FaultFlow: an MDE Library for Dependability Evaluation of Component-Based Systems

Laura Carnevali, *Member, IEEE*, Stefania Cerboni, Leonardo Montecchi, and Enrico Vicario, *Member, IEEE*

Abstract—We present a Model Driven Engineering (MDE) approach to dependability evaluation of component-based coherent dyadic systems, implemented by the FaultFlow library, combining simple high-level modeling with powerful quantitative evaluation methods. In the functional perspective, distinctive features are: modeling of fault propagations within individual components and between different components, possibly not connected through physical or communication interfaces; support for non-Markovian distributions, both for the times to the occurrence of faults and for the duration of fault-to-failure propagations; derivation of the distribution of the time to the occurrence of a given failure; derivation of fault importance measures, for models where each fault does not propagate into multiple failures and, viceversa, each failure does not act as fault to multiple components, achieving evaluation efficiency even for significantly complex systems with hundreds of different faults. In the implementation perspective, distinctive features are: definition of a custom-made extensible metamodel to specify the system structure and failure logic; automated derivation of metamodel instances from Systems Modeling Language (SysML) Block Definition Diagrams (BDDs) and Stochastic Static Fault Trees (SSFTs); automated derivation of the mentioned dependability measures; open source availability. We illustrate the typical modeling and evaluation workflow with relevant uses cases, comparing functionalities with those of other dependability evaluation tools.

Index Terms—Dependability evaluation, component-based coherent systems, Model-Driven Engineering (MDE), SysML Block Definition Diagrams (BDDs), non-Markovian Stochastic Static Fault Trees (SSFTs), importance measures, software tools and libraries.

1 Introduction

In the development of dependable component-based systems [7], models of the failure logic [42] support the analysis of the causes and consequences of failures. The chain of threats to dependability [5] starts when a fault is activated, bringing a component into an error state. Then, the error can propagate in the component up to cause its failure (*intra-component fault-to-failure* propagation). In turn, a component failure can act as external fault for other components (*inter-component failure-to-fault* propagation). When fault-to-failure and failure-to-fault propagations have probabilistic characterization in time, quantitative evaluation of stochastic models of the system failure logic enables derivation of metrics of dependability [36], [57], [65], supporting early validation of design choices and development of predictive analytics for proactive fault management [59].

Model-Driven Engineering (MDE) [18], [61] has been widely employed to automatically derive dependability models from other artifacts, preserving consolidated industrial processes as prescribed by various certification standards [1], [23], [34], [53], [54], while exploiting transformation rules to guarantee affordable complexity of model analysis with respect to the class and size of the underlying stochastic process. To this end, extensions of system design languages like the Unified Modeling Language (UML) and the Architecture Analysis & Design Language (AADL) [27] have been defined to support representation of non-functional properties. Notable extensions include the UML profile for Modeling and Analysis of Real-Time Embedded systems (MARTE) [49], the UML profile for Dependability Analysis and Modeling (DAM) [6], the Systems Modeling Language (SysML) [50], and the AADL Annexes on Error Model [58] and Safety [67]. Many tools leverage these and other high-level artifacts to specify the system failure logic, and automatically transform them into formal stochastic models to analyze dependability. CHESS [9], [46] supports CHESS-ML, a custom UML-based language including tailored subsets of MARTE and SysML, and implements translation into Continuous Time Markov Chains (CTMCs) and Stochastic Petri Nets (SPNs) [16], using mainly stochastic simulation for dependability evaluation. OSATE [20], [25] provides a reference implementation of AADL and its Annexes, with translation into Static Fault Trees (SFTs) and Reliability Block Diagrams (RBDs), and then into Discrete Time Markov Chain (DTMCs) and CTMCs that can be analyzed by PRISM [40]. ADAPT [55] supports translation of AADL models (annotated with dependability attributes) into Generalized Stochastic Petri Nets (GSPNs) [2] to perform the analysis by external tools. ASTRO [62] provides an integrated environment for dependability evaluation by supporting modeling and analysis of RBDs, CTMCs, and SPNs. Except for CHESS, these tools rely on artifacts that intrinsically bind specification of the system structure with its failure logic, modeling *direct couplings* (i.e., low-level failure propagations directly connected among components, by physical or communication interfaces) but not *indirect couplings* (i.e., high-level failure propagations among not directly connected components). Moreover, none of

L. Carnevali, S. Cerboni, and E. Vicario are with the Department of Information Engineering, University of Florence, Via di Santa Marta 3, 50139 Firenze, Italy. E-mail: {laura.carnevali,stefania.cerboni, enrico.vicario}@unifi.it
L. Montecchi is with the Department of Computer Science, Norwegian University of Science and Technology, Sem Sælands vei 7-9, 7034 Trondheim, Norway. E-mail: leonardo.montecchi@ntnu.no
Manuscript received Month Day, Year; revised Month Day, Year.

these tools supports state-space analysis for models with multiple concurrent non-Markovian durations, i.e., with non-Exponential general (GEN) distribution possibly with bounded support, which actually characterize the system behavior in many application domains [69].

Other tools like DFTCalc [4], SAFEST [74], DFTRES [11], SHyFTOO [15], DFTSim [8], RAATS [43], MatCarloRE [44], RADYBAN [45], and Galileo [21] evaluate dependability of systems specified by Dynamic Fault Trees (DTFs) [57], in part or entirely, achieving significantly greater expressivity by modeling dependencies among the behaviors of components, e.g., dependent events, spare components, different operational modes. To limit the greater complexity of the analysis of the underlying stochastic process, these tools either restrain duration distributions in the Markovian setting or resort to simulation-based solution methods.

Other tools like Möbius [19], GreatSPN [3], PRISM [39], Mercury [63], SHARPE [70], SMART [17], TimeNET [76], CPN IDE [72], FIG [10], and ORIS [51] implement quantitative evaluation methods using modeling formalisms that have different expressivity, e.g., Stochastic Activity Networks (SANs) [60], Performance Evaluation Process Algebra (PEPA) [32], GSPNs, SPNs, Deterministic and Stochastic Petri Nets (DSPNs) [41], Coloured Petri Nets (CPNs) [35], Input/Output Stochastic Automata (IOSA) [22], and Stochastic Time Petri Nets (STPNs) [73]. These tools do not support automated derivation of dependability models from high-level artifacts. Thus, using them to model and analyze the failure logic of complex systems requires domain analysts to have strong expertise in stochastic modeling and analysis.

In this paper, we present the MDE approach to dependability evaluation of component-based coherent[1] dyadic[2] systems, implemented by the FaultFlow Java library [24]. It models both *structural* information on the system hierarchical composition, specified by a SysML Block Definition Diagram (BDD) [50], and *behavioral* information on fault propagations, specified by a Stochastic Static Fault Tree (SSFT), a variant of Component Fault Tree (CFT) [31], [37] with probabilistic choices and delays characterized by a continuous-time probability distribution. Fault propagations occur via physical and communication interfaces between components, or stigmergic information flows [38], i.e., component-environment interactions affecting other components within the same environment. The duration to the occurrence or propagation of a given fault is characterized by a non-Markovian Probability Density Function (PDF), possibly with firmly bounded support, facilitating fitting of analytical distributions from statistical data. FaultFlow derives the Cumulative Distribution Function (CDF) of the time to the occurrence of a given failure and, if no fault propagates into different failures and viceversa, importance measures [75] characterizing how faults contribute to failures over time, i.e., numerical ranks used in reliability engineering to quantify the impact of component faults on the entire system and to decide which components to maintain first.

FaultFlow offers a custom-made extensible metamodel, not tied to specific frameworks like the Ecore metamodel of the Eclipse Modeling Framework [66], supporting Model-to-Model (M2M) transformations and integration with tools and libraries for quantitative evaluation of stochastic models. A metamodel instance can be automatically derived from a SysML BDD modeling the system structure and an SSFT modeling the system failure logic, guaranteeing a *correct-by-construction* configuration of metamodel instances. A metamodel instance can be automatically translated into an STPN, encoded as an instance of the metamodel of the Sirio Java library[3] of the ORIS tool, automatically deriving the time-to-failure CDF through the analysis method of [33]. If no fault propagates into different failures and viceversa, a metamodel instance can be automatically translated into the statechart-based formalism of Hierarchical Semi-Markov Processes with parallel regions (HSMPs) [13], encoded as instance of the metamodel of the Pyramis Java library[4]. The M2M transformation enables separate analysis of independent fault propagations by the solution method of [13], guaranteeing efficient evaluation of the time-to-failure CDF and importace measures of faults, even for significantly complex systems with hundreds of different faults.

Major contributions with respect to the FaultFlow API preliminarily presented in [52] include: i) extensive description of the workflow and relevant use cases; ii) integration with Pyramis, implementing automated transformation of instances of the FaultFlow metamodel into instances of the Pyramis metamodel; iii) formal proof of correctness of such model-to-model transformation, by characterizing the set of STPN models that defines the semantics of the two metamodels; iv) application to two case studies, showing efficient computation of importance measures that characterize the contribution of faults to failures over time; v) comparison of the capabilities with those of other competing tools for dependability evaluation; and vi) open-source release of the FaultFlow library under the AGPLv3 licence [24], also supporting replication of the experimental results.

In the following, first we illustrate the FaultFlow capabilities (Section 2), syntax, and semantics (Section 3). Then, we recall the Pyramis syntax and semantics, we translate FaultFlow metamodel instances into Pyramis metamodel instances, and we derive importance measures of faults (Section 4). Next, we illustrate relevant FaultFlow use cases (Section 5) and we compare the FaultFlow capabilities with those of other tools for dependability evaluation (Section 6). Finally, we draw our conclusions (Section 7). The proof of Theorem 1 is reported in the Supplemental Material, together with some details on the experiments of Section 5.

2 FaultFlow overview

In this section, we characterize the class of component-based systems (Section 2.1) as well as the dependability measures supported by FaultFlow (Section 2.2), and we illustrate the typical modeling and evaluation workflow (Section 2.3).

2.1 Component-based systems

FaultFlow supports modeling of component-based systems, with structure specified by a SysML BDD through the com-

1. A coherent system is a system where an additional event cannot cause the top-level event to switch from failed to operational again.
2. A dyadic system is a system that can be either operational or failed.
3. https://github.com/oris-tool/sirio
4. https://github.com/oris-tool/pyramis

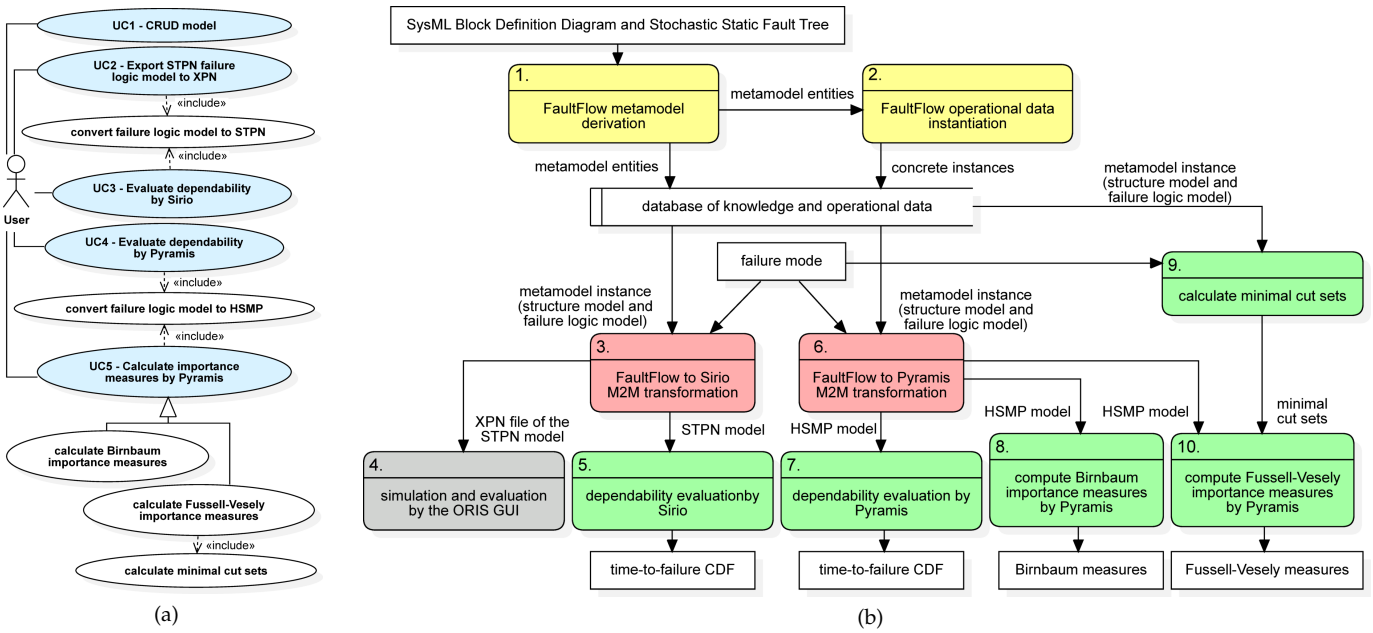

Fig. 1. FaultFlow library: (a) UML use case diagram of the main functionalities provided by the FaultFlow API (light blue highlights main use cases); (b) UML dataflow diagram of the typical modeling and evaluation workflow (yellow is used for modeling activities, red for M2M transformations, green for evaluation activities supported by the API, and gray for evaluation activities manually performed through the GUI of the external tool ORIS).

position of blocks, according to the following EBNF syntax:

$$\texttt{system:=block}$$
$$\texttt{block:=component} \mid \texttt{composite}\{\texttt{block}_1,\ldots,\texttt{block}_n\}$$

FaultFlow models the failure logic of component-based coherent dyadic systems by means of SSFTs, an extension of SFTs defined by the following EBNF syntax:

$$\texttt{SSFT:=node}$$
$$\texttt{node:=leaf} \mid \texttt{gate} \mid \texttt{prop(leaf)} \mid \texttt{prop(gate)}$$
$$\texttt{gate:=AND}\{\texttt{node}_1,\ldots,\texttt{node}_n\} \mid \texttt{OR}\{\texttt{node}_1,\ldots,\texttt{node}_n\} \mid \texttt{VOT}\{k,\texttt{node}_1,\ldots,\texttt{node}_n\}$$

Specifically, leaf nodes model internal faults of components, whose occurrence has a stochastic duration characterized by a time-to-fault PDF (see Definition 1); logical gates model conditions that activate fault-to-failure propagations (which can be activated also by individual faults); and, propagation nodes model fault-to-failure propagations taking a stochastic delay characterized by a fault-to-failure PDF (se Definition 2) and, possibly, failure-to-fault propagations occurring with a given probability(see Definition 3).

Definition 1 (Time-to-fault PDF)**.** Given a fault x of a component, the time-to-fault PDF $f_x^{\text{fault}}(t)$ is the PDF of the duration elapsing from the initial time to the fault occurrence (the fault occurrence is also termed fault activation).

Definition 2 (Fault-to-failure PDF)**.** Given a failure y of a component or system, caused by a set of faults identified by a boolean expression (with AND, OR, and VOT(k/N) operators), the fault-to-failure PDF $f_y^{\text{prop}}(t)$ is the PDF of the duration elapsing from the time at which the boolean expression of faults becomes true to the failure occurrence.

Definition 3 (Failure-to-fault probability)**.** Given a failure y of a component or system, the failure-to-fault probability p_y is the probability (not depending on time) that failure y acts as external fault for other components or systems.

Given that FaultFlow represents coherent systems, SSFTs consist of AND gates (i.e., the output event occurs if all input events occur), OR gates (i.e., the output event occurs if any input event occurs), and VOT(k/N) gates (i.e., the output event occurs if at least k of the N input events occur).

Moreover, the time-to-fault PDFs and the fault-to-failure PDFs are provided in analytical form in the class of expolynomial functions, also termed exponomials [70], defined as the sum of products of Exponential and polynomial terms, i.e., $f(x) = \sum_{m=0}^{M-1} c_m \prod_{n=0}^{N-1} x_n^{\alpha_{mn}} e^{-\lambda_{mn} x_n}$. In particular, exponomials may have analytical representation over the entire domain or be piecewise-defined over multiple subdomains, they facilitate fitting of analytical PDFs from statistics of duration data, and they are supported by the ORIS tool [51] and its Sirio Java library, exploited by FaultFlow for the evaluation of dependability measures.

2.2 Dependability measures

FaultFlow derives the duration CDF of failure processes (not necessarily top-level failures) in numerical form, based on the structure of the SSF, the time-to-fault PDFs, and the fault-to-failure PDFs (as illustrated in Sections 3.2 and 4.3).

Definition 4 (Time-to-failure CDF)**.** Given a failure y of a component, the time-to-failure CDF F_y^{fail} is the CDF of the duration from the initial time to the failure occurrence.

Note that, in Definition 4, the failure may be the top-level system failure. Moreover, the duration is measured since the initial time, not since the activation of faults triggering the fault-to-failure propagation, as done in Definition 2.

Evaluation also supports computation of the Birnbaum measures and the Fussel-Vesely measures of faults [71].

Definition 5 (Birnbaum measure)**.** The Birnbaum measure $I_x^{\mathrm{B}}(t)$ of fault x at time t is the difference between the system time-to-failure CDF $F_{\mathrm{sys}}^{\mathrm{fail}}(t)$ at t given that x has occurred by t and $F_{\mathrm{sys}}^{\mathrm{fail}}(t)$ given that x has not occurred by t:

$$I_x^{\mathrm{B}}(t) = F_{\mathrm{sys}}^{\mathrm{fail}}(t)_{|F_x^{\mathrm{fault}}(t)=1} - F_{\mathrm{sys}}^{\mathrm{fail}}(t)_{|F_x^{\mathrm{fault}}(t)=0} \quad \forall\, t \qquad (1)$$

where F_x^{fault} is the time-to-fault CDF of fault x, i.e., $F_x^{\mathrm{fault}}(t) = \int_0^t f_x^{\mathrm{fault}}(\tau)\, d\tau$, where $f_x^{\mathrm{fault}}(\tau)$ if the time-to-fault PDF.

For each time t, $F_{\mathrm{sys}}^{\mathrm{fail}}(t)_{|F_x^{\mathrm{fault}}(t)=1}$ can be computed as the system time-to-failure CDF by assuming that $F_x^{\mathrm{fault}}(t)$ is the generalized CDF of a Dirac Delta function[5] centered at $t = 0$, and $F_{\mathrm{sys}}^{\mathrm{fail}}(t)_{|F_x^{\mathrm{fault}}(t)=0}$ can be derived as the system time-to-failure CDF by removing fault x and the related fault-to-failure propagations from the system failure logic.

The derivation of the Fussell-Vesely measure requires the computation of the Minimal Cut Sets (MCSs) [68].

Definition 6 (Minimal Cut Set)**.** Given a failure y of a component or system, a Minimal Cut Set (MCS) is a minimal combination of faults that induces the failure y.

For instance, let a fault-to-failure propagation be started when the following boolean expression of faults $x_1, \ldots, x_4$ becomes true: AND$\{x_1, x_2,$ OR$\{x_3, x_4\}\}$). Thus, the MCSs are $\{x_1, x_2, x_3\}$ and $\{x_1, x_2, x_4\}$. Note that $\{x_1, x_2, x_3, x_4\}$ is also a cut set but not an MCS.

Definition 7 (Fussell-Vesely measure)**.** The Fussell-Vesely measure $I_x^{\mathrm{FV}}(t)$ of fault x at time t is the ratio of *i*) the sum of the probabilities $F_{\mathrm{sys}}^{i}(t)$ that MCS Γ_i has occurred by t (i.e., that all faults of Γ_i have occurred by t) for each MCS Γ_i containing x and *ii*) the sum of $F_{\mathrm{sys}}^{i}(t)$ for each MCS Γ_i:

$$I_x^{\mathrm{FV}}(t) = \sum_{i \,|\, x \in \Gamma_i} F_{\mathrm{sys}}^{i}(t) \,/\, \sum_{i} F_{\mathrm{sys}}^{i}(t) \quad \forall\, t. \qquad (2)$$

$F_{\mathrm{sys}}^{i}(t)$ is the system time-to-failure CDF given that only the faults contained in Γ_i can occur. According to this, $F_{\mathrm{sys}}^{i}(t)$ can be derived as the system time-to-failure CDF by removing the faults that do not belong to Γ_i and the related fault-to-failure propagations from the system failure logic.

2.3 FaultFlow workflow

The Use Case Diagram (UCD) of Fig. 1a illustrates the main functionalities of the FaultFlow API, exposed to the user as Java Standard Edition API. The Data Flow Diagram (DFD) of Fig. 1b describes the usual modeling and evaluation workflow supporting the Use Cases (UCs) of Fig. 1a.

- **UC1: CRUD model**: A metamodel instance can be either automatically derived from a JSON file encoding the BDD of the system structure and the SFT of the system failure logic (i.e., processes 1 and 2 in Fig. 1b) or manually created with a programmatic approach. Then, the model can be persisted onto the database.
- **UC2: Export STPN failure logic model to XPN**: Given a failure mode (not necessarily a top-level system failure mode), a FaultFlow metamodel instance can be converted into a Sirio metamodel instance (i.e., process 3), being an STPN modeling the failure logic. In turn, the STPN can be also exported as an XPN file, which can be solved by analysis or simulation through the GUI of the ORIS tool (process 4).
- **UC3: Evaluate dependability by Sirio**: The STPN of a failure mode can be analyzed through Sirio to derive the time-to-failure CDF (i.e., process 5).
- **UC4: Evaluate dependability by Pyramis**: Given a failure mode, a FaultFlow metamodel instance can be converted into a Pyramis metamodel instance (i.e., process 6), being an HSMP modeling the failure logic. In turn, the HSMP can be analyzed by Pyramis to derive the time-to-failure CDF (i.e., process 7).
- **UC5: Calculate importance measures by Pyramis**: The HSMP of a failure mode can be analyzed by Pyramis to derive the Birnbaum and Fussell-Vesely importance measures [71], [75] of faults (processes 8 and 10, respectively). This step requires to compute the MCSs, which can be derived from the metamodel instance and the specific failure mode (process 9).

5. Without loss of generality, we do not provide a formal definition of generalized CDF and generalized PDF of a discrete random variable.

3 FaultFlow syntax and semantics

In this section, we illustrate the abstract syntax (Section 3.1), semantics (Section 3.2), and concrete syntax (Section 3.3) of the FaultFlow library, discussing modeling and evaluation limitations, and exemplifying the described functionalities with a programmatic approach through the Java API.

3.1 Abstract syntax

Fig. 2 shows the metamodel of FaultFlow, which provides an *abstract* and *reusable* specification for component-based systems in a general perspective of knowledge. On the one hand, the metamodel supports the representation of the hierarchical structure of the system. Specifically, a system (represented by class `SystemType`) consists of a hierarchy of hardware/software components (class `ComponentType`). Each component is either a simple component, or a composite component made of subcomponents directly connected through physical or communication interfaces (class `CompositionPortType`). The entire system is modeled by the top-level component (i.e., the `ComponentType` instance represented by the attribute `topLevelComponentType` referenced by the `SystemType` instance).

On the other hand, the metamodel also supports representation of fault-to-failure and failure-to-fault propagations within the chain of threats to dependability of a system [5], exploiting suffix `Mode` for classes that model dependability concepts [5] to highlight abstractness and reusability of the provided system specification. Specifically, fault-to-failure propagations (class `ErrorMode`) model internal failure logic of components, from activation of faults (class `FaultMode`) to manifested failures (class `FailureMode`). A fault-to-failure propagation is defined by: *i*) a boolean expression (attribute `activationFunction` of class `ErrorMode`) defining which faults need to be active to cause propagation into errors and then to the failure; *ii*) a time-to-failure Probability Density Function (PDF) (attribute `faultToFailurePDF`),

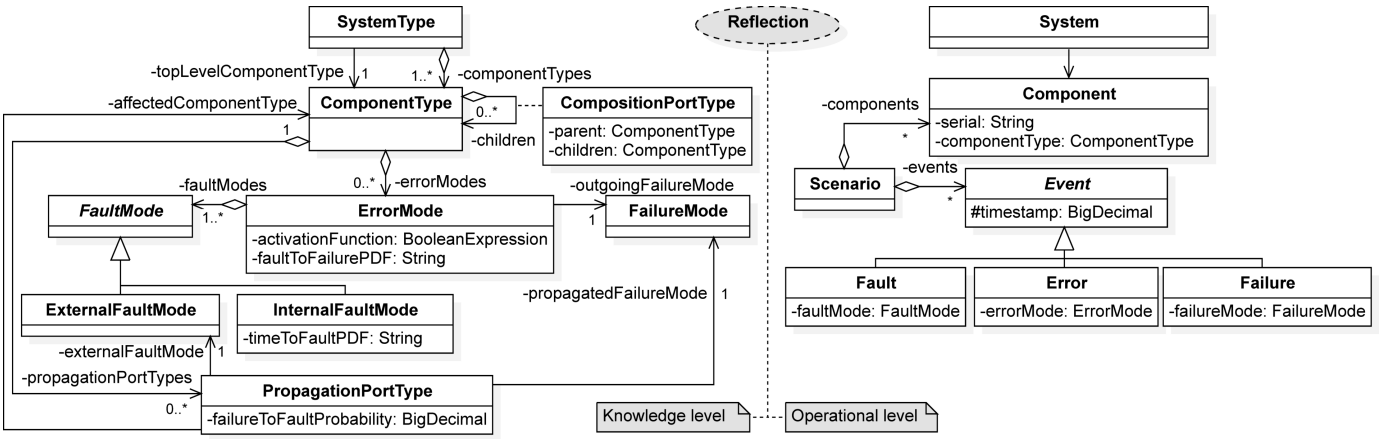

Fig. 2. UML class diagram of the metamodel of the FaultFlow library.

where the time-to-failure is intended as the duration elapsing from the activation of the boolean expression of faults to the failure occurrence. In turn, faults can be either internal faults (class `InternalFaultMode`) activated by interior processes of components and characterized by a time-to-fault PDF (attribute `timeToFaultPDF`), or external faults (class `ExternalFaultMode`) activated by exterior processes propagated from other components (at any hierarchy level). Conversely, failure-to-fault propagations (class `PropagationPortType`) characterize occurrence of failures of source components that act as external faults to other components, and are defined by the probability (attribute `failureToFaultProbability`) that a component failure (attribute `propagatedFailureMode`) affects another component (attribute `affectedComponent`) by activating one of its external faults (attribute `externalFaultMode`).

Moreover, the metamodel provides a stochastic characterization for durations of failure processes (attribute `timeToFaultPDF` of class `InternalFaultMode` and attribute `faultToFailurePDF` of class `ErrorMode`) by supporting any PDF in the class of exponomial functions [70].

Summarizing, three primary entities are defined: *i*) class `CompositionPortType`, modeling the system hierarchy in terms of hardware/software communications among components; *ii*) class `ErrorMode`, characterizing fault-to-failure propagations; and, *iii*) class `PropagationPortType`, characterizing failure-to-fault propagations. Fault-to-failure and failure-to-fault chains of threats either occur across physical and communication interfaces between components (modeled by class `CompositionPortType`), or are due to component-environment interactions affecting other components sharing the same environment (modeled by class `PropagationPortType`). Note that the metamodel represents failure propagation mechanisms with both branches and confluences, notably including multiple contributions of a single fault to the same failure through multiple propagation paths, so that the fault comprises a *repeated event*.

Note that classes at knowledge and operational levels model concepts independent of specific application domains, e.g., specific types of component. Thus, these classes are few and expected to change or increase in number only if the conceptual elements need to be changed or extended.

3.1.1 Running example

We consider a variant of the Gas Detection System (GDS) of [48], consisting of 3 gas detection sensors operating in a confined area of a petroleum installation, i.e., Gas Detectors A, B, and C (GDA, GDB, and GDC, respectively). The area can be logically divided in 2 parts: area X1, where GDA and GDB are located, close to each other (for redundancy); and, area X2 where GDC is located. In turn, each gas detector GDA, GDB, and GDC contains a physical unit to sense the gas, i.e., Initiators A, B, and C (IA, IB, and IC respectively). Fig. 3a shows the object diagram of the FaultFlow metamodel instance modeling the GDS structure. Each component is modeled by a `ComponentType` instance with the same name. Composition relations between components are modeled by a `CompositionPortType` instance. For example, the `ComponentType` instance `GDS` models the top-level component, the `ComponentType` instance `GDA` models the GDA, and the `CompositionPortType` instance `GDS_GDA_CPort` models the fact that the GDS contains the GDA.

The GDS of [48] is extended with a failure logic made of 12 internal faults (i.e., F1, ..., F12), modeling both fault-to-failure and failure-to-fault propagations across the GDS hierarchy. Specifically, each initiator IA, IB, and IC has 2 internal faults, each causing an immediate failure (e.g., IB_Failure1 occurs as soon as F5 or F6 has occurred). In turn, the failure of each initiator IA, IB, and IC acts as an external fault for the corresponding gas detector GDA, GDB, and GDC, respectively (e.g., IB_Failure1 acts as the external fault GDB_EF1 to GDB). Each gas detector also has 2 internal faults affecting its sensing capability. When its external fault and at least one of its internal faults have occurred, each gas detector then fails by missing to detect the gas (e.g., the fault propagation causing GDB_Failure1 is activated as soon as the external fault GDB_EF1 has occurred and the internal fault F7 or F8 has occurred). Detected gas values are computed based on values sensed within a bounded time of $24\,\mathrm{h}$, and the fault-to-failure propagation takes a non-negligible time (e.g., the fault propagation that causes GDB_Failure1 is not immediate). Finally, failure of each gas detector acts as an external fault for the GDS (e.g., GDB_Failure1 acts as the external fault GDS_EF2 to GDS), which fails as soon as both GDA

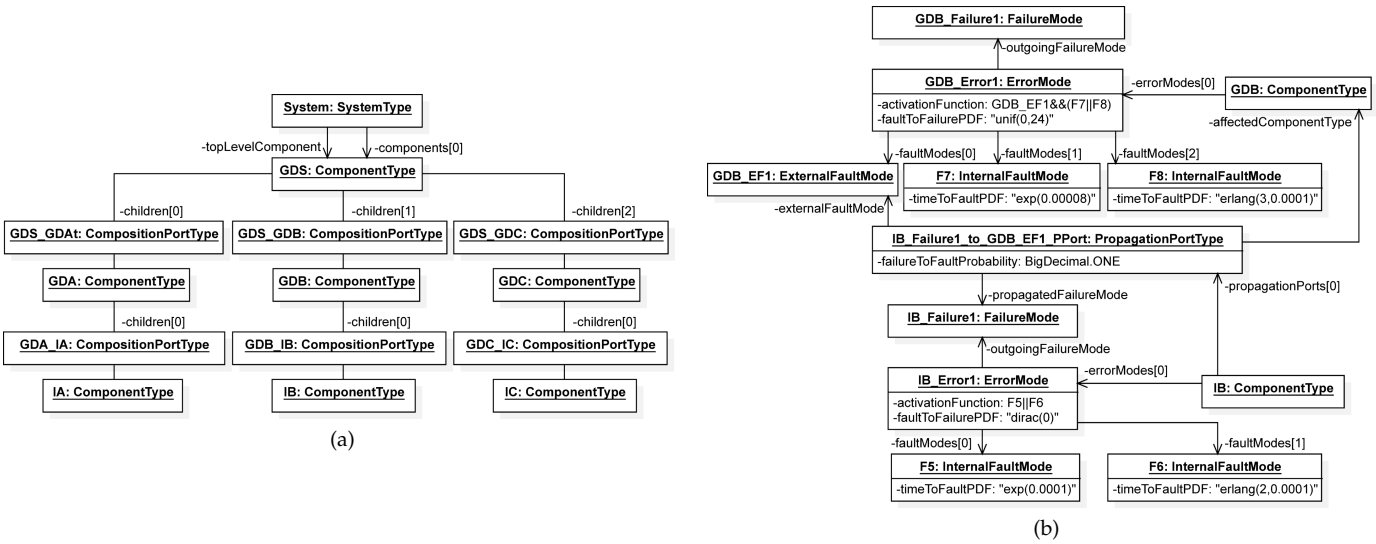

Fig. 3. Gas Detection System (GDS) [48] (times expressed in h): UML object diagram describing (a) the FaultFlow metamodel instance that represents the GDS structure, and (b) the FaultFlow metamodel instance that represents the failure logic yielding Failure1 of component GDB.

and GDB located in area X1 have failed, or GDC located in area X2 has failed (i.e., GDS_Failure1 occurs as soon as both the external faults GDS_EF1 and GDS_EF2 have occurred or the external fault GDS_EF3 has occurred). To facilitate the interpretability of results, the probability of a failure mode (i.e., probability that a failure will eventually occur given that its fault activation function has become true) is not modeled, though it could be accounted by the attribute `failureToFaultProbability` of the `PropagationPortType` instance that specifies such component failure as external fault of the component itself.

Stochastic parameters of the failure logic are selected as follows. For internal faults of initiators and gas detectors, we consider Exponential time-to-fault PDF, fitting expected values in the same order of magnitude as those reported in [48], and Erlang time-to-fault PDF, fitting also an arbitrary defined coefficient of variation. For fault-to-failure propagations of gas detectors, which propagate a fault into a failure within a maximum time of 24 h, we consider a uniform PDF over the min-max interval $[0, 24]$ h, as typically done in the literature on stochastic models by advocating a principle of maximum entropy [6]. The remaining fault-to-failure propagations occur immediately, and failures of initiators and gas detectors act as external faults with probability 1. Note that the approach can be easily tailored to any other specific statistics and approximant, since FaultFlow supports models with non-Markovian duration distributions in the class of exponomial functions, possibly with bounded support.

Fig. 3b shows the object diagram of the FaultFlow metamodel instance representing the failure logic that yields the failure of component GDB. Specifically, `GDB_Failure1` represents a failure of component GDB, which propagates when its external fault `GDB_EF1` has occurred and at least on of its internal faults `F7` and `F8` has occurred. In turn, external fault `GDB_EF1` propagates from failure `IB_Failure1` of component IB, which propagates when at least one of internal faults `F5` and `F6` of IB has occurred.

Listing 1 shows the Java code defining the metamodel instance specified by the object diagrams of Figs. 3a and 3b:

```
faultModes = new HashMap<>();
errorModes = new HashMap<>();
failureModes = new HashMap<>();
propagationPorts = new ArrayList<>();

// Definition of the SoS structure
System SoS = new System("SystemOfSystems");
Component GDS = new Component("GasDetectionSystem");
Component GDA = new Component("GasDetectorA");
Component GDB = new Component("GasDetectorB");
Component GDC = new Component("GasDetectorC");
Component IA = new Component("InitiatorA");
Component IB = new Component("InitiatorB");
Component IC = new Component("InitiatorC");

SoS.addComponent(GDS, GDA, GDB, GDC, IA, IB, IC);
SoS.setTopLevelComponent(GDS);

CompositionPort GDS_GDA_CPort = new CompositionPort(GDA, GDS);
CompositionPort GDS_GDB_CPort = new CompositionPort(GDB, GDS);
CompositionPort GDS_GDC_CPort = new CompositionPort(GDC, GDS);
GDS.addCompositionPorts(GDS_GDA_CPort,GDS_GDB_CPort,GDS_GDC_CPort);

CompositionPort GDA_IA_CPort = new CompositionPort(IA, GDA);
GDA.addCompositionPorts(GDA_IA_CPort);
CompositionPort GDB_IB_CPort = new CompositionPort(IB, GDB);
GDB.addCompositionPorts(GDB_IB_CPort);
CompositionPort GDC_IC_CPort = new CompositionPort(IC, GDC);
GDC.addCompositionPorts(GDC_IC_CPort);

// Definition of the failure logic characterizing GDB_Failure1
InternalFaultMode F5 = new InternalFaultMode("F5");
F5.setArisingPDF("exp(0.000001)");
InternalFaultMode F6 = new InternalFaultMode("F6");
F6.setArisingPDF("erlang(4,0.000005)");
InternalFaultMode F7= new InternalFaultMode("F7");
F7.setArisingPDF("exp(0.000002)");
InternalFaultMode F8 = new InternalFaultMode("F8");
F8.setArisingPDF("erlang(2,0.000003)");
faultModes.put(F5.getName(), F5);
faultModes.put(F6.getName(), F6);
faultModes.put(F7.getName(), F7);
faultModes.put(F8.getName(), F8);

ExternalFaultMode GDB_EF1 = new ExternalFaultMode("GDB_EF1");
faultModes.put(GDB_EF1.getName(), GDB_EF1);

FailureMode IB_Failure1 = new FailureMode("IB_Failure1");
ErrorMode IB_Error1 = new ErrorMode("IB_Error1");
IB_Error1.addInputFaultMode(F5, F6);
IB_Error1.setOutgoingFailure(IB_Failure1);
IB_Error1.setEnablingCondition("F5 || F6", faultModes);
IB_Error1.setPDF("dirac(0)");
errorModes.put(IB_Error1.getName(), IB_Error1);
failureModes.put(IB_Failure1.getDescription(), IB_Failure1);
IB.addErrorMode(IB_Error1);

FailureMode GDB_Failure1 = new FailureMode("GDB_Failure1");
ErrorMode GDB_Error1 = new ErrorMode("GDB_Error1");
GDB_Error1.addInputFaultMode(F7, F8, GDB_EF1);
GDB_Error1.setOutgoingFailure(GDB_Failure1);
GDB_Error1.setEnablingCondition("(F7 || F8) && GDB_EF1", faultModes);
GDB_Error1.setPDF("unif(0,6)");
errorModes.put(GDB_Error1.getName(), GDB_Error1);
failureModes.put(GDB_Failure1.getDescription(), GDB_Failure1);
GDB.addErrorMode(GDB_Error1);

PropagationPort IB_Failure1_to_GDB_EF1_PPort =
    new PropagationPort(IB_Failure1, GDB_EF1, GDB);
IB.addPropagationPort();
```

Listing 1. Java code snippet defining the FaultFlow metamodel instance illustrated by the UML object diagrams of Figs. 3a and 3b.

3.2 Semantics

The FaultFlow semantics is defined through translation of metamodel instances into STPNs [73], a formal model of concurrent timed systems with stochastic temporal parameters and discrete probabilistic choices. In STPNs, transitions (depicted as bars) model stochastic durations of activities, tokens (depicted as dots) within places (depicted as circles) model the discrete logical state of the system, and directed arcs (depicted as directed arrows) from input places to transitions and from transitions to output places model precedence relations among activities. A transition is enabled by a marking (i.e., an assignment of tokens to places) if each of its input places contains at least one token and if its enabling function evaluates to true. Upon enabling, each transition samples a time-to-fire from its PDF, which can be EXP, GEN, or the generalized PDF of a Dirac Delta function. In the latter case, the transition time-to-fire takes a deterministic value τ, and the transition is termed deterministic (DET) if $\tau \neq 0$ and immediate (IMM) if $\tau = 0$. The transition with minimum time-to-fire is selected to fire, removing one token from each of its input places and adding one token to each of its output places. Ties among transitions with equal time-to-fire are solved by a random switch determined by probabilistic weights associated with transitions.

To translate metamodel instances into STPNs, a bottom-up visit of metamodel instances is performed, starting from the `InternalFaultMode` instances. Two transformation rules R1 and R2 are applied, concerning modeling of fault-to-failure and failure-to-fault propagations, respectively. In particular, R1 maps each time-to-fault PDF and each fault-to-failure PDF into a transition with the same PDF:

- **R1**. Each `InternalFaultMode` instance is translated into a transition with PDF equal to the attribute `timeToFaultPDF`, input place modeling activation of the internal process yielding the fault, and output place modeling the fault occurrence, e.g., in Fig. 5a, place `F5` models activation of the internal process leading to fault F5 of the IA, transition `F5_Occurrence` models the fault process duration, and place `F5_Occurred` models the fault occurrence.
  Each `ErrorMode` instance is translated into a transition with PDF equal to the attribute `timeToFailurePDF`, and enabling function derived from the attribute `activationFunction` by replacing each fault name with the name of the place modeling the fault occurrence, i.e., the enabling function is a constraint on token counts in places representing occurrence of the `FaultMode` instances referenced by the `ErrorMode` instance. Moreover, the transition has an input place modeling activation of the fault-to-failure process as soon as the enabling function becomes true, and an output place modeling occurrence of the failure represented by the `FailureMode` instance referenced by the `ErrorMode` instance. For example, in Fig. 5a, transition `IB_Failure1_Occurrence` models duration of propagation from faults F5 and F6 to failure Failure1 of IB: its enabling function is `F5_Occurred || F6_Occurred`; its input place `IB_Error1` models activation of the fault-to-failure propagation as soon as `F5_Occurrence` or `F6_Occurrence` has fired; its output place `IB_Failure1_Occurred` models IB_Failure1 occurrence.
- **R2**. The `PropagationPortType` instances referencing the same `FailureMode` instance are collectively translated into an IMM transition t having the place modeling the failure occurrence as input place, and an output place for each `PropagationPortType` instance `i`, e.g., the `PropagationPort` instance `IB_Failure1_to_GDB_EF1_PPort` in Fig. 3b is translated into the IMM transition `IB_Failure1_to_GDB_EF1` in Fig. 5a, whose input place is `IB_Failure1_Occurred` modeling occurrence of IB_Failure1. For each `PropagationPortType` instance `i`, if the attribute `failureToFaultProbability` p_i is equal to 1, the output place of t is the place modeling occurrence of the `ExternalFaultMode` instance referenced by the `PropagationPortType` instance `i`, e.g., the output place of `IB_Failure1_to_GDB_EF1` is `GDB_EF1_Occurred` representing occurrence of the external fault GDB_EF1 propagated from IB_Failure1.
  Otherwise, if the attribute `failureToFaultProbability` p_i is lower than 1 (see Fig. 4), then the output place of t (i.e., place `IB_Failure1_to_GDB_EF1_router`) is the input place of two IMM transitions t_{ia} and t_{ib} (i.e., transitions `IB_Failure1_PROPAGATED_to_GDB_EF1` and `IB_Failure1_NOT_PROPAGATED_to_GDB_EF1`, respectively) with weight p_i and $1 - p_i$, respectively, where the output place of t_{ia} models the occurrence of the external fault and t_{ib} has no output place. Note that, to guarantee that the propability of propagating the failure into each external fault EF_i is actually equal to p_i, the IMM transitions t_{ia} and t_{ib} associated with each `PropagationPortType` instance `i` have different priority than the IMM transitions associated with any other `PropagationPortType` instance.

The FaulFlow API implements translation of metamodel instances into STPN models according to two modes: in mode `PetriNetExportMethod.FAULT_ANALYSIS`, the STPN represents concurrent execution of all fault propagation processes, i.e., each place modeling activation of the process leading to an internal fault contains one token (and each place modeling activation of a fault-to-failure propagation contains one token); in mode `PetriNetExportMethod.FAULT_INJECTION`, the STPN represents concurrent execution of selected fault propagation processes, i.e., each place that models activation of the process leading to a selected internal fault contains one token (and each

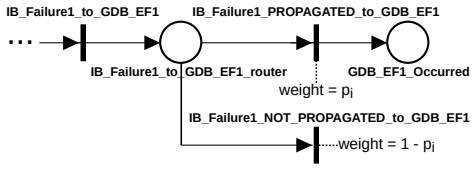

Fig. 4. Fragment of the STPN of Fig. 5a if the attribute `routingProbability` of `IB_Failure1_to_GDB-_EF1_PPort` in Fig. 3b is $p_i < 1$.

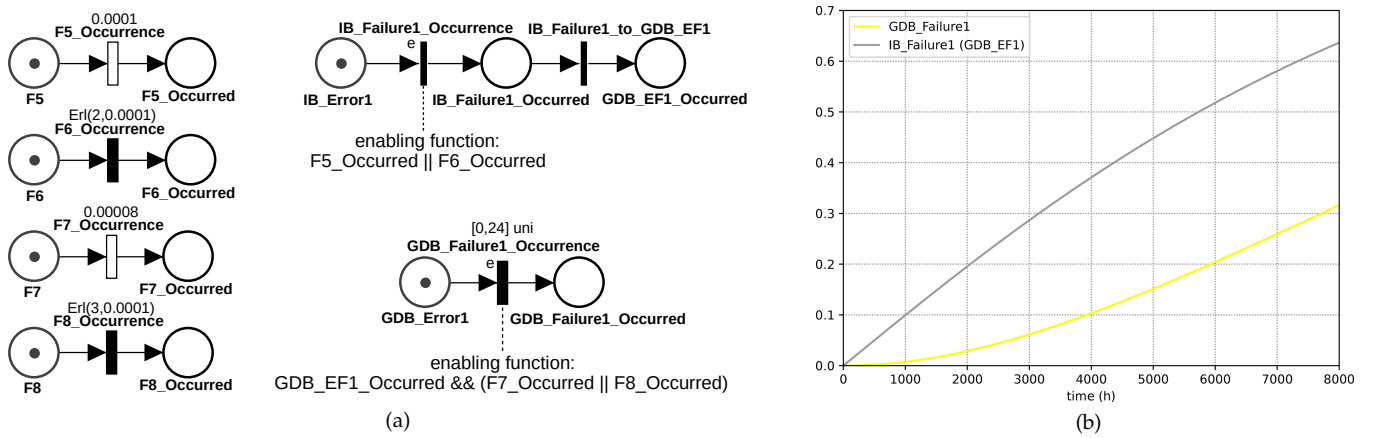

Fig. 5. GDS of Fig. 3 (times expressed in h): (a) STPN model of the failure logic yielding Failure1 of component GDB; and, (b) CDFs of the duration of the failure processes that determine Failure1 of component IB and Failure1 of component GDB.

place modeling activation of a fault-to-failure propagation contains one token) and each transition modeling the occurrence of a selected fault may take a selected deterministic value, thus supporting representation of injected faults.

FaultFlow exploits Sirio to evaluate the numerical form of the time-to-failure CDF $F_y^{\text{fail}}(t)$ of a given failure y, as specified in Definition 4, i.e., $F_y^{\text{fail}}(t)$ is the CDF of the duration elapsing from the initial time to the occurrence of failure y. In particular, once the FaultFlow metamodel instance is translated into an STPN, encoded as an instance of the metamodel of the Sirio library, $F_y^{\text{fail}}(t)$ can be derived as the transient probability of the marking m that assigns one token to the place representing the failure occurrence, by performing regenerative transient analysis [33] of the STPN with a stop condition equal to m. The analysis can be performed either through the Sirio library, or through the ORIS GUI [51] if exported as an XPN file. For example, Fig. 5b shows the CDFs of the duration of the failure processes that determine IB_Failure1 and GDB_Failure1, computed in nearly $1.31\,\text{s}$ and $60.63\,\text{s}$,[6] respectively, by performing the analysis with time step $2\,\text{h}$ and time limit $8000\,\text{h}$, i.e., the time by which a maintenance of the GDB is scheduled and the component can be considered as new [48]. Note that transient analysis of a single flat STPN, modeling the entire chain of threats leading to a failure, may suffer the curse of dimensionality when the failure logic model includes multiple concurrent durations with GEN distribution. This limitation can be overcome by separately analyzing different combinations of fault propagations (by exploiting the `PetriNetExportMethod.FAULT-_INJECTION` mode for STPN generation) or, if the failure logic model does not include repeated events at any level, by exploiting the Pyramis library [13] (see Section 4).

Listing 2 shows the Java code that translates the metamodel instance created by Listing 1 into an STPN, and analyzes it through to derive the time-to-failure CDFs of IB_Failure1 and GDB_Failure1, shown in Fig. 5b:

```
// Conversion of the failure logic model to STPN
System SoS = GasDetectionSystemBuilder.getInstance().getSystem();
String failureName = "GDB_Failure1";
PetriNetTranslator pnt = PetriNetExporter.exportPetriNetFromSystem(
    SoS, PetriNetExportMethod.FAULT_ANALYSIS);
PetriNetReducer petriNetReducer = new PetriNetReducer(
    pnt.getPetriNet(), pnt.getMarking());
petriNetReducer.reduce(failureName,
    GasDetectionSystemBuilder.getPropagationPorts(),
        GasDetectionSystemBuilder.getErrorModes());

// Analysis of the failure logic STPN
PetriNetAnalyzer petriNetAnalyzer = new PetriNetAnalyzer(
    petriNetReducer.getPetriNet(), petriNetReducer.getMarking());
String externalFaultMode = "GDB_EF1";
BigDecimal timeStep = new BigDecimal(2);
BigDecimal timeLimit = new BigDecimal(8000);
BigDecimal error = BigDecimal.ZERO;
TransientSolution<DeterministicEnablingState, RewardRate> rewards =
    petriNetAnalyzer.regenerativeTransient(
        externalFaultMode, timeLimit, timeStep, error);
```

Listing 2. Java code snippet that analyzes the FaultFlow metamodel instance created by Listing 1 to derive the time-to-failure CDFs of IB_Failure1 of component IB and GDB_Failure1 (shown in Fig. 5b).

6. The experiments of Sections 3 and 4 have been executed on a single core of an Intel(R) Xeon(R) Gold 5120 CPU 2.20 GHz with 32.0 GB RAM.

3.3 Concrete syntax

Concrete instances of the FaultFlow metamodel can be automatically derived from SysML BDDs [50], modeling the system decomposition into components, and from SFTs [31], [37], modeling fault-to-failure and failure-to-fault propagations, thus mitigating the effort demanded to domain experts by automatically initializing executable object instances of the metamodel. Specifically, the fragment of concrete instance that represents the system structure can be derived from a BDD by mapping each block to a `Component` instance and each relation between blocks to a `CompositionPortType` instance, with the root block corresponding to the `ComponentType` instance representing the entire system, i.e., the attribute `topLevelComponentType` referenced by the `SystemType` instance. For example, Fig. 6a shows the BDD modeling the structure of the GDS specified by the metamodel instance of Fig. 3a.

The fragment of concrete instance representing the system failure logic can be derived from an SFT where nodes represent events occurring after a stochastic delay characterized by a time-to-occurrence PDF, with leaf nodes modeling internal faults of components, logical gates (i.e., AND, OR, KoN) representing fault-to-failure propagations, and propagation nodes modeling failure-to-fault propagations occurring with a given probability. Specifically, the M2M transformation is performed as follows (see the SFT of Fig. 6b representing the failure logic of the GDS specified by the metamodel instance of Fig. 3b):

- Each leaf node is mapped to an `InternalFaultMode` instance where the attribute `timeToFault-`

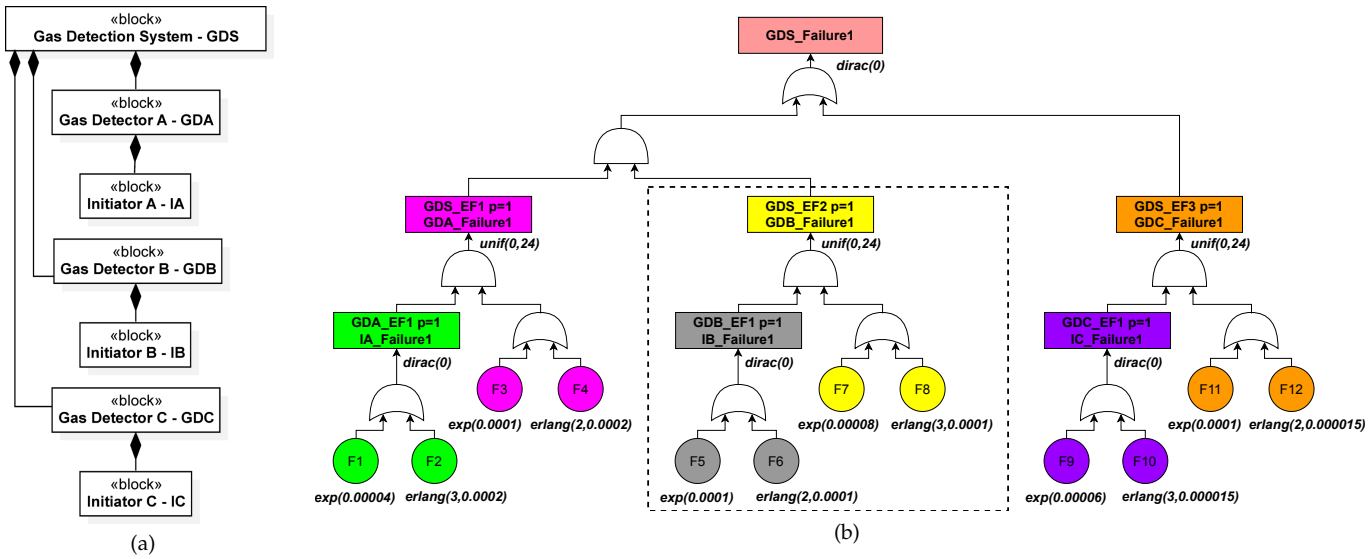

Fig. 6. GDS in Figs. 3 and 6 (times expressed in h): (a) SysML Block Definition Diagram (BDD) defining the structure of the entire GDS (which can be automatically translated into the object diagram of Fig. 3a); and, (b) Stochastic Fault Tree (SFT) defining the failure logic of the entire GDS, where the sub-SFT within the box with dashed outline can be automatically translated into the object diagram of Fig. 3b, and different colors are used to highlight internal faults and failures of different components (e.g., light gray for IB and dark gray for GDB).

`PDF` is the time-to-occurrence PDF associated with the node, e.g., leaf nodes F5 and F6 in Fig. 6b are mapped to instances `F5` and `F6` in Fig. 3b.

- Each propagationno node n is translated into:
  - an `ErrorMode` instance, whose attribute `activationFunction` is the boolean expression defined by the subtree having node n as root and the successor nodes of n as leaves, and the attribute `faultToFailurePDF` is the time-to-occurrence PDF of node n;
  - a `FailureMode` instance referenced by the above mentioned `ErrorMode` instance.

  For instance, node IB_Failure1 in Fig. 6b is mapped to instances `IB_Error1` and `IB_Failure1` in Fig. 3b.
- Each non-root node m is mapped to an `ExternalFaultMode` instance and to a `PropagationPortType` instance referencing the `ExternalFaultMode` instance and the `FailureMode` instance also associated with m, e.g., node IB_Failure1 in Fig. 6b is mapped also to instances `GDB_EF1` and `IB_Failure1_to_GDB_EF1_PPort` in Fig. 3b.

4 Integration with the Pyramis Library

In this section, we discuss syntax and semantics of the Pyramis Java library (Section 4.1) and its integration with FaultFlow by a M2M transformation (Section 4.2), proving that the Pyramis semantics can be expressed by a variant of the set of STPN models defining the FaultFlow semantics. Then, we derive the time-to-failure CDF as well as importance measures of faults (Section 4.3). The described functionalities are exemplified with a programmatic approach through the Java API of FaultFlow and Pyramis.

4.1 Pyramis syntax and semantics

Pyramis supports modeling and evaluation of stochastic systems specified by Hierarchical Semi-Markov Processes with parallel regions (HSMPs) [13], an extension of UML statecharts capturing concurrency, hierarchy, stochastic timing, and probabilistic choices. Fig. 7 shows a refactoring of the Pyramis metamodel [13] aimed to improve its programmatic use and facilitate the FaultFlow-to-Pyramis transformation. Specifically, an HSMP (represented by class `HSMP`) consists of locations (class `LogicalLocation`) which can be either steps (class `Step`), i.e., activities with stochastic duration, or final locations (class `FinalLocation`)i.e., terminated activities. Steps have a weight, used to select the next location by a random switch when multiple steps terminate at the same time (limit condition occurring only when steps have deterministic duration). Steps can be either simple (class `SimpleStep`), modeling atomic activities with duration PDF, or composite (class `CompositeStep`), made of concurrent regions (class `Region`). Regions can be of type either `ENDING` (i.e., terminating in a final location) or `NEVERENDING` (i.e., not terminating), and can be combined in composite step of type either `FIRST` (i.e., terminating as soon as any of its regions terminates) or `LAST` (i.e., terminat-

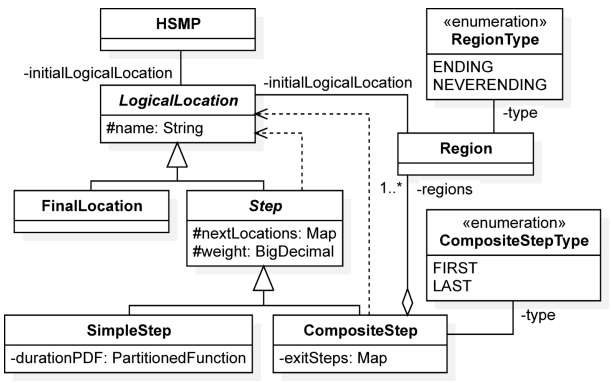

Fig. 7. UML class diagram of the metamodel of the Pyramis library.

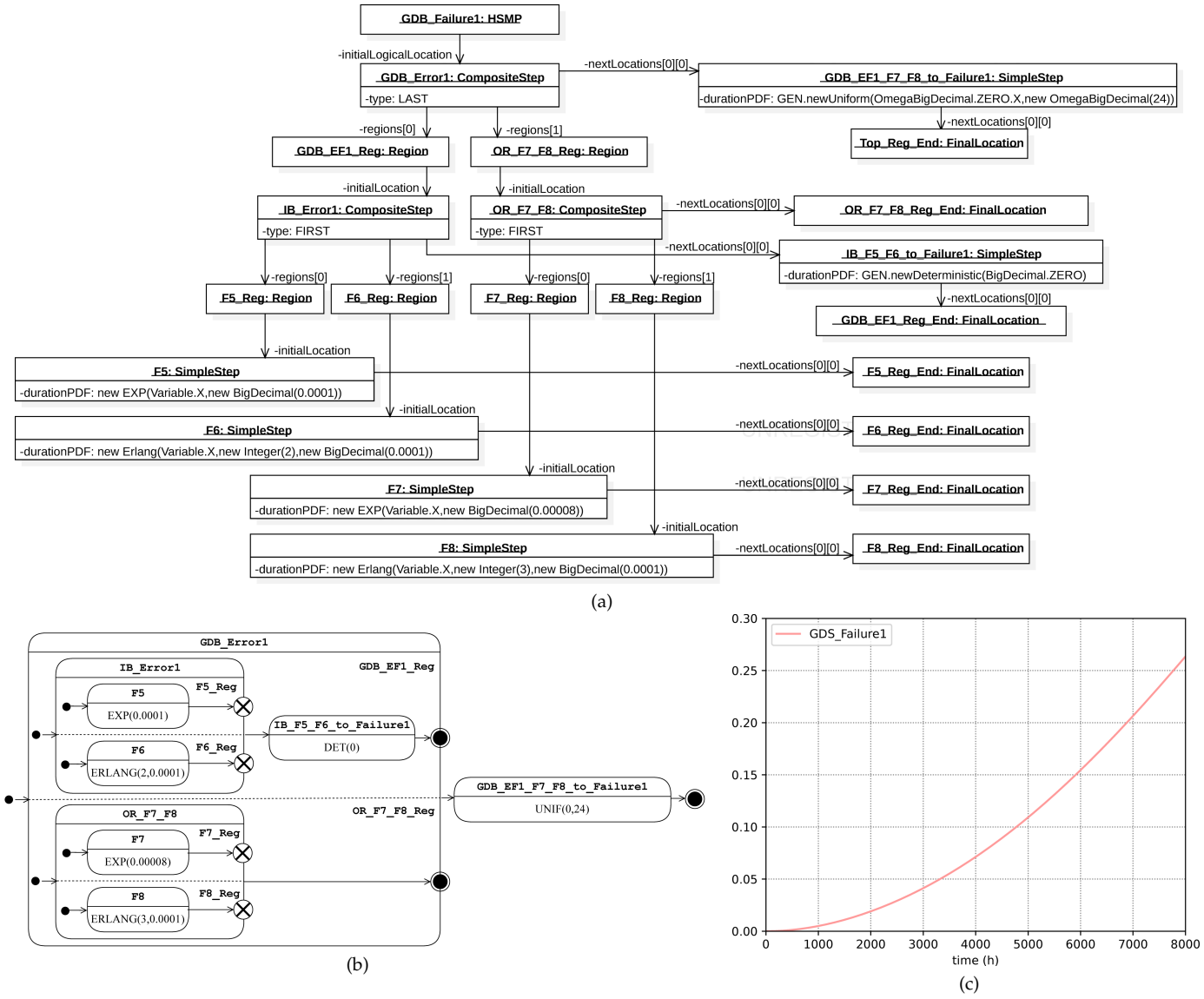

Fig. 8. GDS of Figs. 3 and 6 (times expressed in h): (a) UML object diagram describing the Pyramis metamodel instance that represents the GDS failure logic, which can be automatically derived from the FaultFlow metamodel instance specifying the GDS structure (as shown in Fig. 3a) and failure logic (not shown in Fig. 3b due to space limits, corresponding to the SFT of Fig. 6b); (b) HSMP model of the failure logic yielding Failure1 of component GDB (which compares with the STPN model of Fig. 5a); (c) CDF of the duration of the failure process of the GDS.

ing as soon as all its regions have terminated). Composition of regions and steps yields a hierarchy of HSMPs where each composite step is defined by an HSMP at the next lower level, the top-level HSMP contains a single region, and non-top-level HSMPs contain at least one region of type ENDING.

Fig. 8b shows the HSMP obtained from the FaultFlow metamodel instance of Fig. 3b, in turn obtained from the SSFT in the box with dashed outline in Fig. 6b. Steps are depicted as rounded boxes (e.g., simple step F5 modeling fault F5 of component IB); regions of composite steps are separated by dashed lines (e.g., regions of step IB_Error1, modeling the activation leading to Failure1 of component IB); the initial steps of a region are denoted by directed arcs connecting a black-filled circle to the box of the step (e.g., F5 is the initial step of region F5_Reg); the final location of a region of type ENDING is depicted as an unfilled circle placed on the border of the region, with an X–shaped cross inside if the composite step is of type FIRST (e.g., the final location of region F5_Reg) or with a smaller black-filled circle inside if the composite step is of type LAST (e.g., the final location of region GDB_EF1_Reg); transitions between locations are depicted as directed arcs.

The state of an HSMP collects an active location for each region (of the composite step modeling the HSPM), a time to the next event (TNE) for each active step (i.e., the time to the step completion), and a state for each active composite step. In the initial state of the HSMP of Fig. 8b, steps F5, F6, F7, and F8 are active and sample a TNE from their duration PDF. If F5 or F6 terminates before F7 and F8, then IB_F5_F6_to_Failure1 becomes the active step of region GDB_EF1_Reg. Conversely (i.e., F7 or F8 terminates before F5 and F6), GDB_Error1 terminates and GDB_EF1_F7_F8_-to_Failure1 becomes the active step of the top-level region, until its completion terminates the HSMP execution.

4.2 FaultFlow-to-Pyramis transformation

Instances of the Pyramis metamodel can be derived from instances of the FaultFlow metamodel, provided that the system failure logic is defined by an SSFT that: i) does not include repeated events at any level; ii) contains gates of type AND and OR. While repeated events cannot be modeled due to a structural limitation of both the HSMP formalism and the related solution technique, VOT(k/N) gates could be managed by the M2M transformation by explicitly modeling the $\binom{K}{N}$ combinations of events that lead to the VOT(k/N) event. Specifically, VOT(k/N) can be expressed as the disjunction (OR) of the conjunction (AND) of all subsets of k over N inputs, e.g., VOT(2/3) with inputs x_1, x_2, and x_3 is equivalent to OR{AND{x_1, x_2}, AND{x_1, x_3}, AND{x_2, x_3}}. Based on the transformation rule R3 illustrated in the following, VOT(k/N) can be translated into a composite step of type `FIRST` with $\binom{k}{N}$ regions, each containing a composite step of type `LAST` with k regions (modeling a different combination of k out of N events).

In the implemented M2M transformation, a top-down visit of the FaultFlow metamodel instance is performed starting from the `FailureMode` instance representing the top-level failure, by applying two transformation rules R3 and R4 which concern representation of fault-to-failure and failure-to-fault propagations, respectively. In particular, R3 maps each time-to-fault PDF and each fault-to-failure PDF into a simple step with the same duration PDF:

- **R3**. Each `ErrorMode` instance is mapped to a `CompositeStep` instance derived from attribute `activationFunction`, whose next location is a `SimpleStep` instance with attribute `durationPDF` equal to attribute `faultToFailurePDF`. In turn, the activation function is decomposed into atomic propositions on faults: AND and OR operators are repeatedly mapped to nested `CompositeStep` instances of type `LAST` and `FIRST`, respectively, with a number of `Region` instances equal to the number of children of the boolean logic operator; atomic propositions associated with `InternalFaultMode` instances are translated into `SimpleStep` instances with attribute `durationPDF` equal to attribute `timeToFaultPDF` of `InternalFaultMode`; and atomic propositions associated with `ExternalFaultMode` instances are translated by applying rule R4.
  For example, the `ErrorMode` instance `GDB_Error1` of Fig. 3b is translated into the namesake `CompositeStep` instance of Fig. 8a (also shown in Fig. 8b), which is of type `LAST` and contains two `Region` instances, i.e., `GDB_EF1_Reg` and `OR_F7_F8_Reg`, given that the attribute `activationFunction` of `GDB_Error1` is a boolean logic formula connecting two sub-formulas through the OR operator.
- **R4**. Each `PropagationPort` instance references: *i*) an `ExternalFaultMode` instance; *ii*) a `FailureMode` instance referencing an `ErrorMode` instance. If attribute `failureToFaultProbability` of the `PropagationPort` instance has value $p = 1$, then rule R3 is applied to the `ErrorMode` instance, yielding a sequence made of a `CompositeStep` instance and a `SimpleStep` instance whose next location is the `FinalLocation` of its region. For example, the `PropagationPort` instance `IB_Failure1_to_GDB_EF1_PPort` in Fig. 3b is mapped in Fig. 8a to the `CompositeStep` instance `IB_Error1`, `SimpleStep` instance `IB_F5_F6_to_Failure1`, and final location of region `GDB_EF1_Reg`.
  Conversely, as shown in Fig. 9, if attribute `failureToFaultProbability` has value $p < 1$, then the next location of the `SimpleStep` instance (i.e., `IB_Failure1_PROPAGATED_to_GDB_EF1` in Fig. 9) is: *i*) the `FinalLocation` instance of its region with probability p; *ii*) an absorbing step (i.e., whose next location is itself) with probability $1-p$ (i.e., `IB_Failure1_NOT_PROPAGATED_to_GDB_EF1` in Fig. 9).

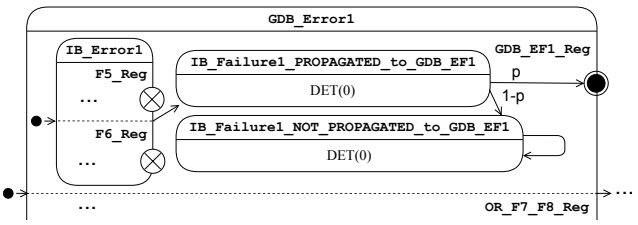

Fig. 9. Fragment of the HSMP of Fig. 8b if the attribute `routingProbability` of `IB_Failure1_to_GDB-_EF1_PPort` in Fig. 3b is $p_i < 1$.

Theorem 1 proves the soundness of this M2M transformation, showing that, for *any* failure, the time-to-failure CDF computed by analyzing the STPN encoding the Pyramis metamodel instance is equal to the time-to-failure CDF computed by analyzing the STPN encoding the FaultFlow metamodel instance. As a by-product, the proof shows that the Pyramis semantics can be expressed by a variant of the set of STPN models that defines the FaultFlow semantics.

Theorem 1. *If a FaultFlow metamodel instance is defined by an SSFT with no repeated event at any hierarchy level, then, for any failure, the time-to-failure CDF computed by analyzing the STPN encoding the FaultFlow metamodel instance is equal to the time-to-failure CDF computed by analyzing the STPN encoding the corresponding Pyramis metamodel instance (**Proposition 1**). Otherwise, the FaultFlow metamodel instance cannot be represented by a Pyramis metamodel instance (**Proposition 2**).*

4.3 Time-to-failure CDF and importance measures

FaultFlow exploits Pyramis to evaluate the numerical form of the time-to-failure CDF $F_y^{\text{fail}}(t)$ of a given failure y, as specified in Definition 4, i.e., $F_y^{\text{fail}}(t)$ is the CDF of the duration elapsing from the initial time to the occurrence of failure y. In particular, once the FaultFlow metamodel instance is translated into an HSMP, encoded as an instance of the metamodel of the Pyramis library, $F_y^{\text{fail}}(t)$ can be derived through the solution technique implemented by Pyramis. This method performs transient analysis until absorption of the HSMP, deriving the CDF of the time to reach each step of the failure process, and thus also the time-to-failure CDF $F_y^{\text{fail}}(t)$. Efficiency of computation is achieved through a bottom-up procedure that, starting from the lowest-level

HSMPs, separately evaluates the duration CDF of the Semi-Markov Process (SMP) underlying each region, and combines them to derive the duration CDF of the HSMP at the next higher level. Fig. 8c shows the duration CDF of the failure process that determines GDS_Failure1 in Fig. 6, i.e., the top-level failure, computed with time step $2\,\mathrm{h}$ and time limit $8000\,\mathrm{h}$ in nearly $16.69\,\mathrm{s}$, which, as expected, is lower than the time needed to compute the duration CDF of the failure process that determines the lower-level failure GDB_Failure1 using Sirio (i.e., $60.63\,\mathrm{s}$, see Section 3.2).

Listing 3 shows the Java code to translate the FaultFlow metamodel instance created by Listing 1 into an instance of the Pyramis metamodel, and to analyze it to derive the CDF of the duration of the GDS failure process (shown in Fig. 8c):

```
System SoS = GasDetectionSystemBuilder.getInstance().getSystem();
double timeStep = 2;
double timeLimit = 8000;
TreeParser treeParser = new TreeParser(SoS);
HSMP hsmp = HSMPParser.parseTree(treeParser.createTree(
    GasDetectionSystemBuilder.getErrorMode("GDS_Failure1")));
HierarchicalSMPAnalysis analysis = new HierarchicalSMPAnalysis(hsmp, 0);
analysis.evaluate(timeStep, timeLimit);
```

Listing 3. Java code snippet that translates the FaultFlow metamodel instance created by Listing 1 into an instance of the Pyramis metamodel and then analyzes it (through the Pyramis library) to derive the CDF of the duration of the failure process of the GDS (shown in Fig. 8c).

FaultFlow uses Pyramis also to calculate the importance measures of faults, as specified by Definitions 5 to 7. Specifically, to compute the Birnbaum measure of fault x, the system time-to-failure CDF given that x has occurred by time t or not, i.e., $F^{\mathrm{fail}}_{\mathrm{sys}}(t)_{|F^{\mathrm{fault}}_x(t)=1}$ and $F^{\mathrm{fail}}_{\mathrm{sys}}(t)_{|F^{\mathrm{fault}}_x(t)=0}$, respectively, can be derived as follows: *i*) two SSFTs are obtained from the system SSFT by assuming that fault x has occurred at the initial time (i.e., by replacing the time-to-fault PDF $f^{\mathrm{fault}}_x(t)$ with a Dirac Delta function centered at $t=0$) and by assuming that fault x will never occur (i.e., by removing fault x and the related fault-to-failure propagations from the system SSFT), respectively; *ii*) according to the data flow diagram of Fig. 1b, the obtained SSFTs (and the system BDD) are translated first into two FaultFlow metamodel instances and then into two HSMPs, encoded as instances of the Pyramis metamodel, which are analyzed by the Pyramis solution technique to derive the system time-to-failure CDF (as discussed at the beginning of this subsection), yielding $F^{\mathrm{fail}}_{\mathrm{sys}}(t)_{|F^{\mathrm{fault}}_x(t)=1}$ and $F^{\mathrm{fail}}_{\mathrm{sys}}(t)_{|F^{\mathrm{fault}}_x(t)=0}$, respectively.

Similarly, to compute the Fussell-Vesely measure of fault x, the probability that MCS Γ_i has occurred by time t, i.e., $F^{i}_{\mathrm{sys}}(t)$, can be derived as follows: *i*) an SSFT is derived from the system SSFT by removing the faults that do not belong to Γ_i and the related fault-to-failure propagations; *ii*) the obtained SSFT (and the system BDD) are translated first into a FaultFlow metamodel instance and then into an HSMP, encoded as instance of the Pyramis metamodel, which is analyzed by the Pyramis solution method to derive the system time-to-failure CDF (as discussed at the beginning of this subsection), yielding $F^{i}_{\mathrm{sys}}(t)$.

TABLE 1
Minimal cut sets of the GDS of Fig. 3, consisting of 2 or 4 faults.

F10, F11	F10, F12	F11, F9	F12, F9
F1, F3, F5, F7	F1, F4, F5, F7	F2, F3, F5, F7	F2, F4, F5, F7
F1, F3, F5, F8	F1, F4, F5, F8	F2, F3, F5, F8	F2, F4, F5, F8
F1, F3, F6, F7	F1, F4, F6, F7	F2, F3, F6, F7	F2, F4, F6, F7
F1, F3, F6, F8	F1, F4, F6, F8	F2, F3, F6, F8	F2, F4, F6, F8

To show that the ranking of faults may vary over time, Figs. 10a and 10b plot the importance measures of the internal faults of the GDS of Fig. 6 far beyond the time limit of $8000\,\mathrm{h}$, i.e., up to time $60\,000\,\mathrm{h}$, with time step $4\,\mathrm{h}$. Evaluation takes $43.05\,\mathrm{min}$ for the Birnbaum measures and $19.38\,\mathrm{min}$ for the Fussel-Vesely measures (evaluation with time limit $8000\,\mathrm{h}$ and time step $2\,\mathrm{h}$ would require just $5.46\,\mathrm{min}$ for the Birnbaum measures and $2.44\,\mathrm{min}$ for the Fussel-Vesely measures). Note that derivation of the Birnbaum measures requires to compute, for each fault, the time-to-failure CDF of the GDS twice, once assuming that the fault has occurred and once that it has not, thus analyzing 24 HSMPs with depth 5 containing 12 simple steps modeling the internal faults F1, . . . , F12, and 7 simple steps modeling fault-to-failure propagations. Conversely, derivation of the Fussel-Vesely measures requires to compute the time-to-failure CDF of all MCS, shown in Table 1, thus analyzing 4 HSMPs with depth 5 containing 2 simple steps modeling internal faults and 3 simple steps modeling fault-to-failure propagations, and 16 HSMPs with depth 5 containing 4 simple steps modeling internal faults and 5 simple steps modeling fault-to-failure propagations. Thus, for our example, evaluation of the Birnbaum measures is more computationally intensive than that of the Fussel-Vesely measures, which occurs also in general, unless the number of MCSs is significantly larger than the that of internal faults.

In the first 10–20 thousand hours, the Birnbaum measure is larger for internal faults with lower depth in the SSFT of Fig. 6b and, for each pair of internal faults of the same component (children of the same OR gate), in most cases the Birnbaum measure is larger for the component with larger mean time-to-fault, e.g., $I^B_{F9}(t) > I^B_{F10}(t)\ \forall\ t$. The Fussel-Vesely measure also ranks faults F9 and F11 as the most important ones, at least until around 20–22 thousand hours. Such result is compatible with the GDS failure logic in Fig. 6b. In fact, F9–F10 and F11–F12 are connected to two OR gates, whose combined activation causes the system failure. Moreover, F9 and F11 have a mean time-to-fault that is shorter than the one of the other internal fault connected to the same OR gate (i.e., F10 and F12, respectively), and each of them is part of two MCSs containing only 2 faults (see Table 1). Furthermore, one of those MCS contains exactly these two faults, which are then enough to make the system fail. Conversely, counter-intuitively, the Fussel-Vesely measure of faults F10 and F12 is lower than that of all the other faults. This behavior can be ascribed to the fact that the otdetermined by each time-to-fault PDF and each fault-to-failure PDF in the model not just their mean values or moments (as well as by the SSFT structure and the failure-to-fault probabilities)in the implementedher faults are part of a larger number of MCSs, though composed of 4 faults rather than 2. Also, due to their occurrence distributions, the other faults connected to the OR gate (which are exactly F9 and F11, respectively) are most likely to have occurred before, and thus the importance of F10 and F12 in causing the system failure is reduced. Again, note that the ranking of faults may change over time, highlighting the significance of the evaluated measures, and the fact that thery are determined by the entire time-to-fault and fault-to-failure PDFs (as well as by the SSFT structure and the failure-to-fault probabilities) not just by their mean values or moments.

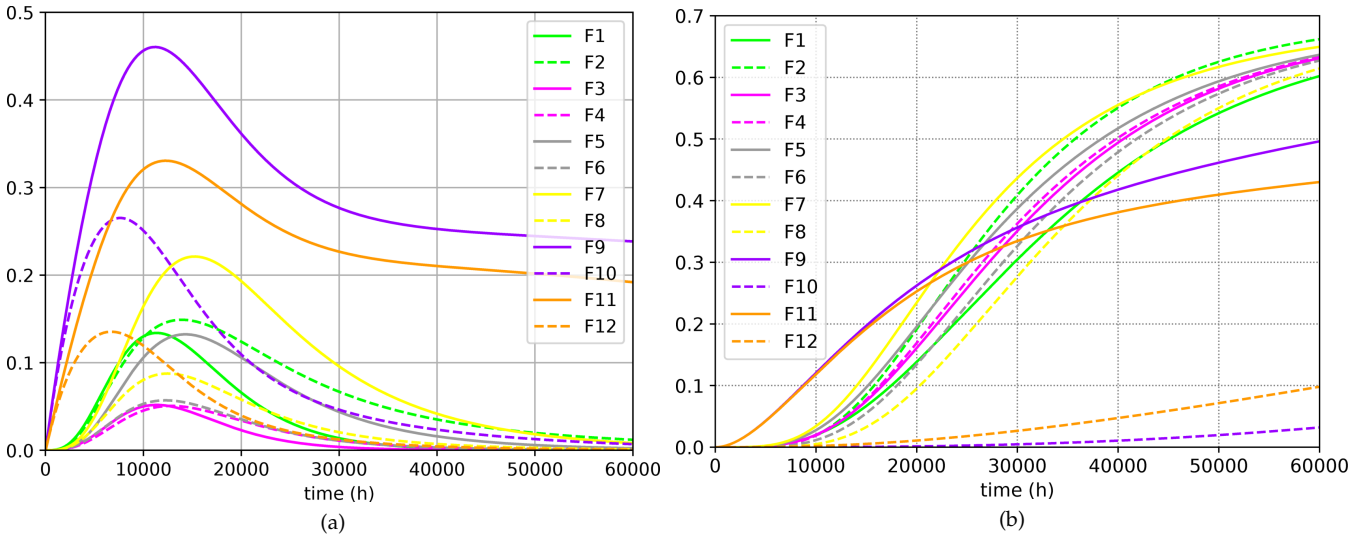

Fig. 10. GDS of Figs. 3 and 6: (a) Birnbaum measures and (b) Fussell-Vesely measures of faults F1, F2, . . . , F12.

The complexity of the solution technique implemented in Pyramis is cubic in the number of steps per region, quadratic in the number of time points, and linear in the number of regions of the HSMP hierarchy [13]. By transformation rules R3 and R4, an HSMP obtained through the FaultFlow-to-Pyramis transformation contains, collectively, one region for each internal fault, and one region for each child of each non-bottom-level gate of the SFT; in turn, each region contains at most 2 steps. Therefore, to test the analysis scalability for the this class of HSMPs, we modify the SSFT of Fig. 6b by increasing its depth and number of internal faults, without increasing its number of failure-to-fault propagations, which in fact would just increase the number of steps of the corresponding regions from 1 to 2 (i.e., it would result in adding a simple step modeling the duration of the propagation, like step `IB_F5_F6_to_Failure1` in Fig. 8b). In particular, we derive 4 different system SSFTs by replacing the sub-SSFT modeling the failure of each gas detector with the sub-SSFT of Figs. 11a to 11d, respectively. The obtained SFTs have depth 5, 5, 6, and 6, respectively, and contain 48, 96, 96, and 192 internal faults, respectively. Each added internal fault is characterized by an Exponential or Erlang time-to-fault CDF, with shape and rate in the same order of magnitude as those of Fig. 6b. Table 2 shows the execution times needed to compute the time-to-failure CDF of the system and the importance measures of *all* internal faults, with time limit 8000 h and time step 2 h. For the evaluation of the time-to-failure CDF of the system, the execution time is less than doubled when the number of internal faults doubles, mainly due to fixed costs of solution. As expected, the execution time needed to evaluate the Birnbaum measures is in the order of twice the time needed to compute the time-to-failure CDF of the system, multiplied by the number of internal faults (given that, for each fault, the time-to-failure CDF of the system is evaluated twice, once assuming that the fault has occurred and once that it has not). Conversely, evaluation of the Fussel-Vesely measures is quite fast, mainly due to the fact that the number of MCSs is much smaller than the number of internal faults. Overall, it is worth noting that evaluation is still feasible and far from bottleneck even in the case with 192 internal faults.

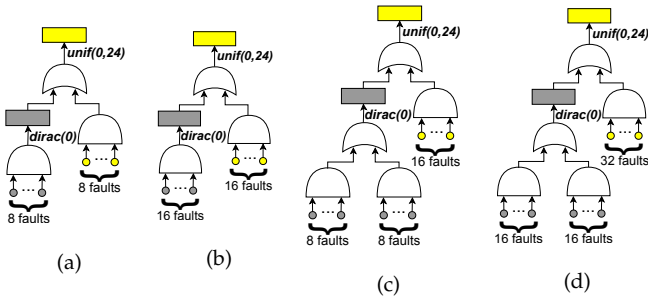

Fig. 11. Variants of the sub-SSFT modeling the failure of a gas detector in Fig. 6b, including (a) 8, (b) 16, (c) 16, and (d) 32 internal faults per component, i.e., 48, 96, 96, and 192 internal faults for the entire GDS.

TABLE 2
Execution times of the analysis of variants of the system of Fig. 6, obtained by replaing the sub-SSFT modeling the failure of each gas detector with the sub-SSFT of Figs. 11a to 11d.

SFT depth	no. internal faults	top event CDF	Birnbaum measures	Fussel-Vesely measures
5	48	24.44 s	35.89 min	47.75 s
5	96	38.65 s	1.87 h	71.09 s
6	96	41.87 s	2.06 h	2.13 min
6	192	67.72 s	6.91 h	3.23 min

5 FaultFlow use cases

In this section, we illustrate two use cases about the manipulation of FaultFlow metamodel instances (Section 5.1) and the estimation of time-to-fault PDFs from statistical data to compute quantitative dependability measures (Section 5.2).

5.1 Manipulating FaultFlow metamodel instances

The Pressure Tank System (PTS) of [64] is commonly used as reference use case and appears in the FFORT collection of fault tree models [56]. Fig. 12a shows the UML object diagram of the PTS structure, made of a tank T maintained in a filled and pressurized condition by a motorized pump.

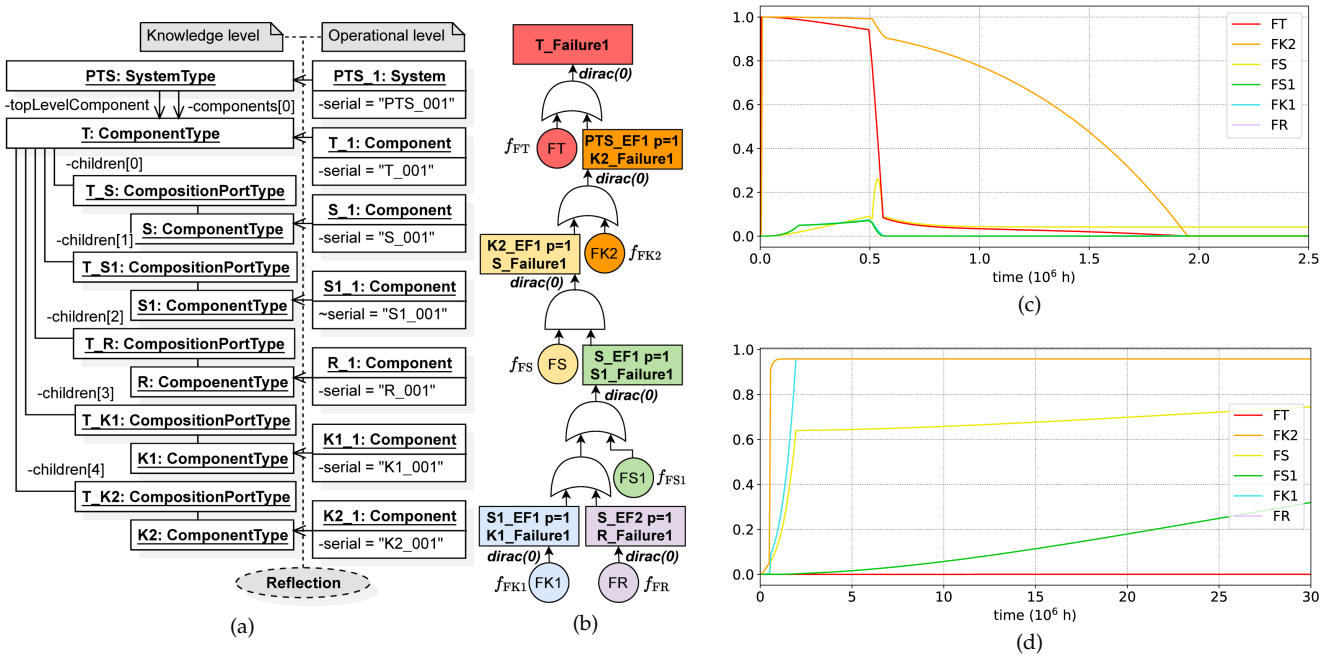

Fig. 12. Pressure tank system [56], [64]: (a) UML object diagram (structure); (b) SSFT (failure logic); (c) Birnbaum and (d) Fussell-Vesely measures.

TABLE 3
Time-to-fault data for the SSFT of Fig. 12 (P5 denotes the 5th percentile, P95 the 95th percentile, μ the mean value). Time is expressed in 10^6 h.

fault x	**time-to-fault statistics**	**time-to-fault PDF**
FT	$\mathrm{P95} = 7042.25$	$f_{\mathrm{FT}}^{\mathrm{fault}}(t) = 4.53776 \cdot 10^{-7} t\, e^{-0.00067\, t}\ \forall\, t \in [0, \infty)$
FS1	$\mathrm{P95} = 123.609$	$f_{\mathrm{FS1}}^{\mathrm{fault}}(t) = 0.00147\, t\, e^{-0.03838\, t}\ \forall\, t \in [0, \infty)$
FS	$\mathrm{P5} = 0.16667$ $\mathrm{P95} = 9.09091$	$f_{\mathrm{FS}}^{\mathrm{fault}}(t) = \begin{cases} t\, e^{10.7895\, t} & \forall\, t \in [0, 0.16667) \\ 0.253289\, t\, e^{0.51545\, x} & \forall\, t \in [0.16667, \infty) \end{cases}$
FR, FK1. FK2	$\mathrm{P5} = 0.49020$ $\mathrm{P95} = 0.55866$ $\mu = 0.52356$	$f_{x}^{\mathrm{fault}}(t) = \begin{cases} t\, e^{-2.85884\, t} & \forall\, t \in [0, 0.49020) \\ t\, e^{6.1204\, t} & \forall\, t \in [0.49020, 0.55866] \\ 96.0395\, t\, e^{-8.90495\, t} & \forall\, t \in [0.55866, \infty) \end{cases} \quad \forall\, x \in \{\mathrm{FR}, \mathrm{FK1}, \mathrm{FK2}\}$

The pump is managed by a control system made of: a manual switch S, a pressure switch S1 monitoring the tank pressure, a timer relay R, two simple relays K1 and K2. The PTS is activated by pressing S, which closes the contacts of K1. In turn, this operation closes the contacts of K2, which then activates the pump motor. When K1 is closed, the timer in R is started: after 60 s, it expires and opens K1, deactivating the pump. If the limit pressure is reached, S1 triggers and deactivates K2, powering the pump off.

The diagram shows the objects of the classes at the knowledge level, modeling the structure of the considered PTS type (e.g., the `ComponentType` object `T` modeling the tank type), and objects of the classes at the operational level, modeling a specific PTS instance (i.e., the one with serial number PTS_001). In fact, thanks to the Reflection architectural pattern, defining a new type of system or a new instance of a specific type of system amounts to implementing objects, not classes (in particular, 12 objects are sufficient to represent the PTS type, while 7 objects are needed to model *each instance* of PTS). This feature facilitates code usage, maintenance, and extendibility, also enabling instantiation at run-time without modifying the metamodel source code. Moreover, changes to the structure of the system impact the objects at the knowledge level only, without affecting the many more objects at the operational level (e.g., an interface between K1 and K2 is modeled just by a `CompositionPortType` object). Similar considerations apply to the metamodel instances of the system failure logic.

5.2 Deriving quantitative dependability measures

The *rupture of the tank* is a hazard, and we use FaultFlow to estimate its likelihood and derive the importance measures of faults. Fig. 12b shows the SSFT of the system failure logic. The top event (tank rupture) may be caused by a physical rupture of tank T itself, or by a malfunctioning of the control system preventing the pump to stop, which may be caused by a failure of relay K2 to actually disconnect the motor, or by a hazardous failure of the logic that controls the pump operation. A failure of either timer relay R or relay K1 may cause the pump to operate indefinitely; similarly, a commission failure of switch S1 may cause the timer to be reset continuously. However, for such failures to actually cause the tank rupture, pressure switch S must also fail.

In [64], no quantitative data on failures is reported. In our analysis, we use data carefully extracted from [14], i.e., the

fault rate values that identify the 5th percentile (P5), the 95th percentile, and the mean value (μ) of the time-to-fault PDF of various component categories. In particular, we use data of the following categories: "Relays – Protective" for K1, K2, and R; "Switches – Pneumatic – Pressure" for S; "Switches – Electric – Pressure" for S1; and, "Catastrophic" failure mode of "Vessel – Pressurized – Metallic" component for T. To show the modeling flexibility provided by exponomial PDFs, for each fault x, we define a different time-to-fault PDF $f_x^{\text{fault}}(t)$ depending on the considered statistics by using Wolfram Mathematica (details on the derivation are reported in the Supplemental Material). In particular: we fit P95 by an exponomial PDF with the same analytical expression over $[0, \infty)$; we fit P5 and P95 by a piece-wise exponomial PDF with two different analytical expressions over $[0, \text{P5})$ and $[\text{P5}, \infty)$; and, we fit μ, P5, and P95 by a piece-wise PDF with three different analytical expression over $[0, \text{P5})$, $[\text{P5}, \text{P95})$, and $[\text{P95}, \infty)$. Table 3. shows the considered statistics and the resulting time-to-fault PDFs .

Figs. 12c and 12d show the Birnbaum and Fussell-Vesely measures, respectively, both ranking FK2 as the most important fault within a large time interval of milliom hours. In fact, FK2 has the lowest P95, like FR and FK1, and, unlike them, comprises a single point of failure. This trend is also shown by the top-level time-to-failure CDF (not shown due to space limits), which is nearly equal to 1 at time $2 \cdot 10^6$ h.

6 Comparison with other tools

We compare the FaultFlow features with those of a selection of *currently supported* tools for dependability evaluation, i.e., CHESS [9], [46], OSATE [20], [25], DFTCalc [4], SAFEST [74], DFTRES [11], and SHyFTOO [15], and by more general-purpose tools for quantitative evaluation of stochastic models, i.e., Möbius [19], SHARPE [70], TimeNET [76], and CPN IDE [72]. We consider modeling, evaluation, and implementation issues, focusing on expressivity of formalisms, characheristics of solution techniques, and availability of source code. Other features beyond the scope of this paper (e.g., safety properties) are not considered.

Modeling. FaultFlow supports high-level modeling of the system structure and failure logic by BDDs and SSFTs, respectively, implementing automated translation into STPNs and HSMPs, and supporting export of STPNs to the XPN format of the ORIS tool, and modeling of indirect couplings (i.e., fault propagations between components that are not directly connected in the system architecture). Under this perspective, FaultFlow shares some similarities with the CHESS and OSATE tools, which are tailored to the model-driven design of embedded systems. They include plugins that support the automated analysis of non-functional properties, with OSATE also supporting model export to PRISM [40] and EMFTA[7] [26]. While both CHESS and OSATE enforce their specific design methodology and language (i.e., CHESS ML and AADL, respectively), which include also concepts and modeling steps that are not relevant for dependability analysis, FaultFlow focuses on dependability analysis and provides a compact and precise language to define dependability-related concepts: in this respect, it is similar to the Intermediate Dependability Model (IDM) defined in the CHESS "State-Based Analysis" plugin (CHESS-SBA) [46]. DFTCalc, SAFEST, and DFTRES support failure logic specification by DFTs, thus considering the timing behavior of components in determing a failure, and supporting modeling of a variety of dependability patterns such as spare management (also note that duration of fault propagations can be represented by DFTs through SEQ gates). SHyFTOO model stochastic hybrid multi-state[8] systems by Stochastic Hybrid Fault Tree Automata (SHyFTA), with system failure and repair processes modeled by a DFT. The other tools are mainly focused on directly creating Petri net models for the entire system: although most of them include modularity mechanisms to some extent, the user is still expected to have significant modeling expertise. Both Möbius and SHARPE also support other formalisms, e.g., SHARPE facilitates failure logic specification through FTs and RBDs. Still, they do not combine those formalism with a system architecture view, like instead FaultFlow does.

A distinguishing feature of FaultFlow is the modeling of non-Markovian (expolynomial) duration distributions [70], possibly with bounded support. The FaultFlow metamodel also supports the representation of repeated events and probabilities of failure modes, while it currently does not support repair activities. CHESS represents selected non-Markovian duration distributions, as well as failure mode probabilities and repair actions. OSATE models Exponential durations and repair activities, while it does not represent repeated events and failure mode probabilities. DFTCalc, SAFEST, DEFTRES, and SHyFTOO model Exponential durations, with DFTCalc and DFTRES supporting also phase-type distributions, and SAFEST and SHyFTOO modeling also the Weibull distribution. All these four tools support repeated events and, except for SAFEST, repair activities.

Evaluation. FaultFlow offers different complementary strategies through M2M transformation (see also Fig. 1b). The CDF of the time-to-occurrence of any failure in the model (not necessarily the top-level one) can be obtained either through the SIRIO library (by performing semi-symbolic analysis based on the method of stochastic state classes), or, if the failure logic does not include any repeated event, through the Pyramis library. In the latter case, also importance measures of faults can be computed. Furthermore, interactive evaluation can also be performed by exporting the STPN model to the ORIS GUI. In CHESS, two plugins address dependability evaluation: CHESS-SBA [47] and CHESS-FLA [28]. CHESS-FLA focuses on qualitative analysis, through a formalism based on the Fault Propagation and Transformation Calculus (FTPC); CHESS-SBA is instead based on SPNs and supports quantitative analysis, but the evaluation can be performed only by discrete-event simulation. OSATE implements evaluation of the system failure probability through numerical analysis, mainly exploiting the external tools PRISM and EMFTA. DFTCalc, SAFEST, DFTRES, and SHyFTOO evaluate dependability measures such as reliability and Mean Time To Failure (MTTF), mainly using probabilistic model checking or simulation, also including Rare Event SImulation (RES).

7. https://github.com/cmu-sei/emfta

8. A multi-state system is a system that can be in multiple degrading states (unlike dyadic systems that can be either operational or failed).

TABLE 4
Comparison among tools for dependability evaluation of systems: main *modeling* (top), *evaluation* (center), and *implementation* aspects (bottom). Tools are grouped by abstraction level: in FaultFlow, CHESS, OSATE, DFTCalc, SAFEST, DFTRES, and SHyFTOO, the user reasons on the system architecture level, while in the others the user directly edits Petri nets or similar models. For the latter group, some dimensions are marked as n.a. (not applicable): they are able to support repeated events or repair activities, for example, but the user has to model them explicitly.

tool/library	high-level modeling formalism	duration distributions	indirect couplings	repeated events	failure mode probabilities	repair activities
FaultFlow & Sirio	SysML BDDs, SSFTs	general (expolynomial)	yes	yes	yes	no
FaultFlow & Pyramis	SysML BDDs, SSFTs	general (expolynomial)	yes	no	yes	no
CHESS	CHESS-ML	general (selected)	yes	yes	yes	yes
OSATE	AADL, SFTs, RBDs	Exponential	no	no	no	yes
DFTCalc	DFTs	Exponential, phase-type	no	yes	no	yes
SAFEST	SFTs, DFTs	Exponential, Erlang, Weibull, log-normal	no	yes	no	no
DFTRES	DFTs	Exponential, phase type	no	yes	no	yes
SHyFTOO	DFTs (in SHyFTA)	Exponential, Weibull	no	yes	no	yes
SHARPE	FTs, RBDs	general (expolynomial)	n.a.	n.a.	n.a.	n.a.
Möbius	FTs	general				
TimeNET	-	general (expolynomial)				
CPN IDE	-	general				

tool/library	evaluation formalism	solution engine	supported quantitative dependability measures
FaultFlow & Sirio	STPNs	semi-symbolic analysis	time-to-failure CDF
FaultFlow & Pyramis	HSMPs	numerical analysis	time-to-failure CDF, fault importance measures
CHESS	SPNs, CTMCs	FTA, simulation	reliability, availability
OSATE	DTMCs, CTMCs	numerical analysis	failure probability
SAFEST	I/O-MC	probabilistic model checking	reliability, MTTF
	CTMCs	probabilistic model checking	unreliability, MTTF, component criticality
DFTRES	CTMCs	statistical model checking, RES	reliability, availability, MTTF
SHyFTOO	SHyFTA	simulation	unreliability
SHARPE	GSPNs	numerical analysis	n.a. (generic reward-based measures)
Möbius	SAN, CTMCs, PEPA	numerical analysis, simulation	
TimeNET	DSPNs	numerical analysis, simulation	
CPN IDE	CPNs	simulation	

tool/library	export to external tool	API	GUI	open-source
FaultFlow & Sirio	yes (ORIS)	yes	no	yes
FaultFlow & Pyramis	no	yes	no	yes
CHESS	partially (PNML)	no	yes	partially
OSATE	yes (PRISM, EMFTA)	yes	yes	yes
DFTCalc	yes (CADP, MRMC, IMCA)	no	yes	yes
SAFEST	no	no	yes	yes
DFTRES	yes (DFTCalc)	no	no	yes
SHyFTOO	no	no	no	yes
SHARPE	no	no	yes	no
Möbius	no	no	yes	no
TimeNET	no	no	yes	no
CPN IDE	no	no	yes	no

Similar dependability measures could be computed through the other tools, though increasing user effort. Möbius provides different evaluation options, including a very robust discrete-event simulator that also supports distributed processing. Efficient exact solvers are available, but most of them assume Exponential distributions for timed events. More precisely, for the analysis engines of SHARPE, Möbius, and TimeNET, numerical evaluation is constrained to the limitation of having at most one timer with general distribution enabled in each state (enabling restriction) [29], [30]. These tools typically support the evaluation of different measures, based on rewards, but the user has to explicitly define the reward structure manually.

Implementation. The FaultFlow API is released open-source under the AGPLv3 licence; being implemented in Java, it is compatible with the most common platforms. Both the OSATE GUI and API are released under the EPLv2 licence, and the GUI is implemented as a customization of the Eclipse platform. The main modeling environment of CHESS is also based on Eclipse, and it is now maintained as an Eclipse project under the Eclipse PolarSys initiative; consequently, it is also released under the EPLv2 licence. However, some specific components are excluded from the main package, most notably the simulator used in the CHESS-SBA plugin, which is released separately. The other tools also provide a GUI, except for DFTRES and SHyFTOO, and none of them provides a documented API. DFTCalc, SAFEST, DFTRES, and SHyFTOO are released open-source, while most of the remaining tools are free for academic and research use, but are distributed under commercial licenses otherwise. CPN IDE has recently replaced the former CPN Tools framework. While CPN Tools was released under an

open source license, at the time of writing its successor CPN IDE is not released open-source and it is only available as pre-compiled binary for the Windows platform.

Remark. This discussion is summarized in Table 4, with additional comparison viewpoints. Some dimensions are not applicable to tools like Möbius or TimeNET, where the user directly creates Petri net models and, for example, has to explicitly model repeated events or repair activities.

7 Conclusions

We presented an MDE approach to dependability evaluation of component-based coherent systems, implemented by the open-source FaultFlow Java library. A custom-made metamodel represents the system structure and failure logic, which can be derived from a SysML BDD and an SSFT, respectively, representing durations with non-Markovian distribution possibly with bounded support. The approach derives the distribution of the time to the occurrence of given failures and importance measures of faults over time.

Flexibility and extensibility of the FaultFlow metamodel enable further developments, notably including: definition of an observation metamodel to generate synthetic data sets of failure propagation events during model simulation, a preliminary version of which has been presented in [12]; integration with other design tools (e.g., Eclipse Papyrus) and dependability tools (e.g., CHESS) by implementing appropriate M2M transformations; and, dynamic management of metamodel instances to support co-evolution of software artifacts and runtime evaluation of dependability.

Acknowledgments

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART"), and by the MUR PRIN 2022 PNRR P2022A492B project ADVENTURE (ADVancEd iNtegraTed evalUation of Railway systEms).

References

[1] *ISO 26262: road vehicles functional safety*, 2011.
[2] M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. on Computer Systems*, 2(2):93–122, 1984.
[3] E. G. Amparore, G. Balbo, M. Beccuti, S. Donatelli, and G. Franceschinis. 30 years of GreatSPN. *Principles of performance and reliability modeling and evaluation*, pages 227–254, 2016.
[4] F. Arnold, A. Belinfante, F. Van der Berg, D. Guck, and M. Stoelinga. DFTCalc: a tool for efficient fault tree analysis. In *SAFECOMP*, pages 293–301. Springer, 2013.
[5] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11–33, 2004.
[6] S. Bernardi, J. Merseguer, and D. C. Petriu. A dependability profile within MARTE. *Soft. & Sys. Model.*, 10(3):313–336, 2011.
[7] J. Boardman and B. Sauser. System of systems-the meaning of of. In *IEEE/SMC Int Conf. on SoS Eng.*, pages 6–pp. IEEE, 2006.
[8] H. Boudali, A. Nijmeijer, A. Nijmeijer, and M. I. A. Stoelinga. Dftsim: A simulation tool for extended dynamic fault trees. In *42nd Annual Simulation Symposium (ANSS)*, page 31. ACM, 2009.
[9] L. Bressan, A. L. de Oliveira, L. Montecchi, and B. Gallina. A systematic process for applying the CHESS methodology in the creation of certifiable evidence. In *EDCC*, pages 49–56. IEEE, 2018.
[10] C. E. Budde. Fig: the finite improbability generator v1. 3. *ACM SIGMETRICS Performance Evaluation Review*, 49(4):59–64, 2022.
[11] C. E. Budde, E. Ruijters, and M. Stoelinga. The dynamic fault tree rare event simulator. In *International Conference on Quantitative Evaluation of Systems*, pages 233–238. Springer, 2020.
[12] L. Carnevali, S. Cerboni, B. Picano, L. Scommegna, and E. Vicario. An observation metamodel for dependability tools. In *European Dependable Computing Conf. (EDCC)*, pages 169–172. IEEE, 2024.
[13] L. Carnevali, R. German, F. Santoni, and E. Vicario. Compositional Analysis of Hierarchical UML Statecharts. *IEEE Trans. on Soft. Eng.*, 48(12):4762–4788, 2021.
[14] Center for Chemical Process Safety. *Guidelines for Process Equipment Reliability Data with Data Tables*. American Institute of Chemical Engineers, January 1989.
[15] F. Chiacchio, J. I. Aizpurua, L. Compagno, and D. D'Urso. Shyftoo, an object-oriented monte carlo simulation library for the modeling of stochastic hybrid fault tree automaton. *Expert Systems with Applications*, 146:113139, 2020.
[16] H. Choi, V. G. Kulkarni, and K. S. Trivedi. Markov regenerative stochastic petri nets. *Performance evaluation*, 20(1-3):337–357, 1994.
[17] G. Ciardo and A. S. Miner. Smart: The stochastic model checking analyzer for reliability and timing. In *Int. Conf. on the Quantitative Evaluation of Systems*, pages 338–339. IEEE, 2004.
[18] A. R. Da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.
[19] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Mobius framework and its implementation. *IEEE Tran. Soft. Eng.*, 28(10):956–969, 2002.
[20] J. Delange, P. Feiler, D. P. Gluch, and J. Hudak. AADL fault modeling and analysis within an ARP4761 safety assessment. Technical report, Carnegie-Mellon Univ. Soft. Eng. Inst., 2014.
[21] J. B. Dugan, K. J. Sullivan, and D. Coppit. Developing a low-cost high-quality software tool for dynamic fault-tree analysis. *IEEE Transactions on reliability*, 49(1):49–59, 2000.
[22] P. R. D'Argenio, M. D. Lee, and R. E. Monti. Input/output stochastic automata: Compositionality and determinism. In *FORMATS*, pages 53–68. Springer, 2016.
[23] EUROCAE. *ARP4754A - Guidelines for Development of Civil Aircraft and Systems*, 2010.
[24] FaultFlow. https://doi.org/10.5281/zenodo.14615110, 2025.
[25] P. Feiler. The open source AADL tool environment (OSATE). Technical report, Carnegie Mellon Univ. Soft. Eng. Inst., 2019.
[26] P. Feiler and J. Delange. Automated fault tree analysis from aadl models. *ACM SIGAda Ada Letters*, 36(2):39–46, 2017.
[27] P. H. Feiler, B. A. Lewis, and S. Vestal. The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems. In *CACSD-CCA-ISIC*, pages 1206–1211. IEEE, 2006.
[28] B. Gallina, M. A. Javed, F. U. Muram, and S. Punnekkat. A model-driven dependability analysis method for component-based architectures. In *Euromicro Conf. on Software Engineering and Advanced Applications*, pages 233–240. IEEE, 2012.
[29] R. German and C. Lindemann. Analysis of stochastic Petri nets by the method of supplementary variables. *Performance evaluation*, 20(1-3):317–335, 1994.
[30] R. German, D. Logothetis, and K. S. Trivedi. Transient analysis of Markov regenerative stochastic Petri nets: A comparison of approaches. In *Int. Work. PNPM*, pages 103–112. IEEE, 1995.
[31] L. Grunske and B. Kaiser. Automatic generation of analyzable failure propagation models from component-level failure annotations. In *Int. Conf. on Quality Software*, pages 117–123, 2005.
[32] J. Hillston. A compositional approach to performance modelling. 1996.
[33] A. Horváth, M. Paolieri, L. Ridi, and E. Vicario. Transient analysis of non-Markovian models using stochastic state classes. *Perform. Eval.*, 69(7-8):315–335, July 2012.
[34] Int. Organization for Standardization. *ISO 9241 - Ergonomic requirements for office work with visual display terminals (VDTs)*, 2000.
[35] K. Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media, 1996.
[36] S. Kabir. An overview of fault tree analysis and its application in model based dependability analysis. *Expert Systems with Applications*, 77:114–135, 2017.
[37] B. Kaiser, P. Liggesmeyer, and O. Mäckel. A new component concept for fault trees. In *Austr. Work. on Safety Critical Sys. and Soft.*, volume 33, page 37–46. Austr. Comp. Society, Inc., 2003.

[38] H. Kopetz, B. Frömel, and O. Höftberger. Direct versus stigmergic information flow in systems-of-systems. In *2015 10th System of Systems Engineering Conference (SoSE)*, pages 36–41, 2015.
[39] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *TOOLS*, pages 200–204. Springer, 2002.
[40] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *International conference on computer aided verification*, pages 585–591. Springer, 2011.
[41] C. Lindemann. Performance modelling with deterministic and stochastic petri nets. *ACM Sigm. Perf. Eval. Rev.*, 26(2):3, 1998.
[42] O. Lisagor. *Failure logic modelling: a pragmatic approach*. PhD thesis, University of York, 2010.
[43] G. Manno, F. Chiacchio, L. Compagno, D. D'Urso, and N. Trapani. Conception of Repairable Dynamic Fault Trees and resolution by the use of RAATSS, a Matlab® toolbox based on the ATS formalism. *Reliability Eng. & System Safety*, 121:250–262, 2014.
[44] G. Manno, F. Chiacchio, L. Compagno, D. D'Urso, and N. Trapani. MatCarloRe: An integrated FT and Monte Carlo Simulink tool for the reliability assessment of dynamic fault tree. *Expert Systems with Applications*, 39(12):10334–10342, 2012.
[45] S. Montani, L. Portinale, A. Bobbio, and D. Codetta-Raiteri. Radyban: A tool for reliability analysis of dynamic fault trees through conversion into dynamic bayesian networks. *Reliability Engineering & System Safety*, 93(7):922–932, 2008.
[46] L. Montecchi, P. Lollini, and A. Bondavalli. Towards a mde transformation workflow for dependability analysis. In *IEEE Int. Conf. on Eng. of Complex Comp. Sys.*, pages 157–166. IEEE, 2011.
[47] L. Montecchi, P. Lollini, and A. Bondavalli. A Reusable Modular Toolchain for Automated Dependability Evaluation. In *Int. Conf. on Perf. Eval. Meth. and Tools*, pages 298–303, Torino, Italy, 2013.
[48] L. Montecchi, A. Refsdal, P. Lollini, and A. Bondavalli. A model-based approach to support safety-related decisions in the petroleum domain. In *DSN*, pages 275–286. IEEE, 2016.
[49] Object Management Group. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems v1.0*, 2009.
[50] Object Management Group, www.omg.org/spec/SysML/1.6/. *OMG Systems Modeling Language 1.6 (OMG SysML)*, 2019.
[51] M. Paolieri, M. Biagi, L. Carnevali, and E. Vicario. The ORIS Tool: Quantitative Evaluation of Non-Markovian Systems. *IEEE Transactions on Software Engineering*, 47(6):1211–1225, 2021.
[52] J. Parri, S. Sampietro, and E. Vicario. Faultflow: a tool supporting an mde approach for timed failure logic analysis. In *European Dependable Computing Conference*, pages 25–32. IEEE, 2021.
[53] Radio Tech. Commis. for Aeronautics. *DO-178C, Software Considerations in Airborne Systems and Equipment Certification*, 2011.
[54] Radio Tech. Commis. for Aeronautics. *DO-331: model-based development and verification supplement to DO-178C and DO-278A*, 2011.
[55] A.-E. Rugina, K. Kanoun, and M. Kaâniche. The ADAPT tool: From AADL architectural models to stochastic petri nets through model transformation. In *EDCC*, pages 85–90. IEEE, 2008.
[56] E. Ruijters et al. FFORT: A Benchmark Suite for Fault Tree Analysis. In *ESREL*. Research Publishing, Singapore, 2019.
[57] E. Ruijters and M. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review*, 15:29–62, 2015.
[58] SAE International. *AADL Error Model Annex, Standards Document AS5506/1*, 2006.
[59] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Computing Surveys*, 42(3):1–42, 2010.
[60] W. H. Sanders and J. F. Meyer. Stochastic activity networks: formal definitions and concepts. In *School organized by the European Educational Forum*, pages 315–343. Springer, 2000.
[61] D. C. Schmidt. Model-driven engineering. *Computer-IEEE Computer Society-*, 39(2):25, 2006.
[62] B. Silva, G. Callou, E. Tavares, P. Maciel, J. Figueiredo, E. Sousa, C. Araujo, F. Magnani, and F. Neves. Astro: An integrated environment for dependability and sustainability evaluation. *Sustainable computing: informatics and systems*, 3(1):1–17, 2013.
[63] B. Silva, R. Matos, G. Callou, J. Figueiredo, D. Oliveira, J. Ferreira, J. Dantas, A. Lobo, V. Alves, and P. Maciel. Mercury: An integrated environment for performance and dependability evaluation of general systems. In *IEEE/IFIP Int. Conf. Depend. Sys. and Net.*, 2015.
[64] M. Stamatelatos et al. Fault Tree Handbook with Aerospace Applications. NASA Office of Safety and Mission Assurance, 2002.
[65] D. H. Stamatis. *Failure mode and effect analysis: FMEA from theory to execution*. Quality Press, 2003.
[66] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.
[67] D. Stewart, J. J. Liu, M. Heimdahl, D. Cofer, and M. Peterson. Safety annex for the architecture analysis and design language. 2018.
[68] Z. Tang and J. B. Dugan. Minimal cut set/sequence generation for dynamic fault trees. In *Sym. Rel. & Maint.*, pages 207–213, 2004.
[69] K. S. Trivedi and A. Bobbio. *Reliability and availability engineering: modeling, analysis, and applications*. Cambridge Univ. Press, 2017.
[70] K. S. Trivedi and R. Sahner. Sharpe at the age of twenty two. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):52–57, 2009.
[71] M. Van der Borst and H. Schoonakker. An overview of PSA importance measures. *Rel. Eng. & Sys. Safety*, 72(3):241–245, 2001.
[72] E. Verbeek and D. Fahland. CPN IDE: An Extensible Replacement for CPN Tools That Uses Access/CPN. In *Int. Conf. on Process Mining Doctoral Consortium and Demo Track*, pages 29–30, 2021.
[73] E. Vicario, L. Sassoli, and L. Carnevali. Using stochastic state classes in quantitative evaluation of dense-time reactive systems. *IEEE Trans. on Software Engineering*, 35(5):703–719, 2009.
[74] M. Volk, F. Sher, J.-P. Katoen, and M. Stoelinga. Safest: Fault tree analysis via probabilistic model checking. In *Annual Reliability and Maintainability Symp. (RAMS)*, pages 1–7. IEEE, 2024.
[75] W. Wang, J. Loman, and P. Vassiliou. Reliability importance of components in a complex system. In *Annual Symposium Reliability and Maintainability, 2004-RAMS*, pages 6–11. IEEE, 2004.
[76] A. Zimmermann. Modelling and performance evaluation with TimeNET 4.4. In *QEST*, pages 300–303. Springer, 2017.

Laura Carnevali is Associate Professor of Computer Science at the School of Engineering, University of Florence, Italy, where she received the Ph.D. degree in Informatics, Multimedia, and Telecommunications Engineering in 2010. Her research is focused on quantitative evaluation of stochastic models and on its application to various domains through model driven engineering.

Stefania Cerboni is Research Fellow at the School of Engineering, University of Florence, Italy, where she received the B.S. degree in Computer Engineering in 2019. Her research is in the area of Software Engineering, specifically in software architectures and methodologies and Model Driven Engineering practices. As part of her research, she contributed to the design and development of the FaultFlow API.

Leonardo Montecchi is Associate Professor at the Norwegian University of Science and Technology in Trondheim, Norway. Previously, he was Assistant Professor at the University of Campinas, Brazil. He received the Ph.D. in Computer Science, Systems and Telecommunications (2014) from the University of Florence, Italy, where he also was Postdoctoral Fellow until 2017. His expertise revolves around modeling of complex systems, including formal models, probabilistic models, and model-driven engineering, applied to verification and validation of complex systems.

Enrico Vicario is a Professor of Computer Science and Engineering and Head of the Department of Information Engineering at the University of Florence, Italy. His research is in the area of Software Engineering, with a present focus on quantitative evaluation of stochastic models, software architecture and methodologies, and on their connection through Model Driven Engineering practices.