 on the left) and (2) a set of (semi-automated) guidelines that specify the modelling and analysis workflows to be carried out by the engineers.

The complete definition of *MDM-KB* is a long-term objective: it requires a comprehensive collection of all the model-driven experiences for NFPs. As a knowledge base, it is defined upon a schema and an instance. This paper proposes a tentative domain model of MDM-KB (Sect. 3.1) and gives an instantiation on this schema with performability in Sect. 5 and followings.

3.1 The model-driven multi-formalism knowledge base

Building such a unifying framework offers several degrees of freedom for the modeller; he/she might choose among different combinations of: high-level languages, high-level model structures, specific metrics to evaluate for the NFP, model transformations, target formalisms, multi-formalism integration techniques and multi-formalism model solution processes. The *MDM-KB* defines, among all the possible

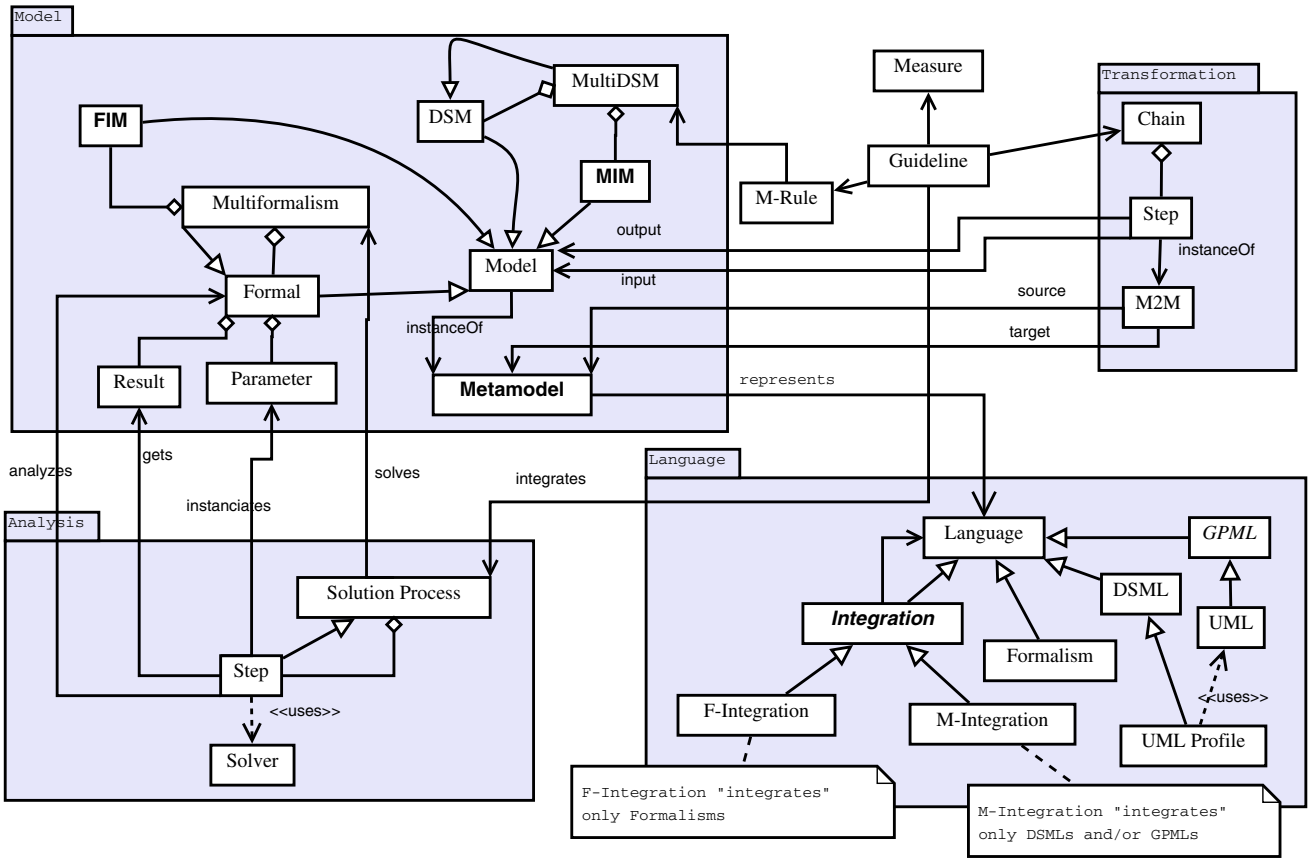

Fig. 2 Structure of MDM-KB

choices, proper modelling and analysis guidelines to make the approach feasible, technically sound and automatable. Figure 2 represents a tentative schema of MDM-KB that sketches the main concepts of the approach. The structure of MDM-KB has four main packages: *language*, *model*, *transformation* and *analysis*.

The main concept is *guideline* which embraces all the main phases of the approach by:

- detecting which are the most proper *measure*s able to evaluate desired NFPs;
- suggesting a modelling approach in terms of languages to use and of ways to structure high-level models (*M-Rules*);
- determining a transformation *chain* able to generate a multi-formalism model on the base of the high-level one;
- defining the most suitable *solution processes* for analysing the multi-formalism model.

An M-Rule is the part of the guidelines addressing modelling concerns: it contains the rules about the language to use (i.e. if one wants to model concurrent distributed systems, UML fits in this scope) as well as the recommendations about the use of such languages (i.e. in the previous example, it is suggested to use a UML deployment diagram, different UML state machine diagrams for the inner evolution of the components and a UML sequence diagram for the message exchange).

The *language* package incorporates the homonymous concept of *Language* and its derivatives comprehending *DSML* and *GPML*[4] (also with their specialisations *UML profile* and *UML*, respectively), as well as *formalism*. A *formalism* is a language used as a target of a model transformation (i.e. a non-user-centred language, suitable to be analysed). A special kind of *language* is *Integration*, representing languages able to connect together different languages. Two specialisations of *integration* are present: *M-integration* which integrates only DSMLs and GPMLs, and *F-integration* which integrates only Formalisms.

In the *model* package, the main concepts are *model* and *metamodel*. A metamodel represents a language and a model an instance of a metamodel and it may be a *DSM* model (compliant with a DSML/GPML) or a *formal* model (compliant with a formalism). Specialisations of model are used for the integration models both at high and low levels:

[4] General-purpose modelling language.

– a *multi-DSM* is a model that is composed of different *DSM*s and one or more *MIM*s (multi-DSM integration model);
– a *multi-formalism* is a model that is composed of different *formal* models and one or more *FIM*s (formalism integration model).

MIMs and FIMs are the integration models (respectively, at domain-specific level and formalism level) compliant, respectively, with M-integration and F-integration languages. Formal models may have input and output parameters (respectively, *Parameter*s and *Result*s).

The *transformation* package collects three concepts: *M2M* (model-to-model transformation) which has source and target metamodels, a transformation *step*, which instantiates an M2M by defining one or more output models generated from input ones, and a transformation *chain* which is a sequence of steps.

The *analysis* package is in charge of defining methods for the evaluation of the *measure*s by the solution of the multi-formalism model. The main concept is the *solution process* that can be composed of several *step*s. A step is in charge of analysing a formal model by instantiating input model parameters, getting analysis results supported by existing/ad hoc solvers.

One of the main aims of the entire approach is to foster and exploit the reuse of existing model-driven artefacts: mainly metamodels and transformation chains; in fact, the scientific literature has produced several and assessed "single formalism approaches" able to evaluate NFPs with a simple model–transform–analyse approach. By means of the modelling methodology and guidelines sketched here, the MDM-KB may be constructed reusing assessed approaches and may generalise the traditional model-driven approach.

4 On the integration model

As stated in Sect. 2.1, the scientific literature addressed the problem of composing homogeneous and heterogeneous formal models and results [37], but some considerations and discussions are due on the issues that arise from the integration steps, mainly during the modelling, transformation and analysis steps. Some generic examples are presented to clarify some points.

Modelling integration in the high-level modelling activities mainly deals with integration between data elements and between model elements.

The first is related to the integration of the variables and their data between the models; more in particular, it is the problem of relating together data present in one submodel (parameters and results) to data of another submodel (parameters). The aim is to define proper procedures (*compositional operators*) which transform data. Input data of the operators can be model parameters or a solution results while output data are generically constituted by model parameters. These functions span from the simple copy (the identity function) to complex data manipulation. An example of this kind of integration is constituted by the case of a decision algorithm which estimates the probability $P(\mathcal{E})$ of an event $\mathcal{E}$ by integrating the evaluations of three different classifiers `A`, `B` and `C` (possibly built on different technologies such as fuzzy logic, neural networks and Bayesian networks). In this situation, a multi-formal model can be constituted by a function operating on $P_A(\mathcal{E})$, $P_B(\mathcal{E})$ and $P_C(\mathcal{E})$ as example by calculating the average of the three probabilities:

$$P_V(\mathcal{E}) = \frac{P_A(\mathcal{E}) + P_B(\mathcal{E}) + P_C(\mathcal{E})}{3} \tag{1}$$

The second aspect of the model integration is the connection of model elements (graphical or textual) which are more related to the behaviour or the structure of the modelled system. Examples span from the synchronisations of different activities between workflows models (expressed as example with UML Activity diagrams, Petri Nets, etc.) to connecting with edges submodels ports (e.g. between an AADL Process port and a UML Component port). Since the integration model is a model itself, we need to specify it into a defined modelling language. This problem of choosing the proper language for the M-integration can be mainly solved in two different ways:

– *ad hoc neutral language* (with respect to submodel languages): while the advantage is to have a single M-integration metamodel to define and to interoperate with, the disadvantage of this solution is the necessity to implement new model transformations from and to this ad hoc language;
– *shared language* (with respect to submodel languages): M-integration language is one of the languages of the connected models. The advantage is the capability to reach a higher level of integration among connected models and multi-DSM since they share the same language. In this way, existing transformation chains may be reused as they are or by extending them. The problem here is constituted by a hardening of the interoperability issue between the languages.

Transformation once the integration model is built up, a proper set of model transformation chains has to be applied to generate formal models. As stated above, the choice of the M-integration language may require additional effort in defining proper model transformations: in particular the

problem addressed here is how to cope with the generation of *n* formal models from *m* high-level models. Two decoupling approaches are detected:

- *intermediate language* by defining an intermediate language, a complex model *m-to-n* model transformation is decoupled into two transformations. An *m-to-1* M2M is defined to generate a model conformant in an ad hoc intermediate language, and then, this model is transformed in *n* different formalism by a *1-to-n* M2M according to the type of analysis to perform. This situation happens, at a different levels of abstraction in the Möbius [19], Klaper [16] and CHESS approaches [34]. While the definition of a single intermediate formalism may ease the construction of a brand new framework, this approach is not able to reuse legacy transformations;
- *no intermediate language* an existing M2M can be reused to generate the *i*th formal model from the jth high-level model as if the multi-formal high-level model were not present. In these cases, the existing transformations are to be extended to cope with the connected models. Few works are present in the scientific literature using this approach; an example is in [31] where two UML profiles and related transformation chains are used to model physical and cyber protection systems: in a joint application of them, the two model transformations are extended to handle concepts of the other profile.

Analysis from the analysis point of view, we explicitly address the problem of reusing analysis algorithms and solvers by orchestrating them into a solution process. As solutions of these models are not totally independent from each other, there are two possibilities according to the dependency relationships of the model solutions (i.e. the dependency of a model solution from the results of another one):

- *hierarchical structure* the model solutions are organised into a hierarchy with no cycle. Here it is possible to find an order of invocation of the solvers. An example is represented by the RAMS (reliability availability maintainability and safety) evaluation of the ERTMS/ETCS (the interoperable European railway signalling system) where system-level failure is decomposed into subsystem and component levels (reliability of trains, availability of central controllers, performance failures of communication networks) and then modelled by different formalisms [24]. Another example is in the power system domain [28] where authors use a hierarchical submodel structure and the analysis of the "upper level" model is possible after solving "lower level" ones;
- *cyclic structure* the presence of cycles in the dependency relationships among solvers does not allow for a linear solution process but rather than asks for iterative process. The problem of convergence methods and times as well as the problem of the initial condition is open and can be reported to the fixed-point theory. A concrete example of this situation is in [6] where two formalisms are used to model a sensor network: Stochastic Activity Networks for modelling the node and the Markovian Agents for modelling the network. Since the solution of the lifetime of each node depends on the network layout (inter-distance among nodes) as well as the network depends on the evolution of each node, a fixed-point solution process is needed. Another example is in [23] where the author represents a computer controlled water tank by means of bond graph and VDM models: in this case the solution process is based on co-simulation and the two simulators (20-Sim and VDMTools) are orchestrated and executed together.

5 A methodology for performability assessment

The paper has presented in Sect. 3 the framework of the approach and the MDM-KB that supports it, while Sect. 4 discussed important aspects on model integration. The rest of the paper is devoted to prove the feasibility of our framework. To this end, we develop a methodology targeted to the performability assessment. In fact, the framework provides the concepts and guidelines needed to develop methodologies for NFP assessment using multi-formalisms. Our methodology can be applied to different choices of modelling languages, transformation chains, formal languages and solution processes. Concretely, this section presents the choices we did to present and develop the methodology and the case study where we will apply it. However, as illustrated by the examples in Sect. 4, the methodology can be applied to other choices.

The case study is taken from the flexible manufacturing system (FMS) domain. In Fig. 3 a simple FMS production cell is depicted. Machines transform materials and they can be moved only by armed robots that transport parts from/to machines and other places inside the cell. Semi-finished parts need to be further worked by machines in the following stage, and then, they can be temporarily stored in buffers.[5] In the figure, we consider two segments: the first one refers to Machine1 and Machine2 served by two faulty robots, the second one is constituted by a single Machine3 and a single non-faulty robot. The main difference between robot and faulty robots is that while the former is assumed to be unbreakable and the latter may fail according to the failure rates of their components. Each robot is made of a robotic arm and a double-redundant control unit. Each control unit

[5] Without loss of generality, we suppose an unlimited buffer.

Fig. 3 An FMS production cell

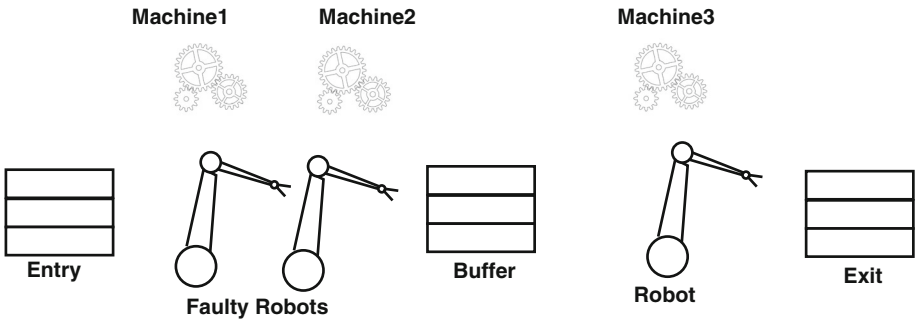

Table 1 FMS parameters

Description	Value (mean)
MTTF of a robot arm	10^5 h
MTTF of a robot microcontroller	1.35×10^4 h
MTTF of a robot IO card	1.35×10^4 h
MTTF of a robot power unit	5.5×10^3 h
Time to finish a piece for the Machine 1	0.75 h
Time to finish a piece for the Machine 2	1 h
Time to finish a piece for the Machine 3	0.5 h
Period to check the availability of a machine	0.05 h

consists of a power unit, a microcontroller and an I/O card. When one of these components fails, the entire control unit fails. The redundancy of the control units allows having repair facilities (technicians or spare parts) able to restore its functionality. We assume that a failed robot is non-repairable. Finally, Table 1 summarises the parameters of the case study, all them were inferred from existing data sheets and similar case studies. We assume that the time required by the robots to move materials is negligible compared to that required by the machines to process them.

The methodology is developed in Sects. 6 and 7 according to Fig. 4. In particular, it applies our general framework to the performability case, intended as the performance of a system under faulty conditions.

Firstly, Sect. 6 provides modelling guidelines aimed at obtaining performance and reliability models:

- The specification of the `Performance View` and `Reliability View` (cf., Fig. 4) is addressed through UML profiling. In particular, by MARTE [40] and DAM [11] profiles, respectively. In the MDM-KB, this corresponds to `Language::UMLProfile`.[6]
- As transformation chains for the `Performance View` and `Reliability View`, we rely on previous works. Concretely on [25] for the performance transformation and in [9] for the reliability transformation. It is worth observing that the two model transformation techniques have been proposed in the literature, so they are not an original contribution of this work. In the MDM-KB these concepts belong to `Transformation::Chain`, while in Fig. 4 they are represented by two grey arrows.
- As formal languages, `Language::Formalism` in the MDM-KB and `performance model` and `reliability model` in Fig. 4, we propose GSPN [32] for performance and RFT [17] for reliability.

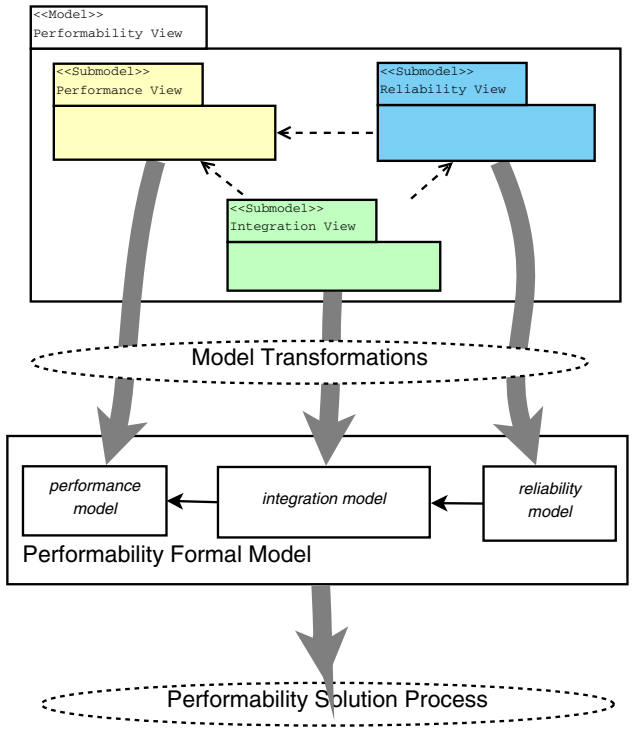

Fig. 4 Approach applied to performability assessment

Secondly, Sect. 7 addresses the rest of the methodology. Section 7.1 defines the `Integration View` (cf., Fig. 4), `Language::Integration` in the MDM-KB. Section 7.2 describes the model transformation from the `Integration View` to the `integration model`.

[6] We reference the elements of the MDM-KB prefixing the name of the package and using roman fonts.

Section 7.3 describes the `Performability Solution Process`, which in the MDM-KB is represented by `Analysis::SolutionProcess`.

6 Performance and reliability modelling guidelines

6.1 Performance modelling

Most of the performance modelling approaches, which address the analysis of software systems specified with UML, rely upon the UML-MARTE profile [40]. They incorporate performance-related parameters in the original UML models and propose model transformation methods to get formal models. The latter can be analysed using techniques which are specific of the target formalism (e.g. in case of GSPN, typically state-based or simulation techniques). In general, two types of modelling views of the system are required for performance analysis purposes:

- structural that can be represented by UML class, component or deployment diagrams, and
- behavioural that are typically represented by a (set of) sequence or activity diagrams.

The structural view defines the system resources, whereas the behavioural one describes a (set of) system process(es) or execution scenario(s).

In the running example, we have used a component diagram (Fig. 5) to represent the high-level view of the system resources and an activity diagram (Fig. 6) to represent the workflow process of the flexible manufacturing system. In particular, the actions in the activity diagram (drawn as rounded rectangles) represent the steps of the process and the decision/merge nodes (both drawn as diamond symbols) model conditional flows.

6.1.1 Performance view

Profiling is the mechanism that UML provides to enhance a system design with specifications beyond the structural and behavioural views. Concretely, a *Profile* is a set of stereotypes and tags that introduce in UML the concepts of a specific domain. Therefore, a profile converts UML into a DSML for such domain. In particular, the MARTE profile enables to enhance the original UML-based specification of the system with performance-related parameters.

A key feature of MARTE is the framework for the specification of non-functional properties (NFP) and the value specification language (VSL). The former allows the modeller to define several properties, such as the *source*—i.e. whether the NFP is a requirement or a measure to be predicted—or the type of *statistical measure* associated with the NFP (e.g. a mean). The VSL enables the specification of variables and complex expressions according to a well-defined syntax (see examples of the *value* field and the *in$* and *out$* variables in the note symbols of Fig. 6).

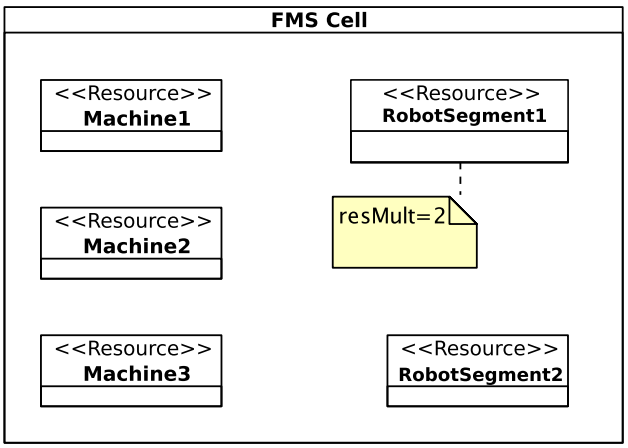

Fig. 5 Component diagram of the system resources

We have developed the system performance view by applying the MARTE extensions to the UML basic specification. In particular, the key concepts in performance modelling are those of **steps**, **resources** and **workload**. MARTE defines an appropriate set of stereotypes and tags for introducing these concepts in the UML diagrams, as follows.

A workflow process represents a set of **steps** that are ordered according to a predecessor–successor relationship. When a workflow step consumes time, it is modelled as an action of the activity diagram stereotyped with «`gaStep`», and its `execTime` tag is used to quantify such consumption (cf., the last four values of Table 1 and the `execTime` tagged values of the `gaStep` actions in Fig. 6).

Resources represent run-time entities that offer services needed to carry out the workflow steps. In the component diagram, resources are identified by the «`resource`» stereotype and when more than one resource of a given type is needed then the `resMult` tagged value indicates it (see the note symbol in Fig. 5). During workflow execution, resources need to be acquired, in an orderly way, for performing activities; for example, a robot is needed for moving a piece. In the same way, resources need to be released when the activity no longer needs them, so they could be used by another activity, which can be blocked while waiting for the resource availability. In the activity diagram (e.g. Fig. 6), the «`gaAcqStep`» and «`gaRelStep`» stereotypes are attached to the transitions that precede and follow the action(s) which need the resource to be executed. Moreover, the `usedResources` tag indicates the name of the resources, while the `priority` tag indicates the priority of this step for getting such resource.

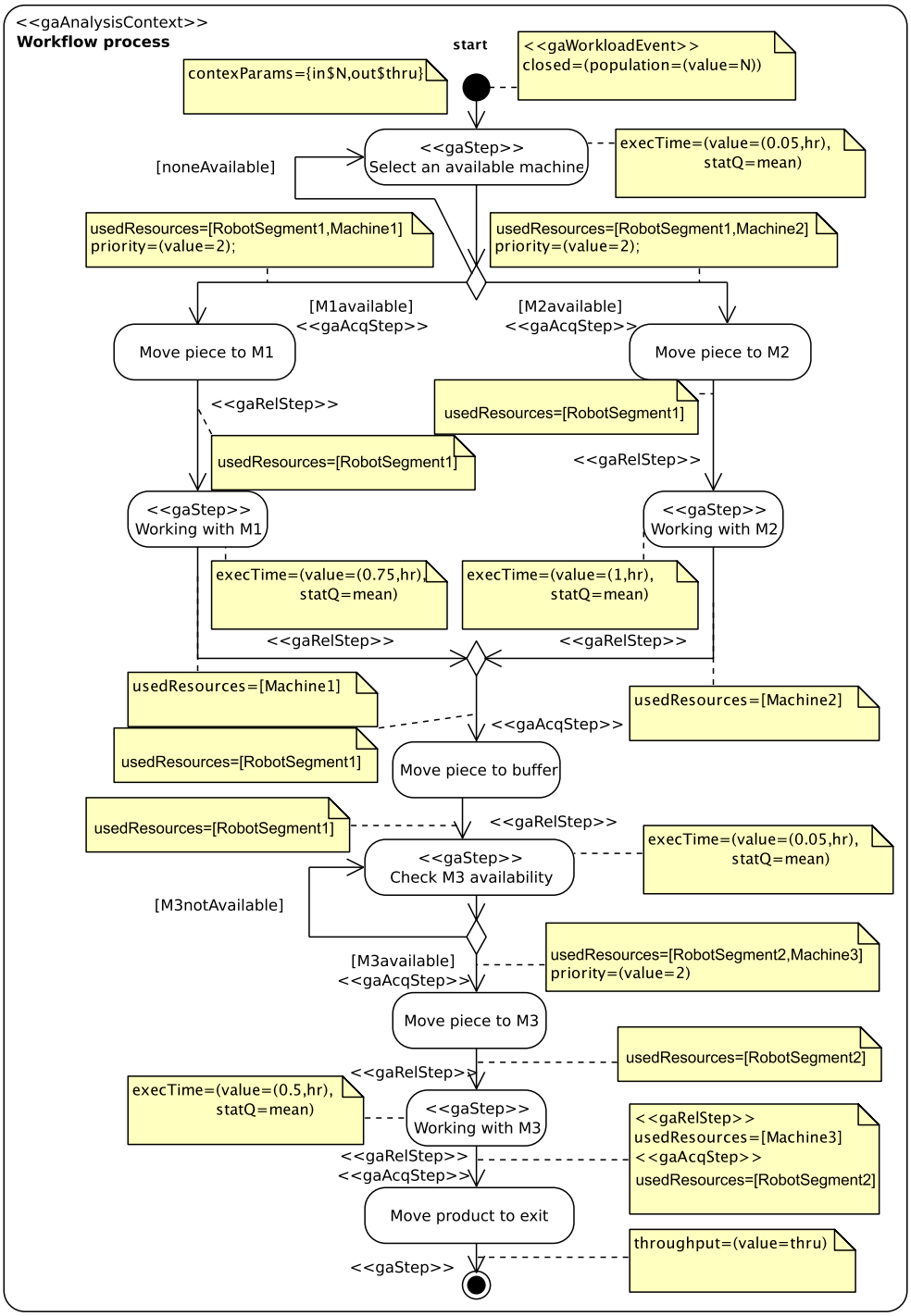

Fig. 6 Activity diagram of the FMS workflow process

The workflow process is executed by an applied load intensity, that is either open or closed. An **open workload** has a stream of requests that arrive at a given rate in some predetermined pattern (such as Poisson arrivals), whereas a **closed workload** has a fixed number of active or potential users or jobs that cycle executing the workflow and, possibly, spending an external delay period (sometimes called a “think time") outside the system, between the end of one response and the next request. The workload is specified by applying the «gaWorkloadEvent» stereotype to the initial node of the activity diagram. In the running example, the workload is closed and it represents the number of raw material and unfinished parts that are loaded into the cell.

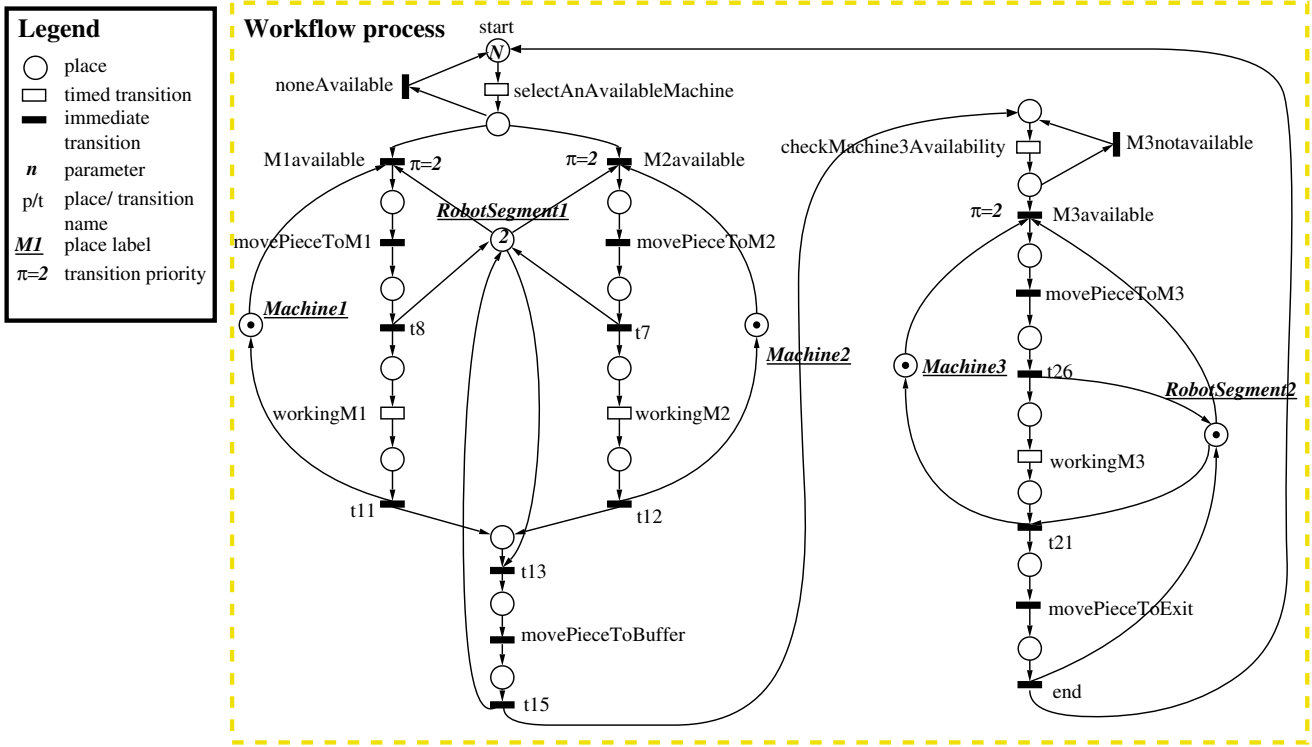

Fig. 7 Performance GSPN model

Finally, all the input and output variables used in the tagged values need to be declared: to this aim the UML diagrams, where such variables are used, are stereotyped with «gaAnalysisContext» and the associated tag contextParams lists the variables, each one characterised by either the *in$* or *out$* property depending whether it represents an input or an output parameter. In the FMS example, there is an input variable (i.e. N), that represents the initial workload, and an output variable (i.e. $thru$), that represents the throughput of the system. The latter is assigned to the throughput tagged value associated with the last step of the FMS workflow.

Observe that, in Figs. 5, 6, 8, 9, and 12, the UML note symbol has been used to show explicitly the MARTE extensions, i.e. the tagged values associated with a given stereotyped model element. However, when an UML tool with MARTE profiling facilities is used (e.g. Eclipse—Papyrus [48]), the stereotypes and tagged values can be easily set using proper GUI.

6.1.2 Transformation to a performance model

The target performance model is a GSPN model and can be obtained by reusing the approach [25], where the same MARTE extensions used in [25] are applied to activity and component diagrams instead of deployment and sequence diagrams. In the following, an informal description of the model transformation is provided considering the running example. The UML-MARTE diagrams of Figs. 5 and 6 are the input of the transformation, whereas the resulting GSPN model is shown in Fig. 7.

Mapping of the system resources The elements of the component diagram considered in the transformation are the «resource» components (Fig. 5, e.g. *Machine1, RobotSegment1*). For each resource component, a GSPN place is associated (Fig. 7, the labelled places *Machine1, RobotSegment1*). The resMult tagged value is used to set the initial marking of the place. When no tagged values are provided, a multiplicity one is assumed, then one token is set as initial marking.

Mapping of the workflow process The translation of activity diagram to a GSPN model is quite straightforward, since in UML2.x the former was restructured to have Petri net-like semantics [41]. Therefore, herein we focus on the mapping of the MARTE extensions.

The «gaWorkloadEvent» stereotype is used to specify the workload. In particular, the closed workload population is mapped to the initial marking of the GSPN place representing the initial node of the activity diagram (cf. the tagged value assigned to the *population*—in Fig. 6—and the initial marking of place *start*—in Fig. 7).

The «gaStep» timed actions are translated to timed transitions of the GSPN model; the execTime tagged value is used to set the transition rate, i.e. the inverse of the value. The «gaAcqStep» («gaRelStep») transitions of the activity diagram represent the acquisition (release) of the resources, where the resources to be acquired (released) are specified by the usedResources tag. Such transitions of the activity diagram are then translated to immediate transitions of the GSPN, that include in their input (output) set the places representing the resources. The priority of the immediate transitions of the GSPN is set to the priority tagged value annotated to the mapped transitions of the activity diagram. By default, the priority is one and the increasing order criterion is assumed for priority assignment.

Mapping of variables The «gaAnalysisContext» stereotype is used to declare the variables used in the diagram; the input variables result in either place marking or transition rate parameters in the GSPN model: e.g. in Fig. 6, the variable *N* is mapped to a place marking parameter. The output variables are translated to GSPN output parameters: e.g. the variable *thru* is mapped to the throughput parameter associated with the GSPN transition *end*.

6.2 Reliability modelling

The reliability analysis can be conducted by annotating the system with the DAM profile [12] and by applying a set of model transformations that are able to transform the annotated UML model into an analysable formal model. In detail, here we refer to the transformation already introduced in [9] that generates an RFT from an UML annotated model. We will apply this transformation to the running example. Obviously, it is possible to apply a different transformation in order to generate a formal model in a different target formalism. As for performance analysis, also the reliability analysis requires two types of modelling views of the system:

- structural, which can be represented by UML class, component or deployment diagrams, and
- behavioural, which are typically represented by a single or a set of state machine diagrams.

The structural view defines the system resources as for performance analysis, whereas the behavioural one describes the failing and the repairing policies related to the system components.

In the running example, we have used a component diagram (Fig. 8) to represent the component-based structure of the faulty robot (i.e. the *RobotSegment1*) and a state machine diagram (Fig. 9) to represent the failing and the repairing states of its control unit. In particular, the states in the state machine diagram (drawn as rounded rectangles) represent the steps between a completely up and running state to its maintenance after a failure.

6.2.1 Reliability view

The features related to reliability aspects of the modelled system can be annotated on the UML model by applying the DAM profile. In fact, DAM annotations are needed on the system structural view and also in detailing the repairing policy for the control unit component of the faulty robot. The faulty robot has been modelled through the component diagram depicted in Fig. 8, while the repairing policy is represented through an annotated state machine, shown in Fig. 9. The DAM profile has been applied, in order to specify through stereotypes and tags the reliability parameters and measures in the UML model, as described in the following.

Once the resources that can fail have been identified in the system (i.e. the robots of the first segment in this case), we need to model their internal structure and basic components as well as parameters related to reliability aspects. According to the DAM definition, the stereotype «daComponent» is used to annotate each entity of the faulty components that can be affected by a thread and interacts with other entities (hardware and/or software) and with the physical world. A component can be made up of other interacting components. We applied this stereotype both on the considered faulty component (i.e. *RobotSegment1*) and on its subcomponents (i.e. *arm*, *control unit*, *microcontroller*, *power* and *IO card*). Among the different tags related to the stereotype «daComponent», we underline the usage of the resMult tag to indicate the number of available instances of a component or subcomponent, and the fault and repair tags to specify, respectively, the intrinsic mean time to failure (MTTF) and mean time to repair (MTTR) of the component. Figure 8 shows the annotation of the reliability parameters, reported in Table 1, on the component diagram of the *RobotSegment1*. Observe that the MTTR associated with the control units is a variable, i.e. $cuMTTR$.

Redundancy is the typical mean to add fault tolerance capabilities in a system, the stereotype «daRedundantStructure» is used to annotate the set of components that should not be a single point of failure, and the related tag ftLevel is used to specify the minimum number of components needed to guarantee the service. In the running example, the subpackage *ControlGroup* has been tagged as a «daRedundantStructure», in which at least one *ControlUnit* is necessary to guarantee the service.

The metrics of interest for the reliability analysis must be also specified in the reliability view with variables. In the example, the metric to be estimated with the reliability formal model is the MTTF of the *RobotSegment1* and it is specified by assigning the variable *X* to failure.MTTF

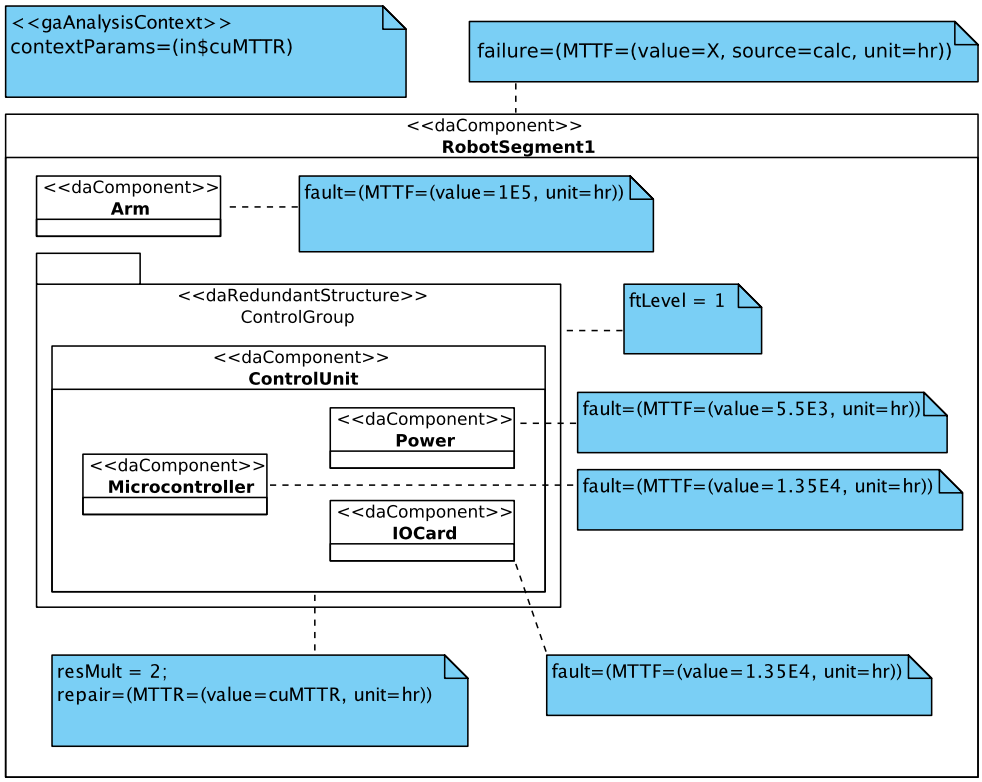

Fig. 8 Component diagram of the faulty robot (internal structure)

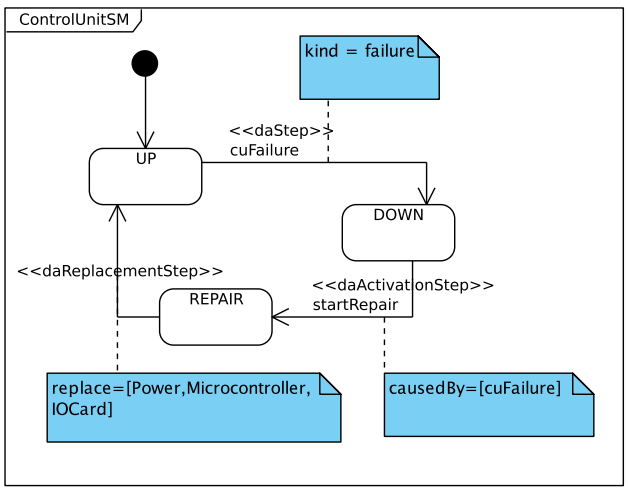

Fig. 9 State machine diagram of the control unit

tag, associated with the faulty robot component, in the UML component diagram.

The repairing policies associated with (sub)components can be modelled by state machine diagrams; see in Fig. 9 the repairing policy of a single control unit. Each policy accounts for: (a) the possible states of the subcomponent—e.g. *up*, *down* and *repair*—and (b) the transitions leading to such states. In Fig. 9, the transition to the *down* state is triggered by the failure causing the activation of the maintenance activities. This transition has been stereotyped by «daStep» with the tagged value `kind` equal to `failure`. The subsequent transition *startRepair*, which leads the subcomponent in the *repair* state, is stereotyped by «daActivationStep», while the last one is a «daReplacementStep» that models the replacement of the *power*, *microcontroller* and *IO card* components to terminate the maintenance.

At last, as in the performance modelling, the diagram is stereotyped by «gaAnalysisContext» in order to declare the variables used in the diagram. Observe that the variable X is also used; however, it has not been declared in the diagram (cf., Fig. 8). Indeed X is a global variable, which is also used in the integration view, and therefore, it will be declared at a higher-level modelling view. We will come back to this issue in the next section.

6.2.2 Transformation to a reliability model

The reliability view enables to generate a repairable fault tree (RFT) model. In order to accomplish this objective, there is a full reuse of the model transformation described in [9]. According to this approach, from the reliability view of the running example—i.e. the component diagram of the *RobotSegment1* (Fig. 8) and the state machine diagram modelling

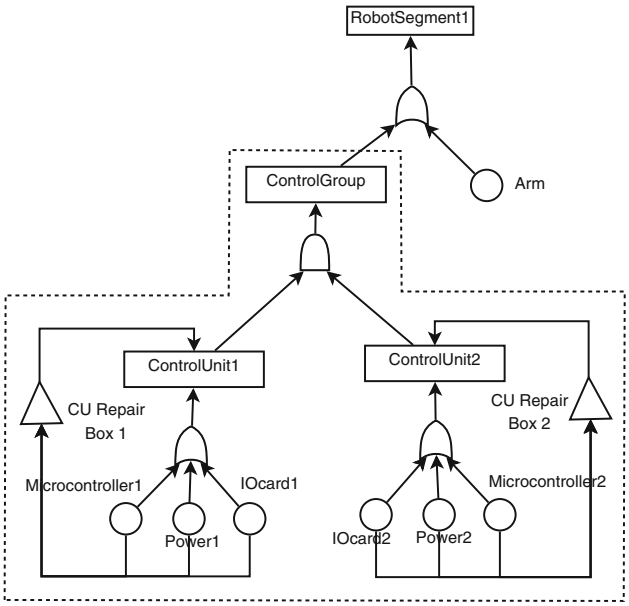

Fig. 10 Reliability RFT model

the repairing policy of the *control unit* (Fig. 9)—the RFT depicted in Fig. 10 has been derived.

Herein, we describe how the transformation approach works with respect to the considered case study; for further information, refer to [9]. Basic concepts of the RFT formalism are provided instead in "Appendix B."

Mapping of the robot structure The elements of the component diagram considered in the transformation are those stereotyped by «daComponent» (e.g. *RobotSegment1, ControlUnit* in Fig. 8) and by «daRedundant Structure» (i.e. *ControlGroup*). Each «daComponent» is translated to a node of the tree that can be either a basic event—i.e. a leaf—or a middle event—i.e. an intermediate node or the root. In particular, it is mapped (1) to as many middle events as the value of `resMult`, if the `fault` tagged value is null, or (2) to as many basic events as specified by the `resMult` value, if the `fault` tagged value is not null. Hence, *arm*, *IO card*, *microcontroller* and *power* generate basic events, whereas *control unit* and *RobotSegment1* generate middle events. Packages stereotyped by «daRedundantStructure» generate other middle events (*Control Group*, in the example). Gates are added to the RFT: an OR gate if the «daComponent» does not belong to a «daRedundant Structure» (*RobotSegment1* and *ControlUnit*), an AND gate if the «daComponent» belongs to a «daRedundant Structure» with *ftLevel*=1 (*ControlGroup*). Arcs are then added to complete the RFT structure according to the hierarchical structure modelled in the UML component diagram.

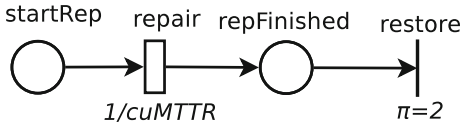

Fig. 11 Maintainability GSPN model

Mapping of the repair mechanism. Both the annotated state machine and component-based structure are used to generate the RFT elements related to the repair facilities, i.e. repair boxes (drawn as up-going triangles). First, a *repair box* is created and connected with a trigger arc to the middle event representing the «daComponent» associated with the state machine (i.e. *ControlUnit*). Then, a repair arc is drawn to the repair box from all the basic events that are "ancestors" of the middle events triggering the repair box (i.e. in this case, *power*, *microcontroller* and *IO card*).

Finally, the GSPN model representing the repair mechanism is generated from the state machine: Fig. 11 depicts the GSPN submodel representing the repair process of the *ControlUnit*. All these elements are added to the RFT for each *ControlUnit* since the generation process is instantiated for each replica of a «daComponent».

7 The approach for performability analysis

In the context of our methodology, Sect. 7.1 defines the `Integration View` (cf., Fig. 4). Section 7.2 describes the model transformation from the `Integration View` to the `integration model`. Section 7.3 describes the `Performability Solution Process` of the methodology. The proposed case study is used as running example.

According to the discussion arisen in Sect. 4, this example just scraped the surface of the problem since it is limited to a specific category. First, we explored the *shared language* solution for the M-integration model, since the performability model is expressed in UML annotated both the MARTE and DAM profiles. Second, we preferred the *no intermediate language* solution for the model transformation, since the integration view is translated into a separate GSPN model by a model transformation which handles in input information coming from the two UML profiles. Finally, there is no need of complex solution process that is simply constituted by a sequential invocation of reliability analysis first and then performability (e.g. *hierarchically structured* solution process).

7.1 Integration view

The integration view for performability modelling (cf., Fig. 4) represents a concrete instantiation of the MIM (multi-DSM integration model) of the multi-formalism Knowledge

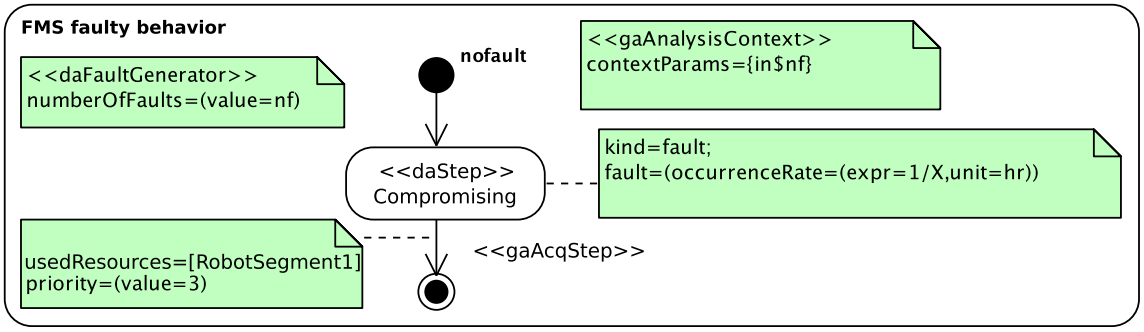

Fig. 12 Faulty behaviour of the FMS workflow

Base structure, shown in Fig. 2. The integration view is aimed at connecting the performance view and the reliability view, which are specified with two different domain-specific languages, i.e. the UML-MARTE and the UML-DAM profiles, respectively. Since performability is a *unified performance reliability measure* [33], it seems natural to choose as integration language (cf., the M-language concept in Fig. 2) the union of the two previous languages, which separately support the modelling and analysis of performance and reliability.

The modelling of the integration view for performability entails two facets: (1) the specification of the faulty behaviour at the system level and (2) the declaration of the global variables. Both facets are considered in the following.

System Faulty Behaviour The two modelling views, described in Sects. 6.1 and 6.2, represent the system at different abstraction levels:

- The performance view (system-level) models the system process under *normal* (i.e. no faulty) condition together with the used resources, whereas
- The reliability view (resource-level) focuses on the internal structure of the faulty resources, and the failure and repair processes of their subcomponents.

To enable performability analysis we need to incorporate the faulty behaviour, due to the failure of the faulty resources, at the system-level specification. This is achieved by building an UML behavioural model, concretely an activity diagram, that specifies the system fault assumption in terms of:

1. The failure occurrence and the characterisation of the resource failure rates.
2. The maximum number of resources that may fail concurrently.
3. The resources that may fail, then provoking a fault at system level according to the fault–error–failure chain in [2].

Concerning the first item, it is addressed by modelling an action that represents the resource failure occurrence. The action is stereotyped by «daStep» (from the DAM profile) and the resource failure rate is specified by the `fault.occurrenceRate` tag.

The second item is addressed only with DAM, by stereotyping the diagram with «daFaultGenerator» and assigning a value to the `numberOfFault` tag.

Finally, once the failure has occurred, the resource is not longer available for the *normal* process. Therefore, a step needs to be explicitly modelled that represents the acquisition of the failed resource by a high priority process (i.e. the fault process). To this aim, this step is stereotyped by «gaAcqStep» and the failed resource is indicated by the `usedResources` tag. Moreover, the priority of the fault process is assigned to the `priority` tag.

Figure 12 shows the faulty behaviour specification of the FMS workflow, due to the failure of the robots used in the first production segment—cf., the `usedResources` tagged value annotated to the transition from the action to the final node. Two stereotypes are assigned to the activity diagram: «daFaultGenerator» and «gaAnalysisContext» to specify the maximum number of resource failures as input variable and to declare the variable (*in$nf*), respectively.

The *compromising* action represents the resource failure occurrence; observe that an expression is assigned to the `fault.occurrenceRate` tag, where a variable X is used in the expression. The latter will be declared as a global variable as described in the following.

Global Variables Declaration In the modelling of the integration view, rules need to be defined to either share or interchange NFP values between the performance and reliability modelling views, which are specified with MARTE and DAM profiles, respectively.

Most of the MARTE and DAM extensions represent NFPs that are expressed using the value specification language (VSL) of MARTE [40]. In particular, VSL enables to declare variables and specify complex expressions—including liter-

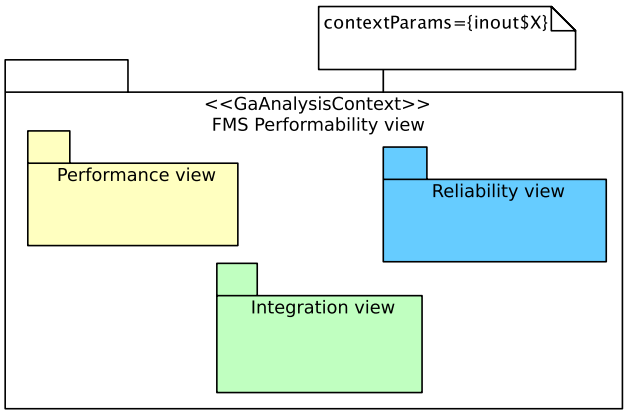

Fig. 13 Global variable declaration

als, variables and mathematical operations—using a well-defined syntax. We exploit the VSL to define four types of integrator operators (see Table 2) that rely on: (1) the declaration of global variables which are used in both the views and (2) the usage of the variables in the two views.

The declaration of a global variable is carried out by stereotyping the package including the two views by «gaAnalysisContext» and by assigning the variable name to the contextParams tag. Depending on the type of integrator operator, the variable can be declared either as input (*in*) or input/output (*inout*).

In the former case—cf., Copy Parameter (CP) and Copy Elaborated Parameter (CEP) operators in Table 2—the two views share the common input variable, whose value will be set during the analysis process.

In the latter case—cf., Copy Result (CR) and Copy Elaborated Result (CER) operators—the two views interchange values throughout the variable which plays two roles: it is an output variable for the view A that is an NFP to be calculated in the analysis of A (observe that the *source* property is explicitly set to *calc*, to indicate that X is an output variable) and it is an input variable for the view B (in this case, no value needs to be set to the *source* property).

The difference between the CP and CEP operators (and, respectively, between CR and CER) is the usage of the variables in the views: the CP (CR) operator assumes that both the views A and B use the variable value (*value* property) while the CEP (CER) operator assumes that the view B uses the variable within an expression (*expr* property).

Figure 13 shows declaration of the global variable X within the performability analysis context of the running example. In particular, the stereotyped package consists of the three modelling views:

- performance view, including the models of Figs. 5 and 6;
- reliability view, including the models of Figs. 8 and 9; and
- integration view, including the model of Fig. 12.

In the FMS example, the CER operator (cf., Table 2) is applied, where *X* is used as output variable in the reliability view (cf., Fig. 8) and as an input variable in the faulty behaviour model (cf., Fig. 12).

7.2 The performability formal model

Similarly to the performance and reliability modelling approaches, detailed in Sect. 6, also in the case of the integration view, a model transformation has to be carried out to obtain an integration formal model (cf., Fig. 4). Moreover, to get the final formal model that can be used for performability analysis, the three formal models—i.e. the performance, reliability and integration ones—need to be *connected* by means of a concrete F-integration language (cf., Fig. 2). In the following, we discuss these two issues.

Transformation to an integration model We can exploit the transformation approach [25], which has been already used in the performance modelling, to get a GSPN model. Indeed, since the integration view is represented by an UML activity diagram, that includes annotations of the MARTE profile, the following mapping rules—already discussed in Sect. 6.1.2—can be applied without changes:

- The mapping of AD transitions that model resources acquisition (i.e. the «gaAcqStep» stereotyped transitions annotated with the usedResources and priority tagged values) to immediate GSPN transitions characterised by input places modelling the resources; and
- The mapping of variables (i.e. the contextParams tagged value associated with the «gaAnalysis Context» stereotyped diagrams) to GSPN input/output parameters.

However, the original approach [25] needs to be extended here to consider the DAM profile annotations in the diagram. Concretely, two new mapping rules are added to translate:

1. The numberOfFaults tagged value of the «daFault Generator» stereotyped diagram, and
2. The fault.occurrenceRate tagged value of the «daStep» stereotyped actions.

Concerning the first rule, the maximum number of faults is mapped to the initial marking of the GSPN place that represents the initial node of the AD, whereas the second rule translates a «daStep» action to a timed transition, which is characterised by a firing rate equal to the fault.occurrenceRate tagged value.

Figure 14 shows the GSPN model generated by the transformation of the activity diagram in Fig. 12. Observe that

Table 2 Integration operators

	Declaration	Usage	
	Package	View A	View B
CP	`contextParams`=(in$X)	`NFP1` = (value=X)	`NFP2` = (value=X)
CEP	`contextParams`=(in$X)	`NFP1` = (value=X)	`NFP2` = (expr=f(X))
CR	`contextParams`=(inout$X)	`NFP1` = (value=X,source=calc)	`NFP2` = (value=X)
CER	`contextParams`=(inout$X)	`NFP1` = (value=X,source=calc)	`NFP2` = (expr=f(X))

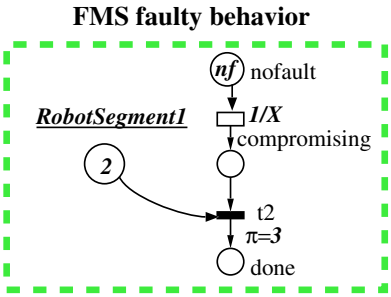

Fig. 14 Integration model

two input parameters have been defined in the model: nf, an initial marking parameter associated with the place *nofault*, and X, used to define the firing rate of the timed transition *compromising*.

Model connection The last modelling issue that has to be solved is the connection of the three formal models—i.e. the performance, reliability and integration models—where all of them are parameterised. In particular, the performance and integration models are GSPN models, whereas the reliability model is an RFT model.

Herein, we use an F-integration language (cf., Fig. 2) that, on the one hand, consists of the place composition operator of GSPNs [20] to connect the performance and the integration models and, on the other hand, enables the definition of global parameters to connect the composed GSPN model and the reliability RFT model.

First, let us consider the composition of the two GSPN models. Therefore, we define a place labelling function $\psi : P \longrightarrow L_P \cup \{\tau\}$ for the GSPN models that assigns to a place $p \in P$ representing a system resource, the name of the resource, and to the rest of places the τ-value. Then, the performance model $\mathcal{F}_p = (\mathcal{LN}_p, \mathcal{P}_p)$ and the integration model $\mathcal{F}_i = (\mathcal{LN}_i, \mathcal{P}_i)$ can be composed using the place composition operator (see "Appendix A"):

$$\mathcal{LN}_{pi} = \mathcal{LN}_p \underset{L_P}{||} \mathcal{LN}_i,$$

where L_P is the set of common labels (i.e. resource names), $\mathcal{LN}_p = (\mathcal{N}_p, \psi)$ and $\mathcal{LN}_i = (\mathcal{N}_i, \psi)$ are the two labelled GSPN models, and $\mathcal{P}_p$ and $\mathcal{P}_i$ are the sets of (input and output) parameters of the two GSPNs.

The composed model $\mathcal{F}_{pi} = (\mathcal{LN}_{pi}, \mathcal{P}_{pi})$ is characterised by a set of parameters $\mathcal{P}_{pi}$ that is the union of the sets $\mathcal{P}_p$ and $\mathcal{P}_i$.

In the FMS example, the set L_P includes one label (i.e. *RobotSegment1*) that identifies the pair of places, in the two GSPN models, representing the robots of the first segment. Figure 15 shows a high-level view (left side) of the composition of the two GSPN models. The composed GSPN model includes three input parameters—N, nf and X—and an output parameter—$thru$.

Secondly, the connection of the resulting GSPN model $\mathcal{F}_{pi} = (\mathcal{LN}_{pi}, \mathcal{P}_{pi})$ and the reliability RFT model $\mathcal{F}_r = (\mathcal{RFT}_r, \mathcal{P}_r)$ is carried out by the definition of the global parameters. The global parameters are those used in different models of a multi-formalism model, then:

$$p \in \mathcal{P}_{pi} \cup \mathcal{P}_r : \begin{cases} p \text{ is global} & \text{if } p \in \mathcal{P}_{pi} \cap \mathcal{P}_r \\ p \text{ is local} & \text{otherwise} \end{cases}$$

Observe that such parameters are mapped from the global variables declared in the performability view (at DSML level). Indeed the latter is a package, stereotyped by `GaAnalysisContext`, that includes all the views (i.e. performance, integration and reliability views); therefore, the variables declared in the context of the performability view can be used in all the included models.

In the running example, there is only one global parameter, i.e. X, that is an output parameter of the reliability model $\mathcal{F}_r = (\mathcal{RFT}_r, \mathcal{P}_r)$ and an input parameter of the composed GSPN model $\mathcal{F}_{pi} = (\mathcal{LN}_{pi}, \mathcal{P}_{pi})$.

7.3 Performability solution process

In this subsection, we address the last phase of the approach (sketched in Fig. 4), by providing a guideline to conduct performability analysis [33] with the multi-formalism model described in the previous subsection.

In particular, the goal of the analysis is the evaluation of the effect of the different repairing policies of the faulty parts of the resources on the overall performance of the system.

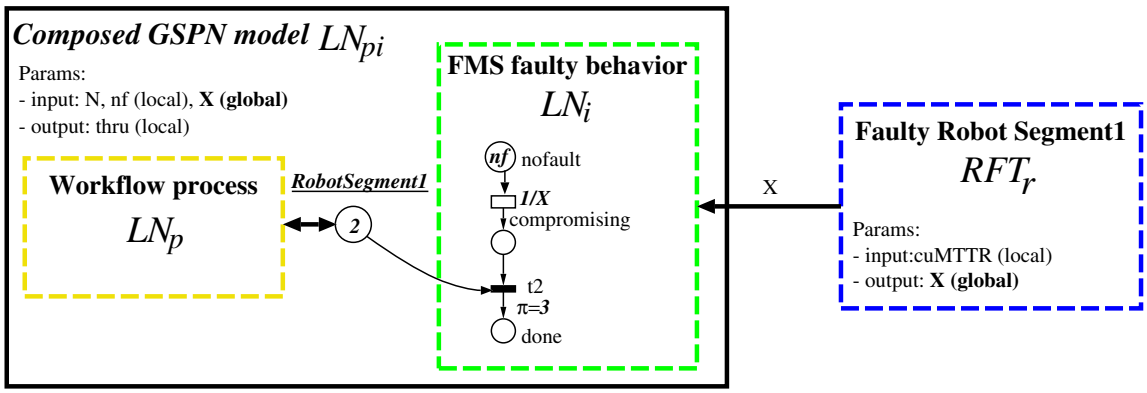

Fig. 15 Performability formal model

The multi-formalism model $\mathcal{MF} = (\mathcal{F}_{pi}, \mathcal{F}_r)$ is characterised by several parameters. In particular, the global parameters are used to exchange values between the RFT model $\mathcal{F}_r = (\mathcal{RFT}_r, \mathcal{P}_r)$ and the GSPN model $\mathcal{F}_{pi} = (\mathcal{LN}_{pi}, \mathcal{P}_{pi})$ and, by construction, the exchange of values is unidirectional, i.e. from the RFT model to the GSPN model. Therefore, the solution approach needs to be sequential: firstly, the RFT model is used to estimate the global parameters, which are then fed to the GSPN model to compute the performability metrics of interest, i.e. the output parameters of the GSPN model. Besides, an issue related to the analysis is the setting of the local input parameters that are unknown,[7] to value(s) which enable to get a meaningful feedback from the results. Parameters' setting is not a trivial task, especially when several parameters have to be considered at the same time.

Therefore, we propose the following solution process that consists of two sequential steps:

Step 1 The performance model $\mathcal{F}_p = (\mathcal{LN}_p, \mathcal{P}_p)$ and the reliability model $\mathcal{F}_r = (\mathcal{RFT}_r, \mathcal{P}_r)$ are analysed independently from each other.

(a) Let $\mathcal{P}_p = \mathcal{P}_p^I \cup \mathcal{P}_p^O$, the goal of the performance analysis is to instantiate the set of input parameters $\mathcal{P}_p^I$ and to evaluate the set of measures of interest $\mathcal{P}_p^O$. The estimated measures will be considered as reference values for the performability analysis.

(b) Let $\mathcal{P}_r = \mathcal{P}_r^I \cup \mathcal{P}^G$, the goal of the reliability analysis is to instantiate the set of input parameters $\mathcal{P}_r^I$ and to evaluate the set of measures of interest $\mathcal{P}^G$ (global parameters). The estimated measures will be used to instantiate the input parameters of the GSPN model $\mathcal{F}_{pi} = (\mathcal{LN}_{pi}, \mathcal{P}_{pi})$.

Step 2 The GSPN model $\mathcal{F}_{pi} = (\mathcal{LN}_{pi}, \mathcal{P}_{pi})$, where $\mathcal{P}_{pi} = \mathcal{P}_p \cup \mathcal{P}_i$, is used to evaluate the set of performance measures of interest $\mathcal{P}_p^O$ under fault assumptions. A subset of input parameters (i.e. $\mathcal{P}_p^I \cup \mathcal{P}^G$) are instantiated according to the (range of) values determined in the previous step. On the other hand, the rest of input parameters have to be instantiated in this second step.

Application of the Solution Process to the FMS Example In the first step (i.e. Step 1.a), we analyse the performance model of Fig. 7 to evaluate the FMS throughput (i.e. number of final items produced per hour). Both steady-state and transient analysis have been carried out by using the reachability graph solvers of the GreatSPN tool [3], with an approximation error of at most 1.0×10^{-5}. The aim of the steady-state analysis is to set the workload parameter (N) to a fixed value such as the system resources are utilised at least 70% of the time. Then, the FMS throughput is calculated for different workload assumptions, considering a Kanban card-like mechanism, until system saturation. Figure 16 plots the curve of the throughput and shows that the maximum reached by the system is 2 products/h. In particular, when $N = 5$, the machines are busy most of the time (i.e. their utilisation is higher than 70%) and the FMS throughput is ≈ 1.75 products/h. In Fig. 18, the curve labelled *no faults* represents the FMS throughput vs/time that has been computed using the performance model with $N = 5$: the asymptote of the curve ($y \approx 1.75$) corresponds to the throughput computed in steady state.

Reliability analysis can be carried out in parallel (i.e. Step 1b) using the reliability model of Fig. 10. Three main steps are considered: modularisation, decomposition and substitution. The modularisation step consists of detecting the RFT state space solution modules (SSMs), i.e. the subtrees in the RFT that must be solved by translating the RFT into a state space formalisms (e.g. the GSPN formalism) and the Combinatorial Solution Modules (CSMs) that can be solved more efficiently by applying combinatorial FT techniques;

[7] Unknown input parameters are those ones which have not been set to a (range of) value(s) in the modelling views, at DSML level.

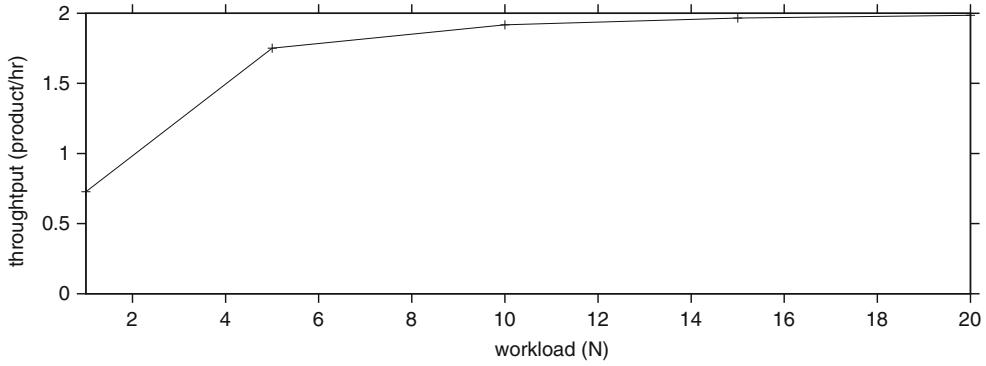

Fig. 16 FMS throughput under different workload assumptions

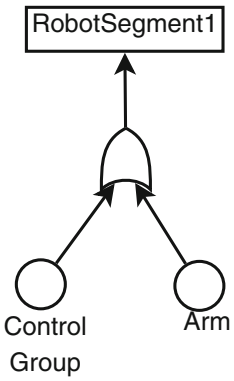

Fig. 17 Reliability derived FT model

the decomposition phase consists of separating and solving each module with the proper technique; in the substitution step, we replace each solved module with a basic event whose probability of failure is equal to the probability of the module to fail. More details about the analysis process of an RFT are in [17].

The subtree of Fig. 10, enclosed in the dashed box, is an SSM and thus should be translated into to a GSPN and then solved. The translation may be supported by model transformations [30] based on advanced transformation techniques as module superimposition [53]. The GSPN resulting from this transformation is not represented for the sake of space. After its analysis (conducted by available solvers as GreatSPN [3]), an FT is generated by collapsing the solved SSM into a basic node (Fig. 17) and solved with available tools [50].

According to the three considered values of the MTTR for the Control Unit (1, 32 h and 42 days), we respectively obtained the following three MTTFs for the RobotSegment1: $\{9.7 \times 10^4, 5.88 \times 10^4, 8.26 \times 10^3\}$ (h).

In the second step of the solution process (i.e. Step 2), the composed GSPN model (Fig. 15, left side) is used to evaluate the FMS throughput under faulty assumptions. The results of the previous performance and reliability analyses are exploited to set the input parameters of the GSPN model. In particular, the workload parameter is set to the fixed value ($N = 5$) and the fault occurrence rate of the *RobotSegment1* is set to the inverse of the MTTF values computed with the reliability model (Step 1.b). There is still an input parameter that needs to be set in this step: the maximum number of faults (nf) that is set to 2, since there are two faulty robots that may fail independently and then leading to FMS throughput degradation.

We carried out transient analysis to compare the system behaviour with the *no faults* case. As in Step 1.a, we have used the reachability graph solver of the GreatSPN tool [3] with an approximation error of at most 1.0×10^{-5}. In Fig. 18, the curves labelled *MTTF* represent the FMS throughput vs

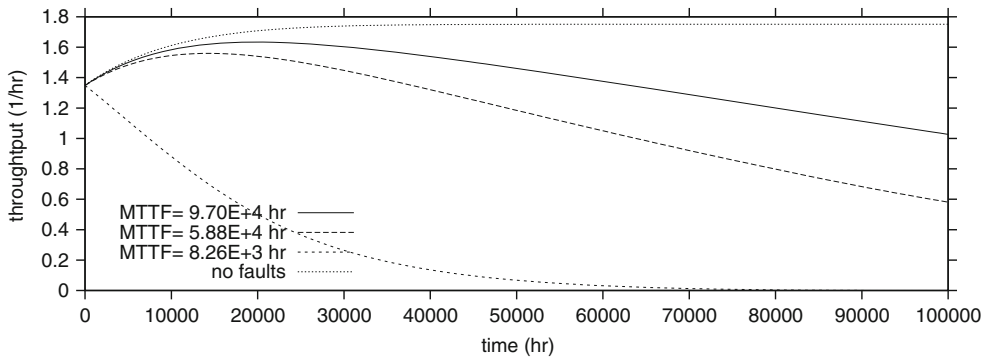

Fig. 18 FMS throughput versus time under different MTTF assumptions

time under different fault occurrence rate assumptions (i.e. 1/MTTF).

The performance degradation can be clearly inferred from the trend of the three curves that, unlike the *no faults* curve, all tend to zero at steady state. However, depending on the MTTF, the curves tend to zero with a different slope. The curves labelled MTTF = 9.7×10^4 h and MTTF = 5.88×10^4 h, reach a maximum value (≈ 1.73 and ≈ 1.70 products/h, respectively) at 10,000 h (i.e. more than one year) which is close to the FMS throughput of the *no faults* case, while the curve labelled MTTF = 8.26×10^3 h is always decreasing and, in particular, the FMS throughput at 10,000 h is ≈ 0.89 products/h.

The analysis results provide an insight into the decision-makers in the selection of an adequate repairing policy of the robot *ControlUnit*. In particular, given that the FMS has to be (almost) fully operational for at least 10,000 h, then the option of having a MTTR of 42 days for the robot *Control Unit* (which corresponds to the curve labelled MTTF = 8.26×10^3 h) should be discarded (i.e. the FMS throughput is reduced to 50% at 10,000 h). On the other hand, the increase in the FMS throughput gained by choosing the option MTTR = 1 h (the curve labelled MTTF = 9.7×10^4 h) instead of MTTR = 32 h (the curve labelled MTTF = 5.88×10^4 h) is less than 2%. Therefore, the choice between the two options is a trade-off between the system performance and the cost of the repairing policies.

8 Conclusions and future work

The analysis of non-functional properties (NFPs) of complex real-world systems requires the availability of proper methodologies and tools supporting both multi-formalism modelling and the automated constructions of models. In the last two decades, many research approaches have been proposed for generating formal models exploiting model-driven principles and techniques, but less effort has been devoted to the automated construction of multi-formalism models.

In this paper, we have defined a framework to support the automated generation of multi-formalism models and its instantiation to the construction of performability models. The framework is based on the model-driven paradigm. Performability aided us in providing a first concrete application of the concepts described in the paper as a full detailed presentation of the framework is not feasible in the scope of a single research paper. We have chosen performability since it has a derived nature (as the conceptual combination of performance and reliability) and it involves different formal models (e.g. GSPNs and RFTs).

Even if this concrete application is just an example used to better explain the aims and the objectives of the proposed framework, it has confirmed some of its features (e.g. the reuse of existing modelling and analysis approaches [9,25] and the suitability of VSL to realise data integration between UML annotated models as well as some open issues (e.g. the need for further modelling guidelines and investigation about different model composition paradigms).

Future research effort will be spent to build the overall framework in a bottom-up manner, starting from diverse domains, varying the languages involved and implementing further transformation chains.

Acknowledgements This research was partially supported by the European Commission under the H2020 Research and Innovation Action [DICE, Grant Agreement No. 644869], the Spanish Ministry of Economy and Competitiveness [ref. CyCriSec-TIN2014-58457-R], the project UZCUD2017-TEC-09, and the Aragonese Government [ref. T94, DIStributed COmputation (DISCO)]

A Generalised Stochastic Petri Nets

Generalised Stochastic Petri Nets (GSPNs) are a well-known modelling paradigm introduced in 1984. GSPNs extend Petri Nets with a temporal specification allowing the description of both the temporal and logical evolution of a system within the same model. Formally, a GSPN is a tuple $\mathcal{N}(P, T, I, O, H, M^0, \Phi, \Lambda)$ where:

- P is the set of places,
- $T = T_I \cup T_E$ is the set of transitions, divided into *immediate* (T_I) and timed *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 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 rate. The firing rate represents the rate parameter of the negative exponential distribution.

Graphically, a GSPN model is a directed bipartite graph in which places are drawn as circles and transitions are drawn as bars (immediate transitions) or boxes (timed ones). The arcs are the oriented edges of the graph. An inhibitor arc is a circle-headed arc from a place to a transition, which prevents the transition to be enabled if the place contains a number

of tokens equal or greater than the multiplicity of the arc. Tokens are markers within places and are used to specify the state of a GSPN. They are drawn as black dots.

The dynamic behaviour of the GSPN is defined by the enabling and firing rules. A transition is *enabled* in a marking M iff both the following conditions occur: (1) its input places contain at least as many tokens as the corresponding arc multiplicities and its inhibitor places contain less tokens than the corresponding arc multiplicities; (2) its priority is greater or equal to the one of the transitions t' also satisfying condition (1) in M. Consequently, only transitions of the same priority level can be enabled in a marking. A transition t, enabled in marking M, may fire then leading to a new marking M', according to the equation: $M'(p) = M(p) + O(p,t) - I(p,t),\ p \in P$. The reader may refer to [32] for an introduction to GSPN modelling.

A.1 GSPN composition

Two GSPN models can be composed over places (or transitions), provided that a labelling function is defined over the set of places (or transitions) [20]. In general, more than one label can be associated with a place (transition). However, the composition operator used in this paper is a simplified version of the one defined in [20], since only place composition and at most one label per place are considered. Therefore, we consider a labelled GSPN $\mathcal{LN} = (\mathcal{N}, \psi)$, where $\mathcal{N}$ is a GSPN and $\psi : P \to L_P \cup \{\tau\}$ is the place labelling function that assigns a label (or τ) to each place.

Given two labelled GSPN $\mathcal{LN}_1 = (\mathcal{N}_1, \psi_1)$ and $\mathcal{LN}_2 = (\mathcal{N}_2, \psi_2)$, the labelled GSPN $\mathcal{LN} = (\mathcal{N}, \psi)$:

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

resulting from the composition over the sets of place labels L_P is defined as follows. Let $E_P = L_P \cap \psi_1(P_1) \cap \psi_2(P_2)$ be the subset of L_P comprising place labels that are common to the two labelled GSPNs, P_1^l be the set of places of $\mathcal{LN}_1$ that are labelled l and $P_1^{E_P}$ be the set of all places in $\mathcal{LN}_1$ that are labelled with a label in E_P. Same definitions apply to $\mathcal{LN}_2$. Then: $P = P_1 \backslash P_1^{E_P} \cup P_2 \backslash P_2^{E_P} \cup \bigcup_{l \in E_P} \{P_1^l \times P_2^l\}$, $T = T_1 \cup T_2$. The functions $F \in \{I(), O(), H()\}$ are equal to:

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

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

$$F(t) = \begin{cases} F_1(t) & \text{if } t \in T_1 \\ F_2(t) & \text{if } t \in T_2 \end{cases}$$

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 place labelling function is equal to:

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

B Repairable fault trees

Repairable fault trees (RFTs) have been introduced to allow the evaluation of the effects of complex repair policies on the availability of a system [17]. RFTs integrate Generalised Stochastic Petri Nets (GSPNs) and fault trees (FTs): repair actions are represented by nodes, called *repair boxes* (RBs), which encapsulate a GSPN model. RBs are connected to a FT which describes the faults that may happen in the system and their contribution to the occurrence of a failure. An RB b is characterised by a repair policy, a vector of parameters related to the repair policy and a vector of repair rates. The inputs of the node may only be *basic events* of the FT and its output is an unique event node, called the *Trigger Event* of b. Hence, an RB is connected to the FT by arcs linking the RB to the event node of the FT which triggers the repair action and to a subset of basic events (i.e. of tree leaves) which represent the repairable components of the systems affected by the repair action. The introduction of RBs requires that each RFT subtree whose event node triggers a repair action is translated into an equivalent GSPN. This technique is based on the hypothesis that there is no mutual dependence between triggering events, in addition a subtree whose root event node is connected to an RB cannot contain other RBs.

References

1. Abouzahra, A., Bézivin, J., Del Fabro, M.D., Jouault, F.: A practical approach to bridging domain specific languages with UML profiles. In: *Proceedings of the Best Practices for Model Driven Software Development (co-located with OOPSLA 2005), San Diego, California, USA, October 16, 2005* (2005)
2. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.E.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. **1**(1), 11–33 (2004)
3. Baarir, S., Beccuti, M., Cerotti, D., De Pierro, M., Donatelli, S., Franceschinis, G.: The GreatSPN tool: recent enhancements. SIGMETRICS Perform. Eval. Rev. **36**(4), 4–9 (2009)
4. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: a survey. IEEE Trans. Softw. Eng. **30**(5), 295–310 (2004)

5. Barbierato, E., Gribaudo, M., Iacono, M.: Modeling hybrid systems in SIMTHESys. Electron. Notes Theor. Comput. Sci. **327**, 5–25 (2016)
6. Battista, E., Casola, V., Marrone, S., Mazzocca, N., Nardone, R., Vittorini, V.: An integrated lifetime and network quality model of large WSNs. In: 2013 IEEE International workshop on measurements networking (MN), pp. 132–137 (2013)
7. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. J. Syst. Softw. **82**(1), 3–22 (2009)
8. Berardinelli, L., Bernardi, S., Cortellessa, V., Merseguer, J.: UML profiles for non-functional properties at work: analyzing reliability, availability and performance. In: Boskovic, M., Gasevic, D., Pahl, C., Schätz, B. (eds.) Proceedings of the 2nd International Workshop on Non-functional System Properties in Domain Specific Modeling Languages (NFPinDSML2009) Affiliated with MoDELS2009
9. Bernardi, S., Flammini, F., Marrone, S., Merseguer, J., Papa, C., Vittorini, V.: Model-driven availability evaluation of railway control systems. SAFECOMP 2011. volume 6894 of LNCS, pp. 15–28. Springer, Berlin, Heidelberg (2011)
10. Bernardi, S., Merseguer, J., Petriu, D.C.: Dependability modeling and analysis of software systems specified with UML. ACM Comput. Surv. **45**(1), 2 (2012)
11. Bernardi, S., Merseguer, J., Petriu, D.C.: Model-Driven Dependability Assessment of Software Systems. Springer, Berlin (2013)
12. Bernardi, S., Merseguer, J., Petriu, D.C.: A dependability profile within MARTE. Softw. Syst. Model. **10**(3), 313–336 (2011)
13. Boulanger, F., Jacquet, C., Hardebolle, C., Rouis, E.: Modeling heterogeneous points of view with modhel'x. In: Ghosh, S. (ed.) Models in Software Engineering: Workshops and Symposia at MoDELS 2009, Denver, CO, USA, October 2009, Reports and Revised Selected Papers, volume 6002 of Lecture Notes in Computer Science, pp. 310–324. Springer, Berlin (2010)
14. Bracchi, B., Cukic, B., Cortellessa, V.: Performability modeling of mobile software systems. In: 15th International Symposium on Software Reliability Engineering (ISSRE 2004), Saint-Malo, Bretagne, France, pp. 77–88. IEEE Computer Society (2004)
15. Brooks, C., Lee, E.A.: Ptolemy II—Heterogeneous Concurrent Modeling and Design in Java. Poster presented at the 2010 Berkeley EECS Annual Research Symposium (BEARS) February (2010)
16. Ciancone, A., Drago, M.L., Filieri, A., Grassi, V., Koziolek, H., Mirandola, R.: The KlaperSuite framework for model-driven reliability analysis of component-based systems. Softw. Syst. Model. **13**(4), 1269–1290 (2014)
17. Codetta Raiteri, D., Iacono, M., Franceschinis, G., Vittorini, V.: Repairable fault tree for the automatic evaluation of repair policies. In: *Proceedings of the 2004 international conference on dependable systems and networks*, pp. 659–668 (2004)
18. de Lara, J., Vangheluwe, H.: Atom^3: a tool for multi-formalism and meta-modelling. In: Fundamental Approaches to Software Engineering, 5th International Conference, FASE 2002, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8–12, 2002, Proceedings, pp. 174–188 (2002)
19. Deavours, D.D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J.M., Sanders, W.H., Webster, P.G.: The Möbius framework and its implementation. IEEE Trans. Softw. Eng. **28**(10), 956–969 (2002)
20. Donatelli, S., Franceschinis, G.: The PSR methodology: integrating hardware and software models. In: Billington, J., Reisig, W. (eds.) Application and Theory of Petri Nets, Volume 1091 of LNCS, pp. 133–152. Springer, Berlin (1996)
21. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Sachs, S., Xiong, Y., Neuendorffer, S.: Taming heterogeneity—the Ptolemy approach. Proc. IEEE **91**(1), 127–144 (2003)
22. Emerson, M., Sztipanovits, J.: Techniques for metamodel composition. In: The 6th OOPSLA Workshop on Domain-Specific Modeling. OOPSLA 2006, pp. 123–139. ACM, ACM Press (2006)
23. Fitzgerald, J., Larsen, P.G., Pierce, K., Verhoef, M., Wolff, S.: Collaborative modelling and co-simulation in the development of dependable embedded systems. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6396 LNCS:12–26. cited By 6 (2010)
24. Flammini, F., Marrone, S., Iacono, M., Mazzocca, N., Vittorini, V.: A multiformalism modular approach to ERTMS/ETCS failure modeling. Int. J. Reliab. Qual. Saf. Eng. **21**(1), 1450001 (2014)
25. Gómez-Martínez, E., González-Cabero, R., Merseguer, J.: Performance assessment of an architecture with adaptative interfaces for people with special needs. Empir. Softw. Eng. **19**(6), 1967–2018 (2014)
26. Hardebolle, C., Boulanger, F.: Exploring multi-paradigm modeling techniques. Simul. Trans. Soc. Model. Simul. Int.l **85**(11/12), 688–708 (2009)
27. Khan, R.H., Machida, F., Heegaard, P.E., Trivedi, K.S.: A performability modeling framework considering service components deployment. Int. J. Adv. Netw. Serv. **5**(3–4), 346–366 (2012)
28. Koziolek, A., Avritzer, A., Suresh, S., Sadoc Menasche, D., Trivedi, K., Happe, L.: Design of distribution automation networks using survivability modeling and power flow equations. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), pp. 41–50 (2013)
29. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., Volgyesi, P.: The generic modeling environment. In: Workshop on Intelligent Signal Processing (2001)
30. Marrone, S., Papa, C., Vittorini, V.: Multiformalism and transformation inheritance for dependability analysis of critical systems. In: Proceedings of 8th Integrated Formal Methods, IFM'10, pp. 215–228. Springer, Berlin, Heidelberg (2010)
31. Marrone, S., Rodríguez, R.J., Nardone, R., Flammini, F., Vittorini, V.: On synergies of cyber and physical security modelling in vulnerability assessment of railway systems. Comput. Electr. Eng. **47**, 275–285 (2015)
32. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets, 1st edn. Wiley, New York (1994)
33. Meyer, J.F.: Performability: a retrospective and some pointers to the future. Perform. Eval. **14**(3–4), 139–156 (1992)
34. Montecchi, L., Lollini, P., Bondavalli, A..: An intermediate dependability model for state-based dependability analysis. Technical Report rcl101115, University of Florence, Dip. Sistemi Informatica, RCL group (2011)
35. Moscato, F., Vittorini, V., Amato, F., Mazzeo, A., Mazzocca, N.: Solution workflows for model-based analysis of complex systems. IEEE Trans. Autom. Sci. Eng. **9**(1), 83–95 (2012)
36. Mosterman, P.J., Vangheluwe, H.: Computer automated multiparadigm modeling: an introduction. Simulation **80**(9), 433–450 (2004)
37. Nicol, D.M., Sanders, W.H., Trivedi, K.S.: Model-based evaluation: from dependability to security. IEEE Trans. Dependable Secure Comput. **1**(1), 48–65 (2004)
38. OMG: UML Profile for Schedulability, Performance, and Time. Version 1.1, formal/05-01-02 (2005)
39. OMG: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. Version 1.1, formal/08-04-05 (2008)
40. OMG: UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems. Version 1.1, formal/11-06-02 (2011)
41. OMG: Unified Modeling Language: Infrastructure and Superstructure. Version 2.4, formal/11-08-05 (2011)

42. OMG: System Modeling Language. Version 1.3, OMG document formal/2012-06-01 (2012)
43. SAE International: Architecture Analysis and Design Language. SAE International, Warrendale (2009)
44. Sangiovanni-Vincentelli, A.: Quo vadis SLD: reasoning about trends and challenges of system-level design. Proc. IEEE **95**(3), 467–506 (2007)
45. Sarjoughian, H.S.: Model composability. In: Proceedings of the Winter Simulation Conference WSC 2006, Monterey, California, USA, December 3–6, 2006, pp. 149–158 (2006)
46. Sprinkle, J., Rumpe, B., Vangheluwe, H., Karsai, G.: Metamodelling: State of the Art and Research Challenges. CoRR, abs/1409.2359 (2014)
47. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0, 2nd edn. Addison-Wesley Professional, Reading (2009)
48. The Eclipse Foundation. Papyrus https://eclipse.org/papyrus/ (2016)
49. Tolvanen, J.-P., Kelly, S.: MetaEdit+: defining and using integrated domain-specific modeling languages. In: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, OOPSLA '09, pp. 819–820. ACM, New York, NY, USA (2009)
50. Trivedi, K.S., Sahner, R.: SHARPE at the age of twenty two. SIGMETRICS Perform. Eval. Rev. **36**(4), 52–57 (2009)
51. Trowitzsch, J., Jerzynek, D., Zimmermann, A.: A toolkit for performability evaluation based on stochastic UML state machines. In: Glynn, P.W. (ed.) Proceedings of the 2nd International Conference on Performance Evaluation Methodolgies and Tools, VALUETOOLS 2007, Nantes, France, 2007, ACM International Conference Proceeding, p. 30. ACM (2007)
52. Vittorini, V., Iacono, M., Mazzocca, N., Franceschinis, G.: The OsMoSys approach to multi-formalism modeling of systems. Softw. Syst. Model. **3**(1), 68–81 (2004)
53. Wagelaar, D., Van Der Straeten, R., Deridder, D.: Module superimposition: a composition technique for rule-based model transformation languages. Softw. Syst. Model. **9**(3), 285–309 (2010)
54. Woodside, C.M., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., Merseguer, J.: Performance by unified model analysis (PUMA). In: Proceedings of the Fifth International Workshop on Software and Performance, WOSP 2005, Palma, Illes Balears, Spain, July 12–14, 2005, pp. 1–12. ACM (2005)
55. Woodside, C.M., Petriu, D.C., Merseguer, J., Petriu, D.B., Alhaj, M.: Transformation challenges: from software models to performance models. Softw. Syst. Model. **13**(4), 1529–1552 (2014)
56. Zhang, T., Jouault, F., Bézivin, J., Li, X.: An MDE-based method for bridging different design notations. ISSE **4**(3), 203–213 (2008)

Simona Bernardi is an Assistant Professor in the Department of Computer Science and Systems Engineering at the University of Zaragoza, Spain. She received a M.S. degree in mathematics and a Ph.D. degree in computer science, in 1997 and 2003, respectively, both from the University of Torino, Italy. Her research interests are in the area of software engineering, in particular model driven engineering, verification and validation of performance, dependability and survivability software requirements, and formal methods for the modelling and analysis of software systems. She has served as a referee for international journals and as a program committee member for several international conferences and workshops. She is a member of the IEEE-SMC Homeland Security technical committee.

Stefano Marrone is an assistant professor in Computer Engineering at Università della Campania "Luigi Vanvitelli", Italy. His interests include the definition of model driven processes for the design and the analysis of transportation control systems, complex communication networks and critical infrastructures. He is involved in research projects with both academic and industrial partners.

José Merseguer received the B.S. and M.S. degrees in computer science and software engineering from the Technical University of Valencia, Spain, and the Ph.D. degree in computer science from the University of Zaragoza, Spain. He is with the Department of Computer Science and Systems Engineering at the University of Zaragoza, Spain, where he teaches software engineering courses at graduate and undergraduate levels. He was the director of the Computer Science Master at the University of Zaragoza. His main research interests include performance and dependability analysis of software systems. He has developed postdoctoral research with Prof. M. Woodside at Carleton University, Ottawa, ON, Canada, and with Prof. R. Lutz at Iowa State University, Ames, IA, USA. He has been a Visiting Researcher with the University of Turin, with the University of Cagliari, and with the Politecnico di Milano, Italy. He is also a member of the Aragon Institute of Engineering Research. He has been serving as a referee for international journals and as a Program Committee member for several international conferences and workshops. He is co-author of the book "Model-driven dependability assessment of software-systems", Springer, and has advised three Ph.D. doctoral thesis.

Roberto Nardone is a postdoctoral fellow at University of Naples Federico II. His research interests are in the area of quantitative evaluation of non-functional properties, with a particular focus on dependability and performability assessment and threat propagation analysis, by means of model-based and model-driven techniques. He is currently involved in research projects with both academic and industrial partners.

Valeria Vittorini is Associate Professor at University of Naples Federico II since 2005. She received the Laurea degree (cum laude) in Mathematics from University of Napoli Federico II in 1991, and a Ph.D. in Computer Engineering from the same University in 1995. She teaches computer programming, formal modeling, workflow and process automation. Her current research interests include dependability and performance evaluation of computer systems, validation and verification of critical systems, critical infrastructures protection and model-driven approaches applied to the automatic generation of formal models.