Reliability Engineering and System Safety 140 (2015) 191–199

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Comparison of modeling formalisms for Safety Analyses: SAML and AltaRica

Michael Lipaczewski [a], Frank Ortmeier [a], Tatiana Prosvirnova [b,*], Antoine Rauzy [b], Simon Struck [a]

[a] Otto-von-Guericke University Magdeburg, Computer Systems in Engineering, Magdeburg, Germany
[b] LIX – Ecole Polytechnique, route de Saclay, 91128 Palaiseau cedex, France

ARTICLE INFO

Article history:
Received 18 November 2013
Received in revised form
1 August 2014
Accepted 28 March 2015
Available online 8 April 2015

Keywords:
Model-Based Safety Analysis
SAML
AltaRica

ABSTRACT

Many states/transitions formalisms have been proposed in the literature to perform Safety Analyses. In this paper we compare two of them: SAML and AltaRica. These formalisms have been developed by different communities. Their "look-and-feel" are thus quite different. Yet, their underlying mathematical foundations are very similar: both of them rely on state automata. It is therefore of interest to study their ability to assess the reliability of systems, their respective advantages and drawbacks and to seek for opportunities of a cross fertilization.

1. Introduction

Model based approach for safety analysis is gradually wining the trust of safety engineers but is still a wide domain of research. "Traditional" risk modeling formalisms, such as Fault Trees (FT) [1], Markov Processes, and Generalized Stochastic Petri Nets (GSPN) [2] are well known and widely used by safety engineers; and efficient algorithms and tools are available to study these models. However, despite of their qualities, these formalisms share a major drawback: models designed with these formalisms are far from the functional architecture of the system under study. As a consequence, models are hard to design and to maintain throughout the life cycle of systems. A small change in the specifications may require a complete revisit of the safety models, which is both resource consuming and error prone. The high-level modeling languages AltaRica [3,4] and SAML [5] have been created to tackle this problem.

SAML was designed as a tool independent formal system specification and modeling language [5]. A SAML model is expressed in terms of finite stochastic state automata. A model may consists of more than one automata, which are all executed in discrete time with parallel composition. Besides technical systems with deterministic behavior SAML may also denote failure models with stochastic behavior and system environments which often have non-deterministic behavior. Due to the combination of stochastic and non-deterministic specification, the semantics of a SAML model is defined as Markov decision process.

AltaRica [3,4] has been designed with engineering perspective. AltaRica models are made of hierarchies of reusable components. Graphical representations are associated to components, making models visually very close to Process and Instrumentation Diagrams. AltaRica is used as internal representation language by several Safety Analyses workshops: Cecilia OCAS (Dassault Aviation), Simfia (EADS Apsys), Safety Designer (Dassault Systèmes) and AltaRica Studio (LaBRI). AltaRica is a formal modeling language. Efficient algorithms have been developed to assess AltaRica models: compilation to Fault Trees, stochastic simulation, model-checking, generation of Markov chains, etc.

It is of interest to compare both formalisms in order to study their ability to assess the reliability of systems, their respective advantages and drawbacks and to seek for opportunities of a cross fertilization. These two formalisms are compared according to the following axes:

- the high-level structural constructs;
- the underlying finite state automata;
- the representation and interpretation of time.

To illustrate our comparison we use a case study: a power supply system. We present some qualitative and quantitative results obtained with both formalisms and the advantages and drawbacks of both formalisms.

The remainder of this paper is organized as follows. Sections 2 and 3 introduce respectively SAML and AltaRica modeling languages.

* Corresponding author.
E-mail addresses: frank.ortmeier@ovgu.de (F. Ortmeier), prosvirnova@lix.polytechnique.fr (T. Prosvirnova), rauzy@lix.polytechnique.fr (A. Rauzy), simon.struck@ovgu.de (S. Struck).

http://dx.doi.org/10.1016/j.ress.2015.03.038
0951-8320/

Section 4 gives an overview of the related works. Section 5 describes a case study, a power supply system that will be used to illustrate both formalisms. Section 6 presents SAML model and AltaRica model of the power supply system. Section 7 gives some qualitative and quantitative results obtained with SAML and AltaRica models. Section 8 compares both formalisms and, finally, Section 9 concludes this paper.

2. SAML

2.1. The language

This section provides a brief introduction to SAML. An extensive explanation of the semantics is out of scope of this paper. The interested reader is referred to Güdemann et al. [5]. In addition we evolved the language in the mean time. Thus we present the model in the currently most up to date version.

Semantically, a SAML model denotes a Markov Decision Process (MDP). This allows the modeling of time and value discrete systems with deterministic, probabilistic and non-deterministic aspects. For a formal definition of MDP see e.g. [6].

From a syntactical point of view SAML consists of a set of *components*. Every component may contain additional components and/or state automaton. Every automaton is defined by one ore more state variables and a set of update rules. The state variables are bounded integer variables. The update rule consists of an activation condition in propositional logic and a set of states reachable if the activation condition is true. The set of reachable states is specified as a set of non-deterministic choices. Within each choice a probability distribution may be denoted. The initial state of the automaton is specified with the initial value of the state variables. *Constants*, *formulas* and *enums* can be used to increase the readability of the model. Formulas are named abbreviations for propositional logic expressions. Enums are primarily used to label system states with handy names.

Multiple automata are combined in terms of synchronous parallel composition. This means that all automata in the model move exactly one step at every time step.

2.2. Tools and analysis

SAML was designed as tool-independent modeling language. Rather than implementing dedicated SAML centric analysis tools we use automatic semantic-preserving model transformations to transform SAML models into the input language of state-of-the-art verification engines. Integrating as much model checkers as possible allows the user to choose the most appropriate one for the problem at hand. So far there are transformations for NuSMV [7] and PRISM [8]. These transformations are semantic preserving and fully automatic [5]. With PRISM as an intermediate converter it is also possible to use MRMC [9] and we are currently busy with a converter to UPPAAL.[1] Note that none of the proposed analysis tools is limited to safety analysis.

We use NuSMV for symbolic model checking of SAML models. Due to the qualitative nature of NuSMV, the transformation from SAML into the NuSMV input language replaces all probability distributions with appropriate non-deterministic choices. The symbolic model checking approach allows the verification of SAML models against arbitrary CTL [10] properties. This solves questions like, “Is a certain (dangerous) state reachable” or “When ever X is true in one state, Y is true in the next state”.

For qualitative safety analysis we use the deductive cause consequence analysis (DCCA) [11] to compute all minimal cut-sets (i.e. critical failure combinations). It is a structured approach to search the space of failure combinations and uses model checking to test whether a hazardous state is reachable if a certain failure combination occurs. The DCCA approach is optimized to use only a minimal number of model-checking runs to exploit the complete search space. In addition to the minimal set of critical failure combinations, every combination is demonstrated by an example (i.e. sequence of states) leading to the hazard.

In addition to NuSMVs qualitative analysis, PRISM is a probabilistic model checker that exploits the stochastic information in the model. It can perform quantitative analysis like “How likely is it in the worse-case that a certain (dangerous) state is reached”. It can analyze any kind of pCTL [6] formula. Rather than calculating sets of failure combinations, the pDCCA calculates the overall hazard probability. The calculation is based on the occurrence probabilities of all discrete failures as well as the functional behavior of the model. In case of non-deterministic model components, the result is either a worse-case or best-case analysis.

3. AltaRica

This section gives an overview of AltaRica modeling language and of the associated assessment tools.

3.1. The language

AltaRica is a high level modeling language dedicated to Safety Analyses. The first version of AltaRica was developed in LaBRI in 1990s [12,3]. A few years later, a second (Data-Flow) version has been developed to handle industrial scale models. A number of assessment tools have been developed for AltaRica such as compilers to Fault Trees, compilers to Markov chains, generators of critical sequences, stochastic simulators and model-checkers. Several Integrated Modeling and Simulation Environments use AltaRica as their internal representation language. Successful industrial applications have been reported [13,14].

The third version (AltaRica 3.0) is still under specification. AltaRica 3.0 will be a major evolution of the language (and the processing tools). This new version integrates notions of object-oriented programming languages, such as inheritance and prototypes. It improves the reusability of components and knowledge capitalization. It adds also the ability to handle looped systems and to define acausal components. The models presented in this paper are written in AltaRica 3.0.

AltaRica is an event-centric language because the primary objective of Safety and Reliability studies is to detect and quantify the most probable sequences of events (failures) leading the system from a nominal state to a degraded state (accident). In AltaRica, the behavior of components is described by means of Guarded Transition Systems [15,16]. Guarded Transition Systems generalize widely used formalisms such as Reliability Block Diagrams, Markov chains and Generalized Stochastic Petri nets. The state of a component is represented by variables (so-called state variables) and their values. The changes of state are possible when, and only when, an event occurs. The occurrence of an event updates the values of variables. Deterministic or stochastic delays can be associated with events in order to obtain (stochastic) timed models. Components can be assembled into hierarchies, their inputs and outputs can be connected and their transitions can be synchronized. So, an AltaRica model can be seen as a hierarchy of interconnected components that can be “flattened” into a unique Guarded Transition System.

3.2. Analysis

The semantics of a Guarded Transition System is a Kripke structure (a reachability graph) that can be interpreted as a Continuous-Time

[1] 2013-03-21: http://www.uppaal.org.

Markov Chain, under the condition that delays associated with transitions are exponentially distributed, or compiled into a Fault Tree.

A number of efficient assessment tools have been developed for Data-Flow Guarded Transition Systems, such as compilers to Fault Trees [17], compilers to Markov chains, generators of critical sequences of events, stochastic and stepwise simulators and model-checkers.

All the underlying algorithms can be extended to a general case of Guarded Transition Systems (without Data-Flow condition). Guarded Transition Systems make it possible to handle systems with instant loops and to define acausal components, i.e. components for which the input and output flows are decided at run time (e.g. electrical systems).

Fault Tree compiler: Fault Trees are widely used to perform Safety Analyses and some regulation authorities require to use them to support the certification process. Since high-level modeling greatly improves the design, the sharing and the maintenance of models, it is of interest to use them to automatically generate Fault Trees. In many cases high-level models can be efficiently compiled into Fault Trees. The generated Fault Tree can be then assessed with calculation engines, such as XFTA [18], in order to calculate minimal cutsets, probabilities of failures, importance factors and other reliability indicators.

Markov chain generator: The compilation into Markov chains requires all the transitions to be either with exponential delays or immediate. Immediate transitions are just collapsed using the fact that an exponential delay with rate λ followed by an immediate transition of probability p is equivalent to a transition with an exponential delay of rate $p\lambda$. The problem of such a compilation is indeed the combinatorial explosion of the number of states and transitions.

Stepwise simulator: Stepwise simulator enables to perform an interactive step by step simulation of the model. This interactive tool can be very useful to debug models, to play different failure scenarios, etc. The stepwise simulator can be coupled with a graphical simulator as illustrated in [19]. Graphical simulation of models can be used to perform virtual experiments on systems, via models, helping to better understand the system behavior.

Stochastic simulator: Stochastic (Monte-Carlo) simulation is used when other assessment methods fail. The principle is to run many histories drawing at pseudo-random the delays of the transitions and to make statistics on these histories. Two types of observers can be defined to calculate reliability indicators: observers on formulas (e.g. the average number of times a formula takes a given value) and observers on events (e.g. the average number of times an event has been fired). The only limit of stochastic simulation is the number of histories and the length of histories that are necessary to stabilize the measures.

Generator of critical sequences of events: A critical sequence is a sequence of events leading from the initial state to a critical state. In some cases, the order of occurrences of events does matter and thus the approximation consisting in extracting minimal cutsets (through a compilation of the model into a Fault Tree) is not suitable. In that case, minimal sequences can be extracted.

4. Related works

Many other high-level modeling languages for Safety Analyses have been defined. Two approaches for (high-level) Model-Based Safety Assessment can be found. The first one consists in creating extensions of high-level modeling languages used in other domains. The second approach consists in defining domain specific languages, dedicated to Safety Analyses. In this section we will cite some of them.

In the first category, we can find [20] who added an Error Model annex to AADL specifications, the modeling formalism for embedded real-time systems.

In the same way the HiP-HOPS workbench [21] enables the addition of reliability data to models imported from different modeling tools: Matlab/SIMULINK, Eclipse-based UML tools, etc., and then to automatically generate Fault Trees, FMEA tables, Temporal Fault Trees [22] and also to perform architecture optimization [23] and SIL allocation [24].

Similarly, translations have been defined from specialized UML or SysML models to Fault Trees or Petri nets (see e.g. [25] or [26]). In [27], functional design phase, using SysML, is combined with commonly used reliability techniques (i.e. FMEA and construction of AltaRica Data-Flow models).

In the second category, we can find Figaro [28], developed by EDF R&D. It is a textual modeling language dedicated to dependability assessment of complex systems. It combines object-orientation languages features, such as inheritance and first order production rules (interaction and occurrence rules). It is used as a description language to create knowledge bases for the workbench KB3 [29] to automatically perform systems dependability assessment: Monte-Carlo simulation, Markov Chain generation, quantification and generation of critical sequences, etc.

5. The case study

The case study comprises of a power supply system. We adopted the case study from [30]. The model is depicted as block diagram in Fig. 1. It features redundant supply lanes to avoid total system failure. In normal mode, the energy is provided from the grid via the first transformer (TR_1) and the first switch (SW_1). In case of a failure, the switch SW_2 is closed and the energy is provided via the second transformer (TR_2). If the grid or the second lane fails, the diesel engine (D) is started and the switch SW_3 is closed.

All components of the system may fail. The transformers are modeled with a per time failure. All three switches are modeled with an on-demand failure. This means that the switches can only fail to close in the instant moment where they are requested to close. Only the diesel engine is modeled with a failure rate and an on-demand failure. The failure rates and on-demand failure probabilities are denoted in Table 1. For all components, a repair rate of $10^{-1}1/h$ is specified.

The three lanes are used according to their priority, where the first lane (TR_1 and SW_1) has the highest and the third lane (D and SW_3) has the least priority. This means that whenever a component with a higher priority is repaired the corresponding lane is used immediately after repairing.

As the case study is from the safety analysis domain, we analyze the reachability of hazardous system states. A hazard in this case is if the system fails to provide power to the *Busbar*.

6. Modeling the system

6.1. SAML model

Our SAML model consists of 13 modules. Eight of them are dedicated to the error modes of the components. The transformers have no internal state, so that their complete behavior is already

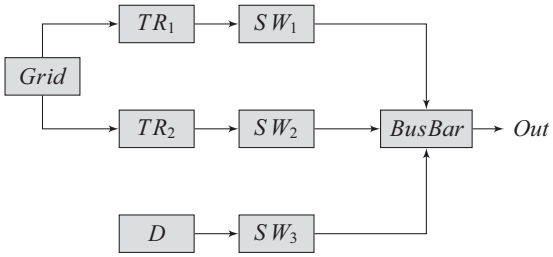

Fig. 1. Block diagram of the power supply system. The arcs denote the energy flow and the blocks denote system components.

Table 1
Specification of the component failures.

Component	Failure rate	On-demand probability
Grid	$10^{-4}1/h$	–
TR_1	$10^{-4}1/h$	–
TR_2	$10^{-4}1/h$	–
SW_1	–	10^{-3}
SW_2	–	10^{-3}
SW_3	–	10^{-3}
D	$10^{-4}1/h$	10^{-3}

covered by the failure modules. Four modules are reserved for the state of the three switches (i.e. open or closed) and the diesel engine (i.e. idle or running). The Busbar has no internal state, neither can it fail. This leaves one remaining module, which we explain later on. Besides the modules, our model heavily exploits formulas, denoting if a component is demanded or if certain lanes are in fail state. Constants are used for the failure and repair probabilities.

An extract of the complete SAML model is listed in Fig. 2. Note that due to page limitations we listed only a minimal summary that shows the principal of the model. The formulas state elementary propositions and are mainly used to increase the maintainability and readability of the model. As there are many similar propositions for the three lanes or redundant components only some of them are listed in Fig. 2.

Following the formulas, Fig. 2 lists three modules: *sw*1, trafo1_err and *sw*1_*demand_err*. These modules were chosen, because they all three are characteristic for different types of modules. The *sw*1 module represents functional behavior of the system. It has two states (representing opened and closed). It closes if *is_l*1_*demand* evaluates to true and if the switch does not suffer from on-demand error (*is_sw*1_*demand_fail*).

The module trafo_err represents a repairable (transient) per-time failure. It has two states which describe if the failure occurs or not. If in the operational state, it turns to failure state with a probability of f_trafo1. Otherwise it stays in the operational state. The behavior in the failed state is similar, but with a different probability.

The third module (*sw*1_*demand_err*) covers an on-demand failure automaton. This is slightly more complicated than the per-time failure. This remains from the following situation. The on-demand failure automaton is only allowed to change its state if the corresponding component is demanded. At the same time the component reacts on the demand. Due to parallel composition and discrete time modeling the component in question and the failure automaton change at exactly the same time. Thus the component can only react on the changing of the failure automaton one step after the demand. The solution is to move the failure automation exactly once some time before it is required. This requires the model to have a zeroed time step where the on-demand failure modules are initialized. That is why the on-demand failure module has three states (initial, failure and operational). A detailed explanation of failure modeling in SAML is provided in [5]. A more detailed discussion about the initial step can be found in [31].

To comply with the discrete time semantics of SAML we assigned a time of 1 min to every step ($\Delta t = 1$ min). This affects the conversion from failure rates in the system specification to per-step probabilities in the model.

The formula *is_no_power* denotes if the system fails to provide power to the BusBar. However it is not sufficient to analyze whether the system may reach such a state. Because SAML uses a discrete timed semantics, an information needs some time to propagate throw the component structure. For the presented case study, the redundant power lanes can only react after the system has failed. Thus a single powerless state is not considered a

```
component main

constant double f_trafo1 := 10E-12; // 1/min
constant double r_trafo1 := 6E-12; // 1/min
constant double f_sw1_demand := 10E-12; // 1/min
constant double r_sw1 := 6E-12; // 1/min
[..]

formula is_l1_demand := !is_l1_fail
formula is_sw1_demand :=
          (is_l1_demand & is_sw1_open);
formula is_trafo1_fail := trafo1_e = 1;
formula is_sw1_demand_fail := sw1_demand_e = 1;
formula is_l1_fail :=
          (is_grid_fail|is_sw1_open|is_trafo1_fail);
formula is_no_power :=
          (is_l1_fail & is_l2_fail & is_l3_fail);
[..]

component sw1
  enum SW_STATE := [OPEN, CLOSE];
  sw1_s : SW_STATE init CLOSE; // OPEN, CLOSE

  sw1_s=OPEN & (!is_sw1_demand | is_sw1_demand_fail) ->
      choice:(1:(sw1_s'= OPEN));
  sw1_s=OPEN & is_sw1_demand & !is_sw1_demand_fail ->
      choice:(1:(sw1_s'= CLOSE));
  sw1_s=CLOSE & !is_l1_demand -> choice:(1:(sw1_s'= OPEN));
  sw1_s=CLOSE & is_l1_demand -> choice:(1:(sw1_s'= CLOSE));
endcomponent

component trafo1_err
  enum ERR_STATE := [OK, FAIL];
  trafo1_e : ERR_STATE init OK; // OK, FAIL

  trafo1_e=OK -> choice:(f_trafo1:(trafo1_e'=FAIL) +
        (1-f_trafo1):(trafo1_e'=OK));
  trafo1_e=FAIL -> choice:(r_trafo1:(trafo1_e'=OK) +
        (1-r_trafo1):(trafo1_e'=FAIL));
endcomponent

component sw1_demand_err
  enum DEMAND_STATE := [OK, ERR, INI];
  sw1_demand_e : DEMAND_STATE init INI; // OK, ERR, INIT

  sw1_demand_e=INI -> f_sw1_demand:(s'=DE_E.ERR) +
      (1-f_sw1_demand):(s'=DE_E.OK);
  sw1_demand_e=OK & !is_sw1_demand ->
      choice:(1:(sw1_demand_e'=0));
  sw1_demand_e=OK & is_sw1_demand ->
      choice:( f_sw1_demand:(sw1_demand_e'=ERR) +
        (1-f_sw1_demand):(sw1_demand_e'=OK));
  sw1_demand_e=ERR -> choice:(r_sw1:(sw1_demand_e'=OK) +
        (1-r_sw1):(sw1_demand_e'=ERR));
endcomponent
[..]

endcomponent
```

Fig. 2. Extract of the SAML model.

hazardous state. To compensate this effect, a last module was introduced. This module counts the subsequent steps where the system fails to provide power. A failure occurs whenever the counter reaches two. The counter has the state variable obs_s. Therefore the hazard is defined as $H{:=}obs\_s = 2$.

6.2. AltaRica model

The power supply system is composed of 4 types of components: a grid, a transformer, a switch and a diesel engine. As discussed in Section 5, we shall consider the following failure modes:

- a grid and a transformer can only fail in operation (stochastic exponentially distributed event with a failure rate λ);
- a switch can fail on demand (with a probability γ) or be turned on successfully;
- a diesel engine can either fail in operation or on demand.

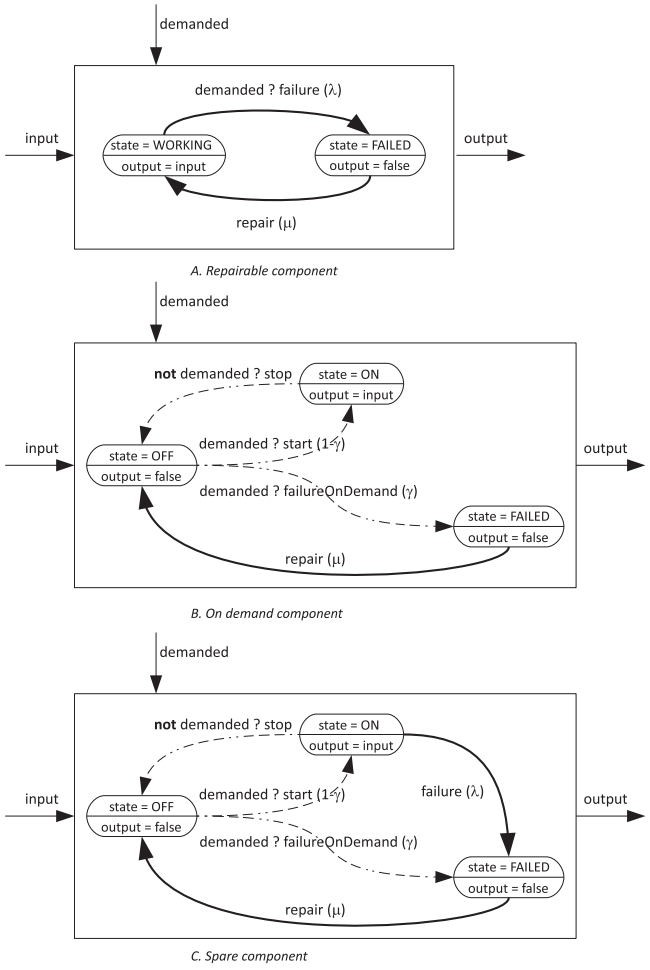

Fig. 3. Patterns of Guarded Transition Systems.

```
domain SpareComponentState { ON, OFF, FAIL }

class SpareComponent
  SpareComponentState state (init = OFF);
  Boolean demanded (reset = false);
  Boolean input (reset = false);
  Boolean output (reset = false);
  Boolean failed (reset = false);
  event start (delay = 0, expectation = 1 - gamma);
  event failureOnDemand (delay = 0,
          expectation = gamma);
  event failure (delay = exponential(lambda));
  event repair (delay = exponential(mu));
  event stop (delay = 0);
  parameter Real gamma = 10e-3;
  parameter Real lambda = 10e-4;
  parameter Real mu = 10e-1;
transition
  start:  state==OFF and demanded → state := ON;
  failureOnDemand:state==OFF and demanded →
          state := FAILED;
  failure:  state==ON → state := FAIL;
  repair:  state==FAIL → state := OFF;
  stop:  state==ON and not demanded → state := OFF;
assertion
  output := if state==ON then input else false;
  failed := (state==FAIL);
end
```

Fig. 4. The AltaRica code for the finite state automaton modeling a spare component.

The behavior of these components can be represented by different modeling patterns of Guarded Transition Systems, pictured in Fig. 3. The AltaRica code corresponding to the finite state machine of a spare component (representing a diesel engine) is given in Fig. 4.

6.2.1. States

The internal state of the `SpareComponent` is represented by means of the state variable `state`. `state` takes its values in the domain `SpareComponentState` declared upfront. The initial values of state variables are specified by means of the attribute `init`.

6.2.2. Events

The state of the component changes under the occurrence of an event. Events are introduced with the keyword `event`. A delay is associated with each event by means of the attribute `delay`. Delays of events `failure` and `repair` are random variables exponentially distributed with respective rates `lambda` and `mu`. Events `start` and `failureOnDemand` are instantaneous (their delay is 0). Both are fireable when the component is `OFF`. `start` has the probability `1 - gamma` to be fired while `failureOnDemand` has a probability `gamma` to be fired in this state. This probability is given through the attribute `expectation`. The expectation of the event e is used to determine the probability that the transition labeled with e is fired in case of several transitions are fireable at the same date. When transitions labeled with $e_1, e_2, \ldots, e_k$ are scheduled at the same date, the probability $p(e_i)$ to fire the transition labeled with e_i $(1 \le i \le k)$ is defined as follows:

$$p(e_i) = \frac{expectation(e_i)}{\sum_{1 \le j \le k} expectation(e_j)}$$

6.2.3. Transitions

A transition is a triple $\langle e, G, P \rangle$, also denoted $e : G \to P$, where e is an event, G is a Boolean expression, so-called the guard (or the pre-condition) of the transition, P is an instruction, so-called the action (or the post-condition) of the transition. Transitions are described in the clause `transition`. If the state of the component is `OFF`, then two transitions are fireable: the transition labeled with the event `start` and the transition labeled with the event `failureOnDemand`. These transitions are deterministic and instantaneous because they are associated with a delay 0. The transition labeled by `failureOnDemand` has the probability to be fired `gamma` and the transition labeled by `start` has the probability to be fired `1-gamma`. If the transition labeled by `failureOnDemand` is fired, then its action is executed: `state` is switched to `FAIL`. In Fig. 3 instantaneous transitions are marked with dashed lines. Transitions `failure` and `repair` are timed and stochastic. They obey typically exponential distributions. In Fig. 3 timed transitions are marked with plain lines. If the state of the component is `ON` and the delay drawn for the transition `failure` is the shortest, then this transition is fired.

6.2.4. Parameters

Parameters are constant values that come with the definition of the AltaRica class. When a class is instantiated, their values may be changed. In the model above, there are three parameters `gamma`, `lambda` and `mu` that define respectively the probability of failure on demand and the failure and repair rates.

6.2.5. Flow variables and assertions

Variables `demanded`, `input`, `output` are Boolean flow variables. The variable `demanded` is used to implement the command, i.e. to tell

```
class PowerSupplySystem
  RepairableComponent Grid, TR1, TR2;
  SpareComponent D;
  OnDemandComponent SW1(s.init = CLOSE);
  OnDemandComponent SW2, SW3;
  Boolean lane1_failed(reset = false);
  Boolean lane2_failed(reset = false);
  Boolean lane3_failed(reset = false);
  observer Boolean failed = lane1_failed and
           lane2_failed and lane3_failed;
assertion
  lane1_failed := Grid.failed or TR1.failed
           or SW1.failed;
  lane2_failed := Grid.failed or TR2.failed
           or SW2.failed;
  lane3_failed := D.failed or SW3.failed;
  Grid.demanded := not lane1_failed or
           not lane2_failed;
  TR1.demanded := not lane1_failed;
  SW1.demanded := TR1.demanded;
  TR2.demanded := lane1_failed and
           not lane2_failed;
  SW2.demanded := TR2.demanded;
  D.demanded := lane1_failed and lane2_failed
           and not lane2_failed;
  SW3.demanded := D.demanded;
  Grid.input := true;
  D.input := true;
  TR1.input := Grid.output;
  SW1.input := TR1.output;
  TR2.input := Grid.output;
  SW2.input := TR2.output;
  SW3.input := D.output;
end;
```

Fig. 5. The AltaRica code for the whole system.

Table 2
Hazard probability.

1 h	10 h	100 h	10 000 h
1.00e−9	1.36e−6	2.24e−5	2.47e−3

when to turn on and off the component. The variables `input` and `output` represent the flow circulating through the component, and in this case it is the electrical power. From a syntactic viewpoint, flow variables are introduced (and distinguished from state variables) by means of the attribute `reset`. Conversely to state variables, that are initialized at the beginning of a run and then modified through actions of transitions, the value of flow variables are recalculated after each transition firing. This recalculation is performed by means of assertions. Assertions are instructions just as actions of transitions. The difference stands in that actions of transitions assign state variables only while assertions assign flow variables only. Moreover, each component has a unique assertion that is applied after each transition firing.

Flow variables and assertions are used to model information flows circulating through a system. They may represent physical connections between components, control commands, fluid circulation, electric power, etc. They offer an easy and elegant way to express dependencies on external factors.

The AltaRica code for the `SpareComponent` and for the other patterns (`OnDemandComponent` and `RepairableComponent`) are quite similar.

6.2.6. Composition

Now we can consider the model for the whole power supply system. AltaRica 3.0 is an object-oriented modeling language. Therefore, the AltaRica class that describes the power supply system embeds an instance of the class `SpareComponent` describing the diesel engine `D`, three instances of the class `OnDemandComponent` representing the switches `SW1, SW2, SW3`, and three instances of the class `RepairableComponent` describing the grid `Grid` and the transformers `TR1, TR2`, as illustrated in Fig. 5.

When the lane 1 is failed, the switch of the second lane `SW2` is attempted to turn on. If it fails then the diesel generator `D` is attempted to start and the switch `SW3` is attempted to turn on. These rules are expressed in the assertion of the class `PowerSupplySystem`.

6.2.7. Observers

Observers are like flow variables, except that they cannot be used in transitions and assertions, i.e. they cannot be used to describe the behavior of a system. Rather, as their name indicates, they are quantities to be observed. They can be used or not by the assessment tools. Observers are updated after each transition firing.

In the AltaRica code of the class `PowerSupplySystem` we declared an observer `failed` that detects if the system is in the hazardous state (when all the lanes are failed and, therefore, the system cannot supply power to the *Busbar*).

7. Analyzing the system

7.1. SAML

We performed a DCCA on the SAML model, which lead to the following minimal cut-sets:

- $\Gamma_1 = \{grid\_e, sw3\_demand\_e\}$
- $\Gamma_2 = \{grid\_e, diesel\_demand\_e\}$
- $\Gamma_3 = \{grid\_e, diesel\_e\}$
- $\Gamma_4 = \{trafo1\_e, sw2\_demand\_e, sw3\_demand\_e\}$
- $\Gamma_5 = \{trafo1\_e, trafo2\_e, sw3\_demand\_e\}$
- $\Gamma_6 = \{trafo1\_e, diesel\_demand\_e, sw2\_demand\_e\}$
- $\Gamma_7 = \{trafo1\_e, diesel\_e, sw2\_demand\_e\}$
- $\Gamma_8 = \{trafo1\_e, trafo2\_e, diesel\_demand\_e\}$
- $\Gamma_9 = \{trafo1\_e, trafo2\_e, diesel\_e\}$

In addition to DCCA we use PRISM to perform a probabilistic deductive cause consequence analysis (pDCCA) [5]. To analyze the overall hazard probability we used the pCTL property $P_{max=?}[trueU\langle=nobs_s=2]$ where n denotes the number of steps to analyze. According to $\Delta t = 1$ min, 1 h leads to $n=60$, 2 h to $n=600$ and so forth. The results of the PRISM based analysis are listed in Table 2.

7.2. AltaRica

In order to perform different types of analyses, the AltaRica model given in Fig. 5 is first of all "flattened" into a unique Guarded Transition System (GTS). For the obtained GTS it is possible to generate a Reachability graph that can be explored in order to compute reliability indicators. In this paper we focus on two types of model analyses:

- the generation of critical sequences of events;
- the compilation into a Markov chain in order to compute the system unreliability $P[T\langle t]$, the probability that the system fails (the property *failed* = = *true* becomes verified) before the time t.

The generated critical sequences are:

1. G.failure SW3.start D.failureOnDemand
2. G.failure SW3.start D.start, D.failure
3. G.failure SW3.failureOnDemand
4. TR1.failure SW2.start TR2.failure SW3.start D.failureOnDemand
5. TR1.failure SW2.start TR2.failure SW3.start D.start D.failure
6. TR1.failure SW2.start TR2.failure SW3.failureOnDemand
7. TR1.failure SW2.failureOnDemand SW3.start D.failureOnDemand
8. TR1.failure SW2.failureOnDemand SW3.start D.start D.failure
9. TR1.failure SW2.failureOnDemand SW3.failureOnDemand

The generated Markov graph contains 72 states and 248 transitions. The obtained results are summarized in Table 3.

8. Comparison and evaluation

To compare safety modeling formalisms we have established several comparison criteria. They are discussed below. An overview of this comparison can be found in Table 4.

Event based: The goal of Safety Analyses is to determine the most probable failure scenarios, i.e. sequences of events leading from the nominal state to a failure/hazardous state. There are potentially different types of events: stochastic, instantaneous, timed deterministic, etc..

AltaRica 3.0 is an event based modeling language: it is possible to explicitly name events and associate them to transitions. Events can be stochastic (e.g. events *failure* and *repair* in the model Fig. 4) and instantaneous (e.g. events *failureOnDemand*, *start* and *stop* in the model Fig. 4).

In the sense of SAML, all continuous functions are sampled and processed in discrete time steps. Though, the subsequent state of a SAML model is solely based on the current state and the decision of non-deterministic and probabilistic choices. Events that occur in between two time steps are cumulatively processed when forming the next state.

Composition: Models of systems should be obtained by composing models of subsystems. States of the system should be given in a implicit way to avoid the user to enumerate all of them and to allow approximations, based on the most probable scenarios/states.

AltaRica 3.0 and SAML are both compositional and represent implicitly state graphs of modeled systems. A SAML model is composed of several components that are all executed in parallel. Recently, templates have been added to SAML: it allows us to create component instances based on a pattern. They not only greatly improve the reusability of models but allow the convenient modeling of equal components. In AltaRica 3.0 each system component is represented by a class; a system model is obtained by instantiating previously defined classes, connecting their inputs and outputs and synchronizing their events (see e.g. the model of the case study Fig. 5). Unlike SAML, in AltaRica only one transition can be fired at a time.

Hierarchy: Models of systems should be obtained by composing models of subsystems or different views of the system into hierarchies.

AltaRica 3.0 integrates notions of object-oriented programming languages such as inheritance and prototypes. It offers constructs to structure models into hierarchies of reusable components. An AltaRica 3.0 model can be seen as a hierarchy of interconnected components (see e.g. the model of the case study Fig. 5).

Recently, the notion of nested component has been added to SAML. A SAML model can be represented as a hierarchy of nested components.

Remote interactions: It should be possible to describe easily remote interactions between components, i.e. flows of matter or information circulating through the system (without enumerating them explicitly).

In AltaRica 3.0 remote interactions are represented by flow variables and assertions. In the example Fig. 5 the assertion calculates system flow variables. The principle is explained in Section 6.2. The assertion is recalculated after each transition firing. Remote interactions can be also expressed by synchronizations of events. Examples can be found in [32].

In SAML the remote interactions can be represented by means of shared variables. In general, every state variable in SAML is globally readable so the current state of one component is implicitly distributed to all others. To increase readability of the model, we use formulas to assign names to relevant (sets of) states. For example, in the model, given Fig. 2 the formulas are used for that purpose.

Graphical representation: Graphical representation of models has its own interest. One should be able to represent graphical models, at least partly, for communication and animation purposes.

It is not possible to have a unique graphical representation of an AltaRica 3.0 model. At least three different graphical views can be used to represent it:

1. Representations, like Process & Instrumentation Diagrams or Block Diagrams (see Fig. 1), can be used to capture the hierarchy, the connections between components and the circulating flows of AltaRica 3.0 models.
2. State diagrams, like those given Fig. 3, can represent the internal behavior of each AltaRica 3.0 class.
3. Thus events in AltaRica 3.0 can be synchronized, diagrams, like UML sequence diagrams, can be used to represent synchronizations.

Table 3
Unreliability.

1 h	10 h	100 h	10 000 h
2.06e−5	2.11e−4	2.12e−3	1.91e−1

Table 4
Comparison of safety formalisms: SAML and AltaRica 3.0.

Comparison criteria	SAML	AltaRica 3.0
Event based	No	Yes
Composition	Templates	Object-oriented
Hierarchy	Nested components	Object-oriented
Remote interactions	Shared variables	Flow variables and assertion, synchronizations
Graphical	State charts	State, sequence and block diagrams
Assessment tools	Model-checking, probabilistic model-checking, stochastic simulation	Compilation to Fault Trees and Markov graphs, stochastic and stepwise simulation
Time	Discrete	Triggered by events

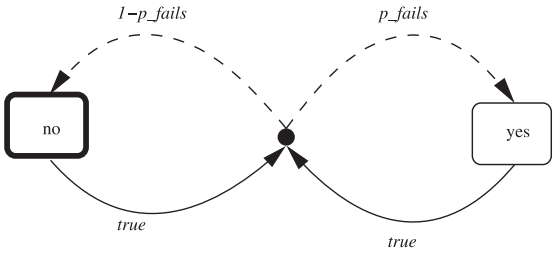

Fig. 6. Graphical representation of transient failure automaton.

Each of these graphical representations gives a partial view of an AltaRica 3.0 model.

SAML naturally maps to state charts. An example for a transient failure module is given in Fig. 6. The combination of non-deterministic and probabilistic choices leads to a slightly more extensive notation. Every probability distribution is denoted by dashed arrows leaving the same black dot. Possible non-deterministic choices can then be expressed by multiple arrows with the same guard and leaving the same state.

Available assessment tools: Prototypes of a set of assessment tools for the new version of AltaRica are currently developed. They include a Fault Tree compiler, a Markov chain generator, a stepwise and a stochastic simulators. These assessment tools will be distributed under a free licence.

At the current state, there exist three assessment tools for SAML. For one there are the NuSMV and Prism model checkers (described in Section 2). In addition the VECS[2] [33] tool contains a step by step simulator. Besides the existing tools, the framework is designed in a flexible way, so that additional tools can be integrated easily.

Interpretation of time: Time in AltaRica and SAML is interpreted differently. SAML is synchronous. It uses a discrete time model. The state of all automaton in the model is updated only at discrete time steps. The successive state solely depends on the current state.

The time in AltaRica is triggered by events. It is an intermediate model between discrete and continuous time. A delay is associated with each event. It can be deterministic or stochastic and may depend on the state. When the transition labeled with the event gets fireable at time t, a delay d is calculated, and the transition is actually fired at time $t+d$ if it stays fireable from t to $t+d$. The semantics of AltaRica model is a Kripke structure (a reachability graph) that can be interpreted as a continuous-time Markov chain, under the condition that delays associated with transitions are exponentially distributed.

9. Conclusion and outlook

In this paper we compared SAML and AltaRica. Both are formal modeling languages for Model-Based Safety Analysis. On the syntactical level the set of language constructs in SAML appeared to be smaller than the one in AltaRica. On the one hand this simplifies the models but on the other hand also makes it more difficult to express large and/or complex systems.

On the semantic level the two languages chose fundamentally different approaches. SAML uses a discrete time model with equidistant time steps. AltaRica is based on continuous time with discrete events. In practice this means that in SAML all automata perform exactly one transition at the same time. In AltaRica only one transition is fired at one time.

In future work we will evaluate the conversion between AltaRica and SAML. Even though not trivial, an automatic conversion between the two languages extends their set of available analysis tools. The main challenge for such a transformation is for sure the different time-model in the two languages.

For SAML we are currently busy with the evaluation of a data-flow based modeling approach like in AltaRica. The case-study we used in this paper mostly consists of data-flow, which was rather difficult to express in SAML. Nevertheless, the SAML language should remain as simple as possible.

References

[1] Andrews J, Moss T. Reliability and risk assessment. John Wiley & Sons; 1993 [ISBN 0-582-09615-4].
[2] AjmoneMarsan M, Balbo G, Conte G, Donatelli S, Franceschinis G. Modelling with generalized stochastic Petri Nets. Wiley series in parallel computing. John Wiley and Sons; 1994.
[3] Arnold A, Griffault A, Point G, Rauzy A. The AltaRica language and its semantics. Fundamenta Informaticae 2000;34:109–24.
[4] Boiteau M, Dutuit Y, Rauzy A, Signoret J-P. The AltaRica data-flow language in use: assessment of production availability of a multistates system. Reliability Engineering and System Safety 2006;91:747–55.
[5] Güdemann M, Ortmeier F. A framework for qualitative and quantitative model-based safety analysis. In: Proceedings of the 12th high assurance system engineering symposium (HASE 2010); 2010. p. 132–41.
[6] de Alfaro L, Faella M, Henzinger TA, Majumdar R, Stoelinga M. Model checking discounted temporal properties. Theor Comput Sci 2005;345:139–70.
[7] Cimatti A, Clarke E, Giunchiglia E, Giunchiglia F, Pistore M, Roveri M, et al. NuSMV version 2: an opensource tool for symbolic model checking. In: Proceedings of the 14th international conference on computer aided verification (CAV 2002). Lecture notes on computer science, vol. 2404. Springer; Copenhagen, Denmark, 2002.
[8] Kwiatkowska M, Norman G, Parker D. Probabilistic symbolic model checking with PRISM: a hybrid approach. In: Proceedings of the 8th international conference on tools and algorithms for the construction and analysis of systems (TACAS 2002). Lecture notes on computer science, vol. 2280. Springer; Grenoble, France, 2002.
[9] Katoen J-P, Khattri M, Zapreev I. A Markov reward model checker. In: Proceedings of the 2nd international conference on quantitative evaluation of systems (QEST 2005). IEEE Computer Society; Torino, Italy, 2005.
[10] Clarke E, Grumberg O, Peled D. Model checking. MIT Press; 2000.
[11] Güdemann M, Ortmeier F, Reif W. Computing ordered minimal critical sets. In: Schnieder E, Tarnai G, editors. Proceedings of the 7th symposium on formal methods for automation and safety in railway and automotive systems (FORMS/FORMAT 2008); 2008.
[12] Point G, Rauzy A. AltaRica: constraint automata as a description language. Journal Européen des Systémes Automatisés 1999;33(8–9):1033–52.
[13] Bernard R, Aubert J-J, Bieber P, Merlini C, Metge S. Experiments in model-based safety analysis: flight controls, In: Proceedings of IFAC workshop on dependable control of discrete systems. Cachan; 2007.
[14] Bernard R, Metge S, Pouzolz F, Bieber P, Griffault A, Zeitoun M. AltaRica refinement for heterogeneous granularity model analysis. In: Actes du congrés Lambda-Mu 16, Avignon; 2008.
[15] Rauzy A. Guarded transition systems: a new states/events formalism for reliability studies. J Risk Reliab 2008;222(4):495–505.
[16] Prosvirnova T, Rauzy A. Guarded transition systems: pivot modelling formalism for safety analysis. In: Barbet J, editor. Actes du congrés Lambda-Mu 18; 2012.
[17] Rauzy A. Modes automata and their compilation into fault trees. Reliab Eng Syst Saf 2002;78:1–12.
[18] Rauzy A. Anatomy of an efficient fault tree assessment engine. In: Virolainen R, editor. Proceedings of international joint conference PSAM'11/ESREL'2012; 2012.
[19] Perrot B, Prosvirnova T, Rauzy A, d'Izarn J-PS, Schoening R. Expériences de couplages de modéles AltaRica avec des interfaces métiers. In: Fadier E, editor. Actes du congrés Lambda-Mu 17 (actes électroniques), IMdR; 2010.
[20] Feiler P, Rugina A. Dependability modeling with the architecture analysis and design language (AADL). Technical report. Carnegie Mellon University; 2007.
[21] Pasquini A, Papadopoulos Y, McDermid J. Hierarchically performed hazard origin and propagation studies. In: Computer safety, reliability and security. Lecture notes in computer Science, vol. 1698; 1999. p. 688.
[22] Walker M, Papadopoulos Y. Qualitative temporal analysis: towards a full implementation of the fault tree handbook. Control Eng Pract 2009;17:1115–25.
[23] Papadopoulos Y, Walker M, Parker D, Rüde E, Hamann R, Uhlig, U., et al. Engineering failure analysis and design optimisation with HiP-HOPS. In: Fourth international conference on engineering failure analysis, part 1, vol. 18(2); 2011. p. 590–608.
[24] Papadopoulos Y, Walker M, Reiser M-O, Weber M, Chen D, Törngren M, et al. Automatic allocation of safety integrity levels. In: Proceedings of the 1st workshop on critical automotive applications: robustness and safety (CARS'10). New York, USA: ACM; 2010. p. 7–10.

[2] The tool was formerly named S3E.

[25] Xiang J, Yanoo K, Maeno Y, Tadano K. Automatic synthesis of static fault trees from system models. In: Conference on secure software integration and reliability improvement; 2011. p. 127-136.
[26] Bernardi S, Donatelli S, Merseguer J. From UML sequence diagrams and statecharts to analyzable petri net models. In: Proceedings of the third international workshop on software on performance; 2002.
[27] David P, Idasiak V, Kratz F. Reliability study of complex physical systems using SysML. Reliab Eng Syst Saf 2010:431–50.
[28] Bouissou M, Bouhadana H, Bannelier M, Villatte N. Knowledge modelling and reliability processing: presentation of the figaro modelling language and associated tools. In: Proceedings of Safecomp'91; 1991.
[29] Bouissou M. Automated dependability analysis of complex systems with the KB3 workbench: the experience of EDF R&D. In: Proceedings of the international conference on energy and environment; 2005.
[30] Bouissou M, Bon J-L. A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. Reliab Eng Syst Saf 2003;82(2):149–63. http://dx.doi.org/10.1016/S0951-8320(03)00143-1 URL ⟨http://www.sciencedirect.com/science/article/B6V4T-49DFH1M-1/2/bd15510dc655e0bbc55f3e5758bdeb42⟩.
[31] Güdemann M. Qualitative and quantitative formal model-based safety analysis [Ph.D. thesis]. Otto-von-Guericke-Universitä Magdeburg, Germany; 2011. URL ⟨http://nbn-resolving.de/urn:nbn:de:gbv:ma9:1-385⟩.
[32] Kloul L, Prosvirnova T, Rauzy A. Modeling systems with mobile components: a comparison between AltaRica and PEPA nets. J Risk Reliab 2013;227(6):599–613.
[33] Lipaczewski M, Struck S, Ortmeier F, SAML goes eclipse-combining model-based safety analysis and high-level editor support. In: Proceedings of the 2nd international workshop on developing tools as plug-ins (TOPI). IEEE;Zurich, Switzerland, 2012. p. 67–72.